М.З. ЗГУРОВСЬКИЙ
Ю.П. ЗАЙЧЕНКО

СИСТЕМИ І МЕТОДИ ШТУЧНОГО ІНТЕЛЕКТУ

НАЦІОНАЛЬНА АКАДЕМІЯ НАУК УКРАЇНИ
НАВЧАЛЬНО-НАУКОВИЙ КОМПЛЕКС
«ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ»
КПІ ім. ІГОРЯ СІКОРСЬКОГО

М.З. ЗГУРОВСЬКИЙ, Ю.П. ЗАЙЧЕНКО

СИСТЕМИ І МЕТОДИ ШТУЧНОГО ІНТЕЛЕКТУ

КИЇВ
АКАДЕМПЕРІОДИКА
2025

https://doi.org/10.15407/akademperiodyka.551.744
УДК 519.8(075)
З-19

Рецензенти:

О.М. ХІМІЧ, д-р фіз.-мат. наук, проф., академік НАН України

Є.В. БОДЯНСЬКИЙ, д-р техн. наук, проф. кафедри штучного інтелекту Харківського університету радіоелектроніки

М.М. МАЛЯР, д-р техн. наук, проф., декан факультету математики та цифрових технологій Ужгородського національного університету

Відповідальний редактор: В.Я. Данилов, д-р техн. наук, проф. кафедри штучного інтелекту ННК ІПСА КПІ ім. Ігоря Сікорського

Рекомендовано до друку
Вченою радою навчально-наукового комплексу «Інститут прикладного системного аналізу» КПІ ім. Ігоря Сікорського (протокол від 17.06.2024 р. № 6)

Видання здійснено за кошти Цільової комплексної програми НАН України «Наукові основи функціонування та забезпечення умов розвитку науково-видавничого комплексу НАН України»

Згуровський М.З.

З-19 Системи і методи штучного інтелекту / М.З. Згуровський, Ю.П. Зайченко; НАН України, ННК ІПСА КПІ ім. Ігоря Сікорського. — Київ: Академперіодика, 2025. — 744 с.

ISBN 978-966-360-551-7

Підручник надає систематичний огляд сучасних методів і технологій штучного та обчислювального інтелекту з акцентом на їх практичне застосування у різних сферах людської діяльності. Окреслено особливості кожного з напрямів, відмінності у підходах до вирішення задач. Особливу увагу приділено детальному аналізу нейронних мереж, зокрема їх архітектури, алгоритмів навчання та ключових теоретичних аспектів, підкріплених прикладами практичного застосування. Висвітлено методи нечіткої логіки та нейронні мережі, побудовані на їх основі, що ефективно використовують для прогнозування та аналізу поведінки складних систем. Розглянуто досягнення в обробці природної мови та еволюцію цього напряму від 1960-х років до сьогодення.

Для широкого кола студентів, аспірантів і фахівців, що вивчають та впроваджують штучний інтелект у своїй професійній діяльності.

УДК 519.8(075)

© НАН України, ННК ІПСА КПІ ім. Ігоря Сікорського, 2025
© Академперіодика, оформлення, 2025

ISBN 978-966-360-551-7

Перелік скорочень

2D CNN	Two-Dimensional Convolutional Neural Network
3D CNN	Three-Dimensional Convolutional Neural Network
ANFIS	Adaptive Network-Based Fuzzy Inference System
Adam	Adaptive Moment Estimation
BERT	Bidirectional Encoder Representations from Transformers
BLEU	Bilingual Evaluation Understudy
BLOOM	BigScience Large Open-science Open-access Multilingual
BPTT	Backpropagation Through Time
BiLSTM	Bidirectional Long Short-Term Memory
BiRNN	Bidirectional Recurrent Neural Network
BoW	Bag of Words
BreakHis	Breast Cancer Histopathological Image Classification
CBOW	Continuous Bag of Words
CNN	Convolutional Neural Networks
CPU	Central Processing Unit
CRF	Conditional Random Fields
CWTA	Алгоритм Кохонена «Conscientious Winner Takes All»
DL	Deep Learning
DNN	Deep Neural Networks
Doc2Vec	Document to Vector
ELMo	Embeddings from Language Models
FAT	Fat Approximation Theorem
FCM	Нечіткий Метод *k*-середніх «Fuzzy C-Means»
FastText	Fast Text Processing
GPS	General Problem Solver
GPT	Generative Pre-Trained Transformer
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
GloVe	Global Vectors for Word Representation
HMM	Hidden Marcov Models
IMDB	Internet Movie Database
LDA	Latent Dirichlet Allocation

LLaMA 3	Large Language Model Meta AI 3
LRS	Label Switching Routers
LSA	Latent Semantic Analysis
LSTM	Long Short-Term Memory
LaMDA	Language Model for Dialogue Applications
MAPE	Mean Absolute Percentage Error
MIL	Multiple Instance Learning
ML	Machine Learning
MLM	Masked Language Modeling
MLP	Багатошаровий Перцептрон
MPLS	Multiprotocol Label Switching
MR	Movie Reviews
MSE	Mean Squared Error
NEFClass	Нечітка нейронна Мережа для Класифікації «Neuro-Fuzzy Classification»
NER	Named Entity Recognition
NFS	Нейро-Нечітка Система
NLG	Natural Language Generation
NLP	Natural Language Processing
NLTK	Natural Language Toolkit бібліотека
NLU	Natural Language Understanding
NMT	Neural Machine Translation
NSP	Next Sentence Prediction
OSINT	Open Source Intelligence
PCA	Principal Component Analysis
POS-tagging	Part-of-Speech Tagging
PSO	Particle Swarm Optimization
QA	Question Answering
RMSprop	Root Mean Square Propagation
RNN	Recurrent Neural Network
ROI	Region of Interest
ReLU	Rectified Linear Unit
S2S	Sequence-to-Sequence
SACO	Simple Ant Colony Optimization
SGD	Stochastic Gradient Descent
SOM	Self-Organizing Maps
SST	Stanford Sentiment Treebank
SVD	Singular Value Decomposition
SVM	Support Vector Machines
Scikit-learn	Science Kit Learn бібліотека
Seq2Seq	Sequence-to-Sequence Model

Sklearn	Scikit-learn бібліотека
TF-IDF	Term Frequency-Inverse Document Frequency
TSK	Нечітка Нейронна Мережа «Takagi — Sugeno — Kang»
TSP	Travelling Salesman Problem
ULMFiT	Universal Language Model Fine-tuning
VAE	Variational Autoencoders
WMT'14	Workshop on Machine Translation 2014
WTA	Алгоритм Кохонена «Winner Takes All»
WTM	Алгоритм Кохонена «Winner Takes Most»
Word2Vec	Word to Vector
spaCy	Space Cyber бібліотека

АА	Автоасоціатор
АК	Автокодувальник
АРО	Алгоритм РО
БКО	Багатокритеріальна Оптимізація
ГА	Генетичні Алгоритми
ГММ	Геометричне Місце Мінімумів
ГРО	Глобальна Ройова Оптимізація
ГСОІ	Гібридна Система Обчислювального Інтелекту
ДЕ	Диференціальна Еволюція
ЕА	Еволюційні Алгоритми
ЕМ	Еволюційне Моделювання
ЕОМ	Електронна Обчислювальна Машина
ЕП	Еволюційне Програмування
ЗНМ	Згорткова Нейронна Мережа
КД	Контрастна Дивергенція
ЛН	Лінгвістична Змінна
ЛПМ	Логічно-Лінійне Поле Маркова
МБ	Машина Больцмана
МГУА	Метод Групового Урахування Аргументів
МНК	Метод Найменших Квадратів
МО	М'які Обчислення
НБП	Нечітка База Правил
НК	Нечіткий Контролер
НЛВ	Система Нечіткого Логічного Висновку
НМ	Нейронні Мережі
НМ ВР	Нейронні Мережі Back Propagation
НМГУА	Нечіткий МГУА
ННМ	Нечіткі Нейронні Мережі
НЧМ	Нечітка Множина (Fuzzy Set)

ОІ	Обчислювальний Інтелект
ОМБ	Обмежена Машина Больцмана
ОПР	Особа, що Приймає Рішення
ПК	Помилкові Класифікації
РБФ	Радіально-Базисні Функції
РМНК	Рекурентний Метод Найменших Квадратів
РО	Ройова Оптимізація
САК	Стековий Автокодувальник
САПП (MAPE)	Середня Абсолютна Процентна Похибка
СГ	Метод Сполучених Градієнтів
СКП	Середньоквадратична Похибка
СНЛ	Системи Нечіткої Логіки
СОК	Самоорганізована Карта Ознак
Ф-І	Фітнес-Індекс
Ф-Ф	Фітнес-Функція
ФН	Функція Належності
ШІ	Штучний Інтелект
ШНМ	Штучні Нейронні Мережі

Передмова

Сучасне суспільство та науково-технічний прогрес людства потребують застосування штучного інтелекту для підвищення ефективності, автоматизації та інтелектуалізації процесів у всіх сферах життя — від економіки, охорони здоров'я та освіти до екології, безпеки і міжособистісної комунікації. Це спрямовано на створення сталого, інноваційного та комфортного середовища, що відповідає викликам сучасності та майбутнього.

Пропонований підручник присвячено штучному інтелекту і підготовлено на основі результатів наукових досліджень та викладацької діяльності авторів, зокрема матеріалів монографій M. Zgurovsky, Yu. Zaychenko. Fundamentals of Computational Intelligence: System Approach. Springer, 2016. 275 p.; M. Zgurovsky, Yu. Zaychenko. Big Data: Conceptual Analysis and Applications. Springer Nature, Switzerland AG, 2019. 275 p.

За час, що минув з моменту публікації монографій, автори отримали нові результати у таких важливих напрямах розвитку штучного інтелекту, як нейронні мережі глибокого навчання (*Deep Learning*), гібридні мережі і згорткові нейронні мережі й їх застосування для завдань комп'ютерного зору, медичної діагностики, обробки природної мови (*Natural Language Processing, NLP*), що знайшло своє відображення в цьому підручнику.

Структура підручника складається зі вступу, 16 розділів та списку літературних джерел. Він висвітлює основні сучасні напрями розвитку штучного і обчислювального інтелекту.

У вступі надано визначення штучного та обчислювального інтелекту як сучасних засобів удосконалення і підвищення ефективності людської діяльності. Обидва напрями мають спільну мету — створення інтелектуальних систем, проте різняться у підходах і методах. Визначено, що головна концептуальна відмінність між штучним (ШІ) та обчислювальним інтелектом (ОІ) полягає в орієнтуванні ШІ на моделювання мислення і поведінки людини, тоді як ОІ спрямований на вирішення обчислювальних та оптимізаційних завдань за допомогою алгоритмів і методологій, не обов'язково пов'язаних з людським інтелектом. Ця відмінність відображає різницю в цілях і методах ШІ та ОІ. Виділено їх головні складові: технології, методи, ал-

горитми, прикладні задачі; наведено їхні характеристики та взаємозв'язки у разі застосування в складних системах.

Розглянуто сутність м'яких обчислень (*Soft Computing*) як сукупності методів і підходів, орієнтованих на вирішення задач з допуском до неоднозначності, неточності і неповноти даних, на відміну від жорстких (*Hard Computing*), що потребують чітко визначених алгоритмів і точних даних. Показано зв'язки і взаємодію м'яких обчислень зі штучним та обчислювальним інтелектом.

Перший розділ присвячено детальному опису нейронних мереж як однієї з найрозвинутіших технологій ШІ. Розглянуто нейронну мережу *Back Propagation*, її архітектуру та властивості. Сформульовано теорему про універсальну апроксимацію для цієї нейронної мережі, яка обґрунтовує універсальне застосування такого класу мереж для завдань прогнозування, апроксимації та класифікації.

Описано основні алгоритми навчання нейронних мереж: градієнтного спуску, спряжених градієнтів та генетичний алгоритм, а також їхні властивості й недоліки. Наведено архітектуру нейронних мереж з радіально-базисними функціями (РБФ) та основи теорії побудови таких мереж, гібридний алгоритм їх навчання і порівняння з класичним алгоритмом градієнтного спуску. Розглянуто сфери перспективного застосування цього алгоритму, а також складніший варіант мереж з гіперрадіально-базисними функціями.

Проведено порівняльний аналіз нейронних мереж *Back Propagation* та РБФ-мереж. Наведено приклад застосування мережі РБФ у задачі апроксимації функцій та сучасні методи визначення оптимальної кількості нейронів.

Другий розділ присвячено рекурентним нейронним мережам (РНМ) та нейронним мережам самоорганізації Кохонена (Теуво Кохонен (Teuvo Kohonen)). Розглянуто РНМ Хопфілда (Джон Хопфілд (John Hopfield)), введено поняття атрактора та його області притягання. Наведено алгоритм асинхронної корекції роботи РНМ Хопфілда та критерії вибору атракторів. Продемонстровано застосування РНМ Хопфілда для розпізнавання літер за різного рівня шуму, проаналізовано властивості та обмеження таких мереж, що звужують їх практичне застосування. Як альтернативу мережам Хопфілда розглянуто мережі Хемінга (Річард Вільям Хеммінг (Richard Wesley Hamming)), позбавлені недоліків РНМ Хопфілда. Наведено їх структуру та алгоритм навчання, а також проведено порівняльний аналіз нейронних мереж Хопфілда і Хемінга на прикладі класифікації літер латинського алфавіту за умови підвищення рівня шуму, що продемонструвало переваги мереж Хемінга.

Сформульовано основну мету змагального навчання за Кохоненом, а також наведено архітектуру мережі Кохонена. Розглянуто базовий алгоритм змагального навчання Кохонена *WTA* (*Winner Takes A*ll — «пере-

можець отримує все»), описано його властивості та недоліки. Показано, що цей алгоритм не повною мірою досягає мети змагального навчання, тому запропоновано його модифікацію — алгоритм з пам'яттю *CWTA* (*Conscientious Winner Takes All* — «сумлінний переможець забирає все»).

Описано модифікований алгоритм самоорганізації Кохонена з функцією сусідства *WTM* (*Winner Takes Most*). Враховано, що алгоритм *WTM* є варіацією алгоритму навчання в штучних нейронних мережах, особливо в контексті конкурентного навчання. Показано, що основна ідея цього алгоритму полягає в тому, що, на відміну від *WTA*, де лише один нейрон (той, що виграв) оновлює свої ваги, в алгоритмі *WTM* оновлюють свої ваги кілька нейронів із найвищими активаціями. Це дає можливість мережі бути гнучкішою та адаптивнішою до вхідних даних.

Розглянуто також алгоритм нейронного газу (*Neural Gas*), як один з методів кластеризації та навчання без нагляду у штучному інтелекті. Метою цього алгоритму є оптимальний розподіл нейронів у просторі вхідних даних для мінімізації похибки квантування. Показано, що алгоритм нейронного газу забезпечує плавну адаптацію нейронів до вхідних даних.

Наведено основні застосування нейронних мереж із самоорганізацією: завдання автоматичної класифікації об'єктів за ознаками та завдання візуалізації багатовимірних об'єктів на основі використання самоорганізаційних карт Кохонена (СОК). Це вид нейронних мереж, які розробив Тейво Кохонен, і вони відомі як *Self-Organizing Maps (SOM)* англійською. СОК використовують для кластеризації та візуалізації багатовимірних даних, організовуючи їх у двовимірну карту, що дає можливість виявляти структури та взаємозв'язки між елементами вхідних даних. Детально проаналізовано різні алгоритми побудови СОК та їхні властивості. Наприкінці розділу наведено приклад застосування різних алгоритмів самоорганізації в завданнях відновлення двовимірного розподілу у вибірці даних.

Третій розділ присвячено опису систем нечіткої логіки та нечітких нейронних мереж (ННМ). Розглянуто лінгвістичні змінні нечітких множин, які використовують для опису нечіткої та якісної інформації. Описано основні алгоритми нечіткого логічного висновку — Мамдані, Цукамото, Сугено і Ларсена (Ехсан Мамдані (Ebrahim H. Mamdani), Кацухіко Цукамото (Katsuhiko Tsukamoto), Мічіо Сугено (Michio Sugeno), Гуннара Ларсена (Gunnar Larsen)) — та проаналізовано їхні властивості. Сформульовано важливу теорему, яка обґрунтовує універсальні властивості систем з нечіткою логікою — *FAT (Fat Approximation Theorem)*, що є аналогом теореми про універсальну апроксимацію для нейронної мережі *BP (Back Propagation* — зворотне поширення помилки). *BP* — це метод навчання багатошарових нейронних мереж, який коригує ваги мережі через обчислення та зворотне поширення градієнта помилки від виходу до входу мережі. Проаналізовано

основні властивості й недоліки систем нечіткої логіки (СНЛ) та засіб їх усунення за допомогою інтеграції звичайних нейронних мереж та СНЛ.

Розглянуто відомі класи нечітких нейронних мереж, що об'єднують звичайні нейронні мережі та СНЛ. Описано нейронну мережу на основі висновків Мамдані та Цукамото і алгоритм її навчання. Далі розглянуто структуру та алгоритм навчання відомої мережі *ANFIS (Adaptive Network-Based Fuzzy Inference System)*. Проаналізовано нечітку нейронну мережу Такагі — Сугено — Канга (Takagi — Sugeno — Kang — *TSK*) (Такеші Такагі (Takeshi Takagi), Мічіо Сугено (Michio Sugeno), Ген Канга (Gen Kang)), наведено її структуру та гібридний алгоритм навчання, що суттєво скорочує обчислювальні витрати порівняно зі стохастичним градієнтним спуском. *TSK* — це специфічний тип нечіткої нейронної мережі. Модель *TSK* використовують для побудови правил нечіткого логічного висновку, де вихід кожного правила подано лінійною або нелінійною функцією вхідних змінних. Завдяки цій структурі *TSK*-моделі можуть ефективно апроксимувати складні нелінійні функції, що робить їх популярними для завдань моделювання та прогнозування. Показано, що *TSK* є одним з типів нейронечітких систем, які об'єднують нейронні мережі і нечітку логіку, а також що *TSK*-моделі широко використовуються завдяки здатності апроксимувати складні нелінійні функції. Проведено порівняльний аналіз нечітких нейронних мереж (ННМ) *ANFIS* та *TSK* у задачах прогнозування на ринку цінних паперів. Відзначено основний недолік ННМ *TSK*, який полягає у потребі навчання як вхідних функцій належності нечітких множин, так і вихідних ваг правил, що потребує значних обчислювальних витрат і часу.

Як альтернативу ННМ *TSK* розглянуто неонечіткий нейрон та каскадну неонечітку нейронну мережу на його основі, де потрібно навчати лише виходи правил, що суттєво скорочує обчислювальні витрати і час навчання. Ефективність неонечіткої нейронної мережі досліджено на прикладі задачі прогнозування індексу на ринку цінних паперів. У результаті знайдено оптимальні параметри такої мережі: кількість каскадів, кількість входів та фазифікаторів.

Розділ 4 присвячено методу групового врахування аргументів (МГУА), розробленому академіком Олексієм Григоровичем Івахненком у 1960-х рр. Цей метод є одним із перших у світі підходів до індуктивного моделювання та автоматизованої побудови моделей складних систем на основі експериментальних даних, що стало підґрунтям для розвитку сучасного машинного навчання. Розглянуто основні принципи МГУА: принцип множинності моделей, принцип самоорганізації, принцип зовнішнього доповнення та свободи вибору. Описано поліноміальний алгоритм МГУА та проаналізовано його властивості. Підкреслено, що цей метод, на відміну від інших, зокрема регресійних, вирішує завдання структурної ідентифікації та дає змогу синтезувати оптимальну структуру нейронної мережі.

Класичний поліноміальний алгоритм МГУА має також певні недоліки. Для визначення коефіцієнтів моделі він використовує метод найменших квадратів (МНК), що призводить до нестійких коефіцієнтів і високої волатильності моделі у разі, якщо матриця коефіцієнтів нормальних рівнянь Гаусса (Карл Фрідріх Гаусс (Carl Friedrich Gauss)) вироджена. Крім того, класичний МГУА не може працювати з неповною або нечіткою вхідною інформацією. У підручнику розглядається нечіткий МГУА, вільний від зазначених недоліків.

Описано основи теорії нечітких МГУА (НМГУА). Наведено інтервальну модель регресії, сформульовано вимоги до її адекватності та побудовано основну математичну модель для визначення коефіцієнтів часткових описів нечіткої поліноміальної моделі, використовуючи задачі лінійного програмування. Розглянуто алгоритм нечіткого МГУА, що розвиває класичний алгоритм для нечітких моделей. Визначено основні властивості НМГУА і проведено порівняльний аналіз з класичним МГУА. Розглянуто метод МГУА з ортогональними поліномами як частковими описами, з акцентом на переваги ортогональних поліномів над поліноміальними моделями: можливість підвищення ступеня полінома без перегляду коефіцієнтів при нижчих ступенях сприяє ефективній оптимізації самого ступеня моделі. Проаналізовано моделі НМГУА з ортогональними поліномами Чебишева (Пафнутій Львович Чебишев (Pafnuty Lvovich Chebyshev)) як частковими описами.

Розглянуто також НМГУА з тригонометричними поліномами, їхні властивості та можливість адаптації моделей МГУА. Запропоновано алгоритми адаптації коефіцієнтів моделей НМГУА на основі стохастичної апроксимації та рекурентного МНК. Проведено експериментальні дослідження МГУА та НМГУА з різними частковими описами у задачах прогнозування курсів акцій та індексу Доу-Джонса (Чарльз Генрі Доу (Charles Henry Dow)) на фондовому ринку. Знайдено оптимальні типи часткових описів для цих задач та виконано порівняльний аналіз методів МГУА і нечіткого МГУА для прогнозування макроекономічних показників України.

Розділ 5 присвячено аналізу нечітких нейронних мереж у завданнях класифікації, які широко застосовуються у техніці, економіці, соціології, медицині тощо. Розглянуто базову нечітку нейронну мережу для класифікації — *NEFClass (Neuro-Fuzzy Classification)*, зокрема алгоритм навчання бази правил і функції належності нечітких множин. Показано, що *NEFClass* — це модель нечіткої нейронної мережі, яка поєднує можливості нейронних мереж і нечіткої логіки для вирішення завдань класифікації. *NEFClass* використовує нечіткі правила для навчання та класифікації, що дає можливість обробляти невизначеність у вхідних даних.

Окремо розглянуто завдання розпізнавання об'єктів електрооптичних зображень (ЕОЗ). Наведено рішення цього завдання з використанням моделі *NEFClass-M (Neuro-Fuzzy Classification Model with Modified Learning*

Algorithm), що є розширенням *NEFClass* із модифікованим алгоритмом навчання для підвищення точності класифікації. Основні особливості *NEFClass-M* полягають у вдосконаленні методів навчання і адаптації правил, що сприяє ефективнішому обробленню складних та нечітких даних. Проведено порівняльний аналіз різних алгоритмів навчання *NEFClass-M* у завданні розпізнавання об'єктів ЕОЗ.

Шостий розділ присвячено аналізу методів кластеризації — автоматичної класифікації об'єктів за набором їхніх ознак. Наведено постановку задачі кластерного аналізу, обговорено основні критерії якості кластеризації та відповідні метрики.

Подано класифікацію методів кластерного аналізу: чіткі, нечіткі, ієрархічні та неієрархічні. Розглянуто приклади ієрархічних методів, зокрема чіткий метод кластеризації *k*-середніх із прикладом його застосування. Основну увагу зосереджено на аналізі нечітких методів кластерного аналізу.

Описано нечіткий метод *k*-середніх (*FCM, Fuzzy C-Means*), включно з його моделлю і алгоритмом. Показано, що цей метод дає можливість віднести кожний об'єкт до кількох кластерів із різними ступенями належності, що є розвитком класичного методу *k*-середніх і особливо корисним для роботи з нечіткими даними. Розглянуто вплив вибору початкового розміщення центрів кластерів на результати, а також методи локалізації цих центрів. Проаналізовано обчислювальні витрати різних підходів.

Для задач із різноманітними масштабами вимірювання ознак запропоновано модифікацію нечіткого методу *k*-середніх, а саме метод Густавсона — Кесселя (Девід Густавсон (David E. Gustafson) та Вільям Кессел (William C. Kessel)), який використовує матрицю масштабування для коректнішої оцінки відстаней. Наведено приклади практичного застосування нечітких методів кластеризації у задачах автоматичної класифікації, аналіз результатів для різної кількості кластерів та оцінку ефективності застосованих методів.

Далі розглянуто можливісні алгоритми кластеризації (*Possibilistic clustering Algorithms*), які знімають обмеження на суму ступенів належності кожної точки до кластерів. Це дає змогу вирішувати задачі з шумовими викидами та забезпечує робастність алгоритмів. Розглянуто роботу таких алгоритмів у задачах адаптивної кластеризації в режимі *online* і проведено порівняння з класичним алгоритмом нечіткої кластеризації.

Сьомий розділ присвячено застосуванню генетичних алгоритмів (ГА) та еволюційного програмування (ЕП) у завданнях штучного інтелекту.

Наведено основи теорії генетичних алгоритмів і проаналізовано основні оператори: кросовер, мутацію та селекцію, а також способи подання індивідів (бінарне та з точкою рухомою комою). Описано реалізацію кросовера для обох представлень, наведено порівняльний аналіз. Далі розглянуто оператори мутації, їхні властивості та основні методи кросовера. Наведено

класифікацію основних типів ГА: детерміністичні, адаптивні та самопристосовні алгоритми, а також проаналізовано їхні властивості та можливості застосування. Описано різні типи оператора селекції та їхні особливості.

Розглянуто острівні генетичні алгоритми, їхні властивості та варіанти реалізації операторів ГА. Описано приклади застосування ГА у задачах штучного інтелекту, зокрема у синтезі структури розподілених комп'ютерних мереж з використанням технології *MPLS (Multiprotocol Label Switching)*, орієнтуючись на мінімізацію вартості за умови дотримання вимог до якості передання. Показано, що *MPLS* дає змогу ефективно передавати дані через мережу, використовуючи маркування (*labeling*) пакетів.

Докладно розглянуто еволюційні методи у завданнях еволюційного програмування (ЕП) як складову штучного інтелекту. Описано основні оператори мутації та селекції ЕП. Наведено приклади реалізації цих операторів, проаналізовано їхні властивості. Розглянуто стратегічні параметри ЕП та методи їх налаштування, класифікацію основних типів алгоритмів ЕП, їхні властивості та галузі ефективного застосування.

Розглянуто цікавий напрям в еволюційному програмуванні — диференційну еволюцію (ДЕ), в якій крім базових операторів мутації та селекції використовується також третій оператор генетичного алгоритму (ГА) — кросовер. Зазначено, що метод диференційної еволюції має значно ширші можливості порівняно з класичним ЕП і є ефективнішим для вирішення практичних задач. Він застосовується для розв'язання глобальних оптимізаційних завдань і ґрунтується на принципах природного добору та генетичних операцій, подібно до інших еволюційних алгоритмів, проте має свої особливості та переваги. Наведено приклад реалізації диференційної еволюції.

У розділі також розглянуто приклад застосування ГА у задачі пошуку оптимальних шляхів на графі, проаналізовано отримані результати та здійснено порівняння цього алгоритму з класичним алгоритмом Дейкстри (Едсгер Вібе Дейкстро (Edsger Wybe Dijkstra)) за точністю та кількістю ітерацій на графах із різною кількістю вузлів. Обидва алгоритми починали роботу з вихідної точки (вузла) і поступово знаходили найкоротші відстані до всіх інших вузлів графа, використовуючи «жадібний» підхід. Було показано, що, хоча класичний алгоритм Дейкстри є ефективним і надійним для задач пошуку найкоротшого шляху, його обчислювальна складність за великої кількості вузлів і ребер значно вища порівняно з ГА.

Восьмий розділ присвячено ройовим алгоритмам обчислювального інтелекту. Розробка цих алгоритмів була натхнена спостереженнями за соціальною поведінкою тварин. Рій — це група тварин або комах, які діють окремо за простими алгоритмами, але в колективі у процесі взаємодії демонструють соціальну поведінку і виявляють колективний інтелект. Наприклад, вони колективно будують житло, знаходять поживу, ховають

мертвих особин тощо, без спеціального координатора, керуючись взаємодією між собою.

На основі спостережень за поведінкою комах і птахів було розроблено моделі та алгоритми ройової оптимізації, як глобальної, так і локальної. В алгоритмах глобальної оптимізації всі учасники (частинки) рухаються в напрямі найкращої позиції, знайденої всіма учасниками рою, та найкращої позиції, знайденої окремим учасником. У локальній ройовій оптимізації використовується найкраща позиція серед групи сусідів, а не всього рою.

Розглянуто приклади завдань, у яких рух частинок рою застосовується для пошуку глобального мінімуму функцій в алгоритмах глобальної та локальної ройової оптимізації. Проаналізовано проблему вибору оптимальних параметрів алгоритмів ройової оптимізації (РО), а також розглянуто гібридні алгоритми, що комбінують ройову оптимізацію з генетичними алгоритмами, описано їхні властивості та приклади їх застосування.

Як приклад реалізації ройових алгоритмів розглянуто мурашині алгоритми, засновані на поведінці мурах під час пошуку поживи. Розроблено алгоритми оптимізації колоній мурах (*ACO — Ant Colony Optimization*), вивчено їхні властивості та оцінено ефективність у досягненні поставленої мети. Наведено приклад вирішення завдання комівояжера, показано результати вирішення цього завдання різною кількістю мурах у різних мурашиних алгоритмах та проаналізовано їхню ефективність.

Наступні п'ять розділів присвячено перспективним методам і технологіям штучного інтелекту (ШІ). Докладно розглянуто нейронні мережі глибокого навчання (*Deep Learning, DL*), зокрема структуру та алгоритми глибокого навчання, їхні властивості, а також методи регуляризації, які застосовують для поліпшення результатів. Проаналізовано переваги й недоліки, зазначено, що головним недоліком є нестача ефективних методів вибору оптимальної структури таких мереж.

Зокрема, у десятому розділі описано новий клас гібридних нейронних мереж глибокого навчання, побудованих на основі методу самоорганізації МГУА. Розглянуто низку гібридних нейронних мереж із різними типами вузлів: нейронами Ванга — Менделя (Лі-Сін Ван (Li-Xin Wang), Джеррі Мендель (Jerry Mendel)), неонечіткими нейронами тощо. Описано методи оптимізації структури таких мереж та їхніх параметрів, наведено приклади їх застосування у задачах прогнозування в макроекономіці, фінансовій сфері та задачах класифікації.

Одинадцятий розділ підручника присвячено розгляду згорткових нейронних мереж (*Convolutional Neural Networks, CNN*) у контексті їх використання для обробки даних із локальними залежностями, таких як зображення та відео. Основою *CNN* є поняття згортки — операції, що фільтрує вхідні дані і виявляє локальні ознаки за допомогою набору фільтрів.

Серед важливих особливостей згорткових нейронних мереж виділено автоматичне навчання ознак, ієрархічну структуру та ефективність у розпізнаванні шаблонів. Наголошено, що архітектура *CNN* складається з кількох типів шарів, кожен з яких виконує специфічну функцію. Зокрема, згортковий шар здійснює операцію згортання, виявляючи локальні шаблони в даних. Шар підвибірки (*pooling*) зменшує розмірність вхідних даних, чутливою зменшуючи чутливість моделі до зсувів. Пакетна нормалізація прискорює навчання та стабілізує нейронну мережу, знижуючи залежність від початкової ініціалізації ваг. Метод виключення (*dropout*) запобігає перенавчанню випадковим вимкненням деяких нейронів під час навчання. Повнозв'язний шар відповідає за інтеграцію ознак та прийняття остаточних рішень.

Показано, що *CNN* довели свою ефективність у задачах комп'ютерного зору та інших сферах, де необхідне оброблення просторових даних, завдяки здатності автоматично навчати релевантних ознак з мінімальним попереднім обробленням.

Дванадцятий розділ присвячено розгляду застосування згорткових і гібридних нейронних мереж для задач аналізу медичних зображень та класифікації пухлин молочної залози. У дослідженні використовували набір даних *Break His (Breast Cancer Histopathological Image Classification)*, що містить 9 109 мікроскопічних зображень тканин пухлин молочної залози, зібраних від 82 пацієнток за різних значеннях збільшення (40×, 100×, 200× та 400×). Зображення розподілено на 2 480 доброякісних та 5 429 злоякісних зразків, кожен з яких має роздільну здатність 700 × 460 пікселів, зберігається у форматі *PNG* з 8-бітною глибиною кольору на канал. Цей набір даних створено у співпраці з лабораторією *P&D Laboratory — Pathological Anatomy and Cytopathology* (Парана, Бразилія).

Було обрано архітектури *CNN*, здатні ефективно виділяти ознаки з медичних зображень, особливо для класифікації пухлин. Під час дослідження налаштовувалися гіперпараметри моделей, зокрема кількість шарів, розміри фільтрів та швидкість навчання, що дало змогу підвищити якість моделей. Результати бінарної класифікації пухлин молочної залози продемонстрували високі показники точності, що свідчило про здатність ефективно розрізняти доброякісні та злоякісні утворення. У багатокласовій класифікації зображень *CNN* показали здатність розрізняти різні типи пухлин.

Гібридну *CNN* на основі нечіткої нейронної мережі (ННМ) *NEFClass* застосовували для класифікації пухлин молочної залози, використовуючи алгоритм навчання, що комбінує переваги згорткових мереж та нечіткої логіки. Експериментальні дослідження показали, що така гібридна архітектура підвищила точність діагностики. Також продемонстровано гібридну *CNN* на основі автокодувальника для виявлення раку молочної залози, яка

показала високу ефективність у діагностиці. Зазначено, що використання згорткових і гібридних нейронних мереж є перспективним підходом для класифікації пухлин молочної залози та аналізу медичних зображень завдяки високій точності й ефективності.

Тринадцятий розділ присвячено розгляду систем виявлення раку легенів на основі стратегій глибокого навчання. Для аналізу медичних зображень використовували двовимірні згорткові нейронні мережі (*2DCNN — Two-Dimensional Convolutional Neural Network*) із багаторазовим навчанням. Цей тип нейронних мереж добре придатний для оброблення даних із просторовою структурою, таких як зображення. Для підвищення точності моделі також використовували рекурентні нейронні мережі з механізмом уваги в поєднанні з *2D CNN*.

Для врахування об'ємної структури легенів було застосовано тривимірну згорткову нейронну мережу (*3DCNN — Three-Dimensional Convolutional Neural Network*), що дало можливість аналізувати тривимірні дані. Набори даних зазнавали попереднього оброблення, зокрема нормалізації та видалення шумів, що підвищувало якість навчання моделей.

Аналіз серії експериментів показав, що використання моделі *DenseNet (Densely Connected Convolutional Network)* у поєднанні з багаторазовим навчанням *(Multiple Instance Learning, MIL)* та розрідженим позначенням міток значно поліпшило результати виявлення раку. Це пов'язано з тим, що в моделі *DenseNet* кожен шар отримує на вхід дані від усіх попередніх шарів, що сприяє збереженню більшої кількості інформації на кожному етапі оброблення і поліпшенню злиття ознак та градієнтів. Рекурентна нейронна мережа з механізмом уваги також показала високу ефективність у системі виявлення раку легенів. Зазначено, що застосування глибоких нейронних мереж, зокрема згорткових та рекурентних із механізмом уваги, є перспективним напрямом для підвищення точності та ефективності виявлення раку легень на основі обробки медичних зображень.

Чотирнадцятий, п'ятнадцятий і шістнадцятий розділи підручника присвячено сучасним системам оброблення природної мови (*Natural Language Processing, NLP*) як галузі штучного інтелекту, що займається взаємодією між комп'ютерами та людською мовою. *NLP* представлено як складну адаптивну систему, враховуючи постійно змінюваний лінгвістичний ландшафт, різноманітність мов, культурні контексти та широке коло застосувань. Підкреслено, що мовні моделі постійно навчаються на динамічних даних, еволюціонують з часом і потребують регулярної адаптації до нових умов.

Розглянуто еволюцію NLP у період з 1964 по 2024 рр.: від простих моделей, як-от *ELIZA* (1960-ті рр.), через статистичні методи 1980—1990-х рр. до сучасних трансформерних моделей, як-от *BERT (Bidirectional Encoder Representations from Transformers)* і *GPT (Generative Pre-trained Transformer)*.

Зростання обчислювальних потужностей і розвиток алгоритмів сприяли появі моделей, здатних обробляти великі обсяги тексту і виконувати завдання на рівні людського розуміння.

Зазначено, що традиційні методи машинного навчання в *NLP* базувалися на статистичних підходах, таких як наївний Баєс, дерева рішень та частотний аналіз слів. Ці методи забезпечували базовий рівень автоматизації текстового аналізу й залишалися популярними до появи глибокого навчання. Глибоке навчання стало революційним етапом у розвитку *NLP*, оскільки моделі на основі нейронних мереж, такі як рекурентні нейронні мережі (*Recurrent Neural Networks, RNN), LSTM (Long Short-Term Memory)*, трансформери та інші, значно підвищили точність обробки тексту завдяки здатності вивчати складні лінгвістичні структури та семантичні відносини.

Розглянуто принципи роботи методів, моделей та алгоритмів *NLP*, включно з попередньою обробкою даних: лематизацією для визначення базової форми слів, сегментацією речень для подальшого аналізу, видаленням стоп-слів для підвищення ефективності, токенізацією (розбиття тексту на компоненти) та виявленням ключових ознак, таких як частота слів і довжина речень. Описано також методики оцінки важливості термінів у документі щодо корпусу документів і алгоритми перетворення слів на багатовимірні вектори для кращого розуміння контексту.

Підкреслено, що системи *NLP* постійно вдосконалюються, і завдяки застосуванню сучасних методів та потужних обчислювальних платформ, які діють на базі центральних (*Central Processing Unit, CPU*), графічних (*Graphics Processing Unit, GPU*), тензорних (*Tensor Processing Unit, TPU*) і нейронних процесорів (*Neural Processing Unit, NPU*) для масивних обчислень досягається висока точність в обробці та розумінні природної мови.

Підручник орієнтовано насамперед на студентів другого і третього рівнів навчання за спеціальностями 122 «Комп'ютерні науки» та 124 «Системний аналіз і управління». Він також буде корисним студентам, викладачам і науковцям інших спеціальностей, які займаються застосуванням штучного інтелекту в своїх дослідженнях і практичній діяльності.

Автори висловлюють щиру вдячність рецензентам: академіку НАН України О.М. Хімічу, д-ру техн. наук, професору кафедри штучного інтелекту Харківського національного університету радіоелектроніки Є.В. Бодянському, д-ру техн. наук, професору, декану факультету математики та цифрових технологій Ужгородського національного університету М.М. Маляру та відповідальному редактору, д-ру фіз.-мат. наук, професору кафедри штучного інтелекту Національного технічного університету України «КПІ ім. Ігоря Сікорського» В.Я. Данилову.

Вступ

Штучний інтелект (ШІ) нині є однією з найбільш революційних галузей сучасної науки. Його історія почалася в середині XX століття і пов'язана з іменами видатних учених, чиї ідеї та досягнення визначили розвиток інтелектуальних машин. Одним із перших, хто висунув ідею створення машин, здатних мислити, був британський математик Алан Тюрінг. У 1950 р. він запропонував концепцію, відому як тест Тюрінга, для перевірки, чи може машина поводитися як людина. Це питання стало основоположним для подальших досліджень у сфері ШІ. Тюрінг припустив, що якщо розмістити людину і комп'ютер у різних кімнатах, а оператор через клавіатуру ставитиме їм питання, і за умови, що відповіді комп'ютера і людини будуть однаковими, то таку машину можна вважати інтелектуальною.

Тюрінг передбачав, що до 2000 року комп'ютер із обсягом пам'яті 10^9 біт і швидкістю оброблення 10^6—10^7 операцій за секунду зможе пройти цей тест протягом 5 хвилин з імовірністю 70 %. Проте, попри значне зростання обсягу пам'яті та швидкості обробки сучасних комп'ютерів, повністю пройти тест Тюрінга ще не вдалося. Це зумовлено складністю завдань, пов'язаних із розумінням комп'ютером природної мови, автоматизацією пошуку рішень у різних галузях, а також іншими невирішеними проблемами.

Офіційним початком розвитку штучного інтелекту стала Дартмутська конференція 1956 року, на якій американський вчений Джон Маккарті ввів термін «штучний інтелект» і започаткував дослідження у цій галузі. У 1958 р. Маккарті розробив мову програмування *LISP*, яка стала основним інструментом для роботи з ШІ на десятиліття. Абревіатура *LISP* означає *LISt Processing* (обробка списків) і була створена спеціально для роботи з символічними даними, що є ключовим аспектом у багатьох завданнях штучного інтелекту. Основною структурою даних у *LISP* є список, що забезпечує її гнучкість для маніпуляцій з деревоподібними структурами, які широко використовують у ШІ.

У 1956 р. Герберт Саймон та Аллен Ньюелл створили першу програму, здатну вирішувати логічні задачі, — *Logic Theorist*, що стало одним із перших практичних застосувань концепцій ШІ. Їхня наступна програма — *General Problem Solver (GPS)*, розроблена у 1957 р., стала основою для розвитку ме-

тодів розв'язання складних задач, подібних до тих, які вирішує людина. На відміну від спеціалізованих програм GPS стала універсальним підходом до вирішення завдань за допомогою методів штучного інтелекту і когнітивної науки, відкриваючи нові можливості для автоматизації процесу мислення.

Паралельно з Маккарті, Саймоном та Ньюеллом ще один піонер у галузі штучного інтелекту — американський вчений Марвін Мінський також зосередився на дослідженні можливостей машинного сприйняття і мислення. Він зробив значний внесок у розробку експертних систем, що імітували розумову діяльність людини. У 1959 р. Мінський заснував Лабораторію штучного інтелекту Массачусетського технологічного інституту (MIT), відому як *MIT AI Lab*, яка відіграла ключову роль у розвитку досліджень у галузі ШІ.

Період із 1970-х до 1990-х років часто називають «зимою ШІ» через скорочення фінансування та зниження інтересу до галузі. Технології того часу мали значні обмеження, що не давали можливості досягти очікуваних результатів. Лише з появою великих обчислювальних потужностей і нових методів навчання, зокрема глибокого навчання, ШІ набув нового імпульсу для розвитку.

У XXI ст. дослідники Джеффрі Хінтон, Ян Лекун та Йошуа Бенжіо здійснили значний *прорив у розробленні нейронних мереж та методів глибокого навчання (Deep* Learning), що стали основою сучасного штучного інтелекту. Їхня робота зумовила створення нових методів навчання багатошарових нейронних мереж. Ці технології сприяли розвитку таких галузей, як-от розпізнавання образів, обробка природної мови та комп'ютерний зір, що дало поштовх для практичного застосування ШІ на новому рівні.

Фундаментальний внесок у розвиток сучасного штучного інтелекту зробили Стюарт Рассел та Пітер Норвіг у своїй монографії «Штучний інтелект: сучасний підхід» (1995), де вони систематизували та глибоко проаналізували основні підходи до створення ШІ. Монографію було перевидано чотири рази. Останнє, четверте, видання (2021) описує широкий спектр технік і методів, таких як машинне навчання, обробленя природної мови, робототехніка, логіка та планування, зосереджуючи увагу на їх поєднанні для створення інтелектуальних систем. Рассел і Норвіг виокремили чотири ключові парадигми:

1) думати як людина — моделювання когнітивних процесів на основі психологічних досліджень;

2) діяти як людина — створення систем, що імітують поведінку людини (наприклад, роботи чи агентні системи);

3) думати раціонально — застосування логіки та алгоритмів для розв'язання задач на основі знань.

4) діяти раціонально — розроблення агентів, що ефективно досягають своїх цілей у змінному середовищі.

Монографія Рассела і Норвіга стала базовим підручником для багатьох поколінь дослідників у галузі ШІ.

Системи штучного інтелекту мають низку унікальних властивостей, що надав їм більшої гнучкості та адаптивності порівняно з традиційними комп'ютерними системами і технологіями автоматизації. Основні відмінності можна визначити так:

1. Наявність мети або групи цілей — ШІ-системи орієнтовані на досягнення конкретних цілей із можливістю вибору підходів для їх виконання.

2. Здатність до планування та пошуку рішень — ШІ-системи можуть планувати свої дії та розробляти стратегії для досягнення поставлених цілей, а також знаходити рішення для нових завдань.

3. Здатність до навчання та адаптації — ШІ-системи здатні навчатися на основі даних, покращуючи свої результати без ручного втручання.

4. Робота в умовах невизначеності — ШІ здатний обробляти неоднорідні дані, працювати з нечіткими інструкціями та адаптуватися до змін середовища.

5. Здатність до самоорганізації та саморозвитку — ШІ вдосконалює власні стратегії та структуру в процесі роботи, розвиваючи нові методи досягнення цілей.

6. Розуміння природної мови — ШІ-системи здатні інтерпретувати природну мову, що полегшує взаємодію з користувачами.

7. Здатність до узагальнення досвіду — ШІ-системи можуть узагальнювати накопичені знання та використовувати їх для вирішення нових завдань.

Значний внесок у розвиток штучного інтелекту зробили і українські вчені Віктор Глушков, Микола Амосов і Олексій Івахненко. Академік Віктор Глушков був одним із засновників кібернетики в СРСР. У 1960-х рр. він розробив концепцію автоматизованих систем управління, що стала основою для розвитку ШІ, а в 1962 р. запропонував проєкт Єдиної державної системи обробки інформації (ОГАС) — першої автоматизованої системи управління економікою. В. Глушков розробив також математичні моделі для автоматизованих машин, які заклали фундамент обчислювальної техніки. Він одним із перших досліджував принципи «адаптації», «самоорганізації» та «самовдосконалення» у теорії автоматів.

В. Глушков активно працював над проблемами розуміння мов і автоматичного доведення теорем як способів моделювання людського інтелекту. У 1961 р. на конгресі *IFIP (International Federation for Information Processing)* у Мюнхені він репрезентував метод семантичних мереж для розуміння змісту речень, що здобув міжнародне визнання. Його науково-дослідницьку програму «Алгоритм очевидності» 1970 року покладено в основу системи автоматизованого доведення теорем (САД), яка успішно розв'язувала задачі з лінійної алгебри.

У відділі академіка Миколи Амосова, запрошеного В. Глушковим до Інституту кібернетики в 1960-х рр., проводилися дослідження з розпізнавання образів, оброблення природної мови та проєкту «штучної руки» для автоматизації моторних функцій роботів. Тут також розроблялися інтелектуальні транспортні роботи (ТАІР — транспортні автоматизовані інтелектуальні роботи) і закладалися основи моделювання розумових процесів на базі концепції нейронних мереж.

Метод групового урахування аргументів (МГУА), створений Олексієм Івахненком у 1960-х рр., став одним із перших алгоритмів для моделювання складних систем на основі експериментальних даних. Завдяки принципу структурної ідентифікації МГУА дає можливість синтезувати оптимальні структури нейронних мереж, що зробило цей метод революційним у галузі ШІ та надзвичайно затребуваним і дотепер.

Таким чином, головними особливостями сучасних ШІ-систем є здатність до навчання, адаптації, планування, роботи в умовах невизначеності та прийняття рішень, що надав їм значно більшої гнучкості й ефективності порівняно з традиційними комп'ютерними технологіями та системами автоматизації [1—6].

Отже, історія штучного інтелекту — це шлях від теоретичних ідей до революційних технологій. Від ранніх теорій Алана Тюрінга до сучасних досягнень у глибокому навчанні ШІ продовжує стрімко розвиватися і змінювати світ. Імена таких учених, як Алан Тюрінг, Джон Маккарті, Марвін Мінський, Джеффрі Хінтон, Ян Лекун, Йошуа Бенжіо, Стюарт Рассел, Пітер Норвіг, Віктор Глушков, Микола Амосов, Олексій Івахненко та інші, назавжди залишаться в історії науки як піонерів, що наблизили людство до створення справжнього інтелекту в машинах.

Системи штучного інтелекту відрізняються від традиційних комп'ютерних технологій і систем автоматизації завдяки унікальним властивостям, які роблять їх гнучкими й здатними адаптуватися до різноманітних завдань і середовищ. Основні відмінності можна структурувати так [7]:

1. Наявність мети або групи цілей функціонування, здатність до цілепокладання: ШІ-системи орієнтовані на досягнення певних цілей або завдань, маючи можливість обирати власні шляхи їх реалізації.

2. Здатність до планування та пошуку рішень: ШІ може планувати свої дії й розробляти стратегії для досягнення визначених цілей, а також знаходити рішення проблем, які виникають, що складно для традиційних систем.

3. Здатність до навчання та адаптації: ШІ-системи можуть навчатися на основі накопичених даних і покращувати свої результати без необхідності втручання людини, тоді як традиційні системи використовують фіксовані алгоритми і не мають такої гнучкості.

4. Здатність працювати в умовах невизначеності та в недостатньо структурованому середовищі: ШІ здатний обробляти великі обсяги неоднорідних даних, працювати з нечіткими інструкціями й адаптувати свою поведінку відповідно до змін у середовищі, тоді як традиційні системи потребують чіткої структури і точних вказівок.

5. Здатність до самоорганізації та саморозвитку: ШІ може вдосконалювати власні стратегії та структуру в процесі роботи, розробляючи нові методи для досягнення своїх цілей, що недоступно для традиційних систем.

6. Здатність розуміти тексти природною мовою: ШІ-системи можуть обробляти й інтерпретувати природну мову, що дає їм можливість взаємодіяти з користувачами у більш природний спосіб. Традиційні системи працюють лише з формальними інструкціями.

7. Здатність до узагальнення накопиченої інформації: системи ШІ здатні узагальнювати досвід і застосовувати його до нових завдань, створюючи рішення на основі попередніх знань.

Щоб створити машини, здатні наближатися за можливостями до мозку людини, потрібно насамперед зрозуміти сутність людського інтелекту та механізми мислення. Протягом останніх десятиліть цій проблемі присвячено чимало наукових праць. Окремо варто відзначити монографію Джеффа Гокінса та Сандри Блейкслі «Про інтелект» (2007) [1], в якій автори на стику нейробіології, психології та кібернетики розробили новаторську теорію відтворення роботи людського мозку. Основними елементами цієї теорії є запам'ятовування минулого досвіду та прогнозування мозком результатів сприйняття навколишнього світу і власних дій. Автори наводять багато переконливих прикладів поведінки людини в різних ситуаціях, що підтверджують цю концепцію.

Дж. Гокінс зазначає: «... Прогнозування, на мою думку, — це не просто одна з функцій кори головного мозку. Це первинна функція неокортексу і основа інтелекту. Кора мозку є органом передбачення. Якщо ми хочемо зрозуміти, що таке розум, що таке творчість, як працює наш мозок і як навчитися створювати розумні машини, нам потрібно осягнути природу прогнозування й зрозуміти, яким чином кора головного мозку їх формує. Навіть поведінку можна найкраще уявити як проміжний продукт процесу прогнозування» [1].

Загалом у результаті 70-річного процесу розвитку сучасний ШІ набув чіткої структуризації. Можна виокремити його основні сучасні напрями та піднапрями:

1. Машинне навчання (*Machine Learning, ML*):

• навчання з учителем (*Supervised Learning*): використовує мічені дані для навчання моделей;

• навчання без учителя (*Unsupervised Learning*): працює з немічеными даними, зокрема для кластеризації та пошуку шаблонів;

• навчання з підкріпленням (*Reinforcement Learning*): агент досліджує середовище через винагороди та покарання;

• напівкероване навчання (*Semi-Supervised Learning*): використовує як мічені, так і немічені дані для навчання;

• глибоке навчання (*Deep Learning*).

2. Нейронні мережі різних класів:

• штучні нейронні мережі (*Artificial Neural Networks, ANNs*): багатошарові нейронні мережі для завдань класифікації, прогнозування тощо;

• згорткові нейронні мережі (*Convolutional Neural Networks, CNNs*): переважно призначені для оброблення зображень і відео;

• рекурентні нейронні мережі (*Recurrent Neural Networks, RNNs*): ефективні для роботи з послідовними даними, як-то текст або часові ряди.

3. Обчислювальний інтелект (*Computational Intelligence, OI*):

• еволюційні алгоритми (*Evolutionary Algorithms*): використовують принципи природних процесів, як-от селекція і мутації, для вирішення оптимізаційних завдань;

• нечітка логіка (*Fuzzy Logic*): дає змогу працювати з нечіткими або частковими істинами, забезпечуючи гнучкість у прийнятті рішень;

• штучні нейронні мережі (*ANNs*): застосовуються як у класичному штучному інтелекті, так і в обчислювальному інтелекті;

• ройовий інтелект (*Swarm Intelligence*): оптимізує рішення завдяки взаємодії групи агентів (наприклад, алгоритми мурашиних колоній, алгоритми рою частинок тощо).

4. Автономні системи штучного інтелекту і робототехніка (*Autonomous Systems of Artificial Intelligence and Robotics*):

• технології автономного виконання завдань: системи, що працюють без прямого людського контролю, завдяки використанню алгоритмів машинного навчання, комп'ютерного зору, оброблення природної мови та інших методів для виконання завдань, які раніше потребували участі людини;

• автономні транспортні засоби: самокеровані автомобілі і дрони, які здійснюють поїздки чи доставки без участі водія або оператора, використовуючи сенсори, камери та алгоритми для розпізнавання оточення й ухвалення рішень у реальному часі;

• роботи-помічники: роботи, які виконують завдання в домашніх умовах, як-от прибирання, приготування їжі або догляд за рослинами;

• індустріальні роботи: роботи, що працюють на виробничих лініях, виконуючи завдання з високою точністю та швидкістю;

• агороботи: роботи, що здійснюють завдання на фермах, як-от збирання врожаю, обробіток ґрунту або догляд за рослинами (наприклад, автономні трактори чи дрони для моніторингу рослин);

• роботи для медичних застосувань: роботи, що допомагають у хірургії, реабілітації або догляді за пацієнтами. Наприклад, робот-хірург, здатний виконувати складні операції з високою точністю;

• інтелектуальні агенти: віртуальні помічники, які можуть автоматично виконувати завдання або надавати інформацію, як-то чатботи чи системи рекомендацій.

5. Оброблення природної мови (*Natural Language Processing, NLP*):

• синтаксичний аналіз (*Syntax Analysis*): аналіз структури мови;

• семантичний аналіз (*Semantic Analysis*): інтерпретація значення слів і речень;

• машинний переклад (*Machine Translation*): автоматичний переклад між різними мовами;

• розпізнавання та генерація мовлення (*Speech Recognition and Generation)*: перетворення мовлення на текст і навпаки.

6. Комп'ютерний зір (*Computer Vision*): технології аналізу та інтерпретації зображень і відео для розпізнавання об'єктів, класифікації та виявлення подій.

7. Системи підтримки ухвалення рішень (*Decision Support Systems*): інтелектуальні системи, які допомагають людям або іншим системам ухвалювати обґрунтовані рішення, базуючись на даних та аналітиці.

8. Експертні системи (*Expert Systems*): системи, що імітують рішення експерта в певній галузі за допомогою баз знань і логічних правил.

9. Гібридні системи глибокого навчання: новий клас гібридних систем глибокого навчання на основі самоорганізації (метод МГУА). Цей напрям було розроблено у спільних дослідженнях наукової школи з обчислювального інтелекту КПІ (проф. Ю. Зайченко) та наукової школи Харківського національного університету радіоелектроніки (ХНУРЕ) під керівництвом проф. Є. Бодянського. Основна особливість гібридних систем полягає у можливості вибору оптимальної структури мережі за принципом самоорганізації. Завдяки пошаровому навчанню гібридної нейронної мережі усувається проблема згасання або вибуху градієнта, що виникає у багатошарових мережах у разі використання алгоритму зворотного поширення помилки (*Back Propagation*).

10. Штучний загальний інтелект (*Artificial General Intelligence, AGI*): напрям на створення систем, здатних виконувати будь-які завдання на рівні людського інтелекту або навіть перевершувати його.

11. Етичні та соціальні аспекти штучного інтелекту: вивчення етичних і соціальних викликів, пов'язаних із застосуванням ШІ, таких як конфіденційність, відповідальність за прийняття рішень, вплив на суспільство тощо.

Ці розділи охоплюють більшість сучасних підходів і напрямів розвитку штучного інтелекту. Однак окремо варто виділити розділ 3 наведеної струк-

туризації — обчислювальний інтелект (*Computational Intelligence, OI*). Наприкінці 1980-х і на початку 1990-х років, у відповідь на потребу створення адаптивних систем для вирішення завдань, де традиційні методи ШІ, як-от символьне моделювання та формальна логіка, виявилися недостатніми, сформувався новий напрям під назвою «обчислювальний інтелект». Цей напрям ґрунтувався на біологічних, еволюційних та адаптивних процесах і охоплював такі підгалузі: нейронні мережі, нечіткі системи та еволюційні алгоритми.

Термін «обчислювальний інтелект» уперше запропонував Джеймс Бездек (James C. Bezdek) у 1992 р., а пізніше, у 1994 р., поняття було уточнене [2, 3]. Бездек визначив обчислювальний інтелект як систему, що оперує цифровими (низькорівневими) даними; має компоненти розпізнавання образів; використовує знання у сенсі штучного інтелекту і демонструє обчислювальну адаптивність, відмовостійкість та рівень помилок, наближений до продуктивності людини.

У 1993 р. Роберт Маркс (Robert J. Marks II) доповнив визначення OI, наголошуючи на його основних технологічних складових [4]: «…нейронні мережі, генетичні алгоритми, нечіткі системи, еволюційне програмування та штучне життя є будівельними блоками OI». Пізніше Девід Фогель (David B. Fogel) об'єднав ці технології під терміном «обчислювальний інтелект», описуючи його як методи, здатні адаптуватися до нових завдань без використання явних людських знань [5].

Важливий внесок у розвиток OI зробили Збігнєв Міхалевіч (Zbigniew Michalewicz) та Джеймс Макклиленд (James L. McClelland). Міхалевіч зосередився на біоінспірованих методах ШІ, а Макклиленд був одним із розробників моделей обчислювальної нейробіології, що вплинуло на формування концепції обчислювального інтелекту. У 1994 р. на Першій міжнародній конференції з обчислювального інтелекту, організованій *IEEE Computational Intelligence Society*, OI було офіційно визнано окремим науковим напрямом у межах штучного інтелекту.

Обчислювальний інтелект має такі особливості. Він використовує адаптивні системи, здатні навчатися на основі досвіду та адаптуватися до змін у середовищі. Його ключова особливість полягає в здатності працювати з нелінійними, стохастичними процесами, часто без повного розуміння фізичних принципів, на яких вони ґрунтуються. OI використовує такі підходи, як штучні нейронні мережі, нечітку логіку та еволюційні алгоритми, що дає можливість вирішувати складні завдання в умовах невизначеності.

До переваг обчислювального інтелекту належать:

• *адаптивність:* системи OI можуть самостійно навчатися на нових даних і змінювати свою поведінку, адаптуючись до нових умов;

• *стійкість до шуму та до невизначеностей*: OI добре працює з неповними або неточними даними завдяки методам нечіткої логіки та нейронних мереж;

• *паралельна обробка:* нейронні мережі здатні паралельно обробляти інформацію, що робить їх ефективними для вирішення складних завдань у реальному часі;

• *гнучкість:* ОІ можна застосовувати в різних галузях, таких як робототехніка, медична діагностика, прогнозування та автоматизація.

Водночас у разі практичного застосування обчислювального інтелекту потрібно враховувати і деякі його недоліки. Основний недолік ОІ полягає в проблемі «чорної скриньки»: часто важко інтерпретувати внутрішні процеси, що відбуваються в ОІ-системах, зокрема в нейронних мережах, що ускладнює пояснення результатів, отриманих із використанням таких систем. Крім того, тренування нейронних мереж та еволюційних алгоритмів потребує значних обчислювальних ресурсів і часу, а методи ОІ не завжди гарантують оптимальне рішення для складних задач. Для ефективного навчання ОІ-систем часто потрібна велика кількість даних, що може ускладнювати реалізацію алгоритмів.

Отже, обчислювальний інтелект став важливою віхою у розвитку технологій штучного інтелекту, пропонуючи нові підходи до вирішення завдань в умовах невизначеності та складності. Завдяки використанню нейронних мереж, нечіткої логіки та еволюційних алгоритмів обчислювальний інтелект відкриває широкі можливості для створення адаптивних і стійких систем. Водночас певні обмеження, зокрема складність інтерпретації результатів і значні вимоги до обчислювальних ресурсів, залишаються викликами для подальшого розвитку цієї галузі.

Аналіз праць згаданих науковців дає змогу сформулювати узагальнене визначення обчислювального інтелекту: *це сукупність технологій, моделей, методів і програмних засобів, призначених для вирішення неформальних, творчих завдань у різних сферах людської діяльності, використовуючи підходи, що відображають когнітивні процеси людини (такі як нечіткість мислення, інтуїтивність, креативність, самонавчання тощо), зокрема в ухваленні рішень, класифікації та розпізнаванні образів* [7].

Структуру обчислювального інтелекту зображено на рис. 1.

У структурі обчислювального інтелекту можна виділити такі основні компоненти [7]:

• технології;

• моделі, методи та алгоритми;

• прикладні задачі.

До технологій обчислювального інтелекту входять:

• нейронні мережі (НМ);

• системи нечіткої логіки (СНЛ) та нечіткі нейромережі (ННМ);

• еволюційне моделювання (ЕМ).

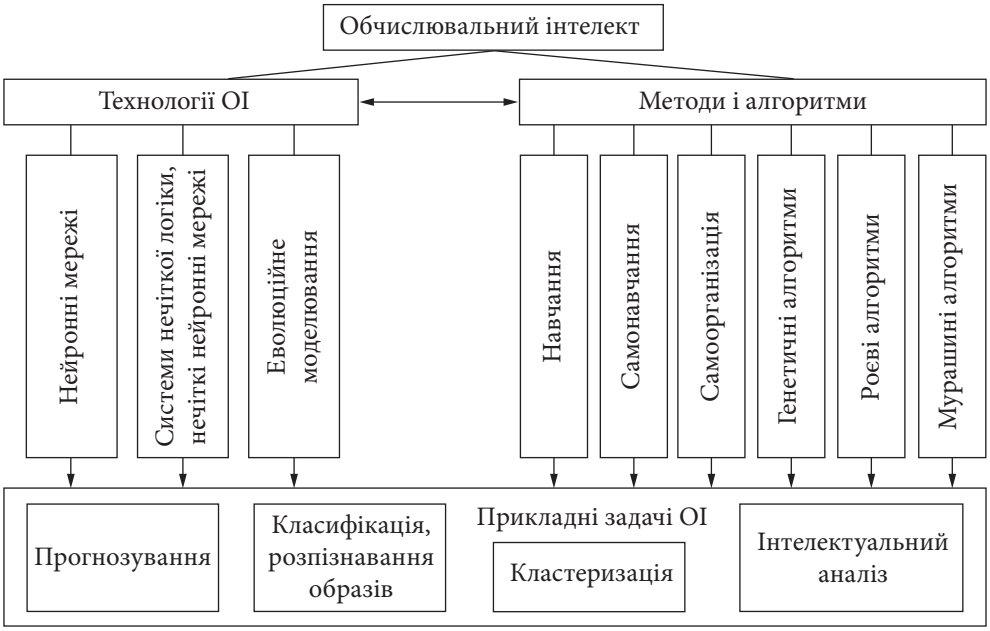


Рис. 1. Структура обчислювального інтелекту

До методів і алгоритмів обчислювального інтелекту належать:

- методи навчання;
- методи самонавчання;
- методи самоорганізації;
- генетичні алгоритми (ГА);
- ройові алгоритми;
- мурашині алгоритми.

Технології і методи ОІ використовують для вирішення завдань штучного інтелекту. Основні класи завдань ОІ, характерні для розумової діяльності, містять:

- прогнозування та передбачення;
- класифікацію та розпізнавання образів;
- кластеризацію, тобто мимовільне розбиття множини об'єктів на класи однорідних об'єктів;
- інтелектуальний аналіз даних;
- нечіткий логічний висновок;
- ухвалення рішень.

З позицій системного аналізу між технологіями і методами ОІ можна виділити такі взаємозв'язки:

1. Методи навчання широко застосовують для налаштування ваг зв'язків у нейронних мережах та нечітких нейронних мережах.

Як критерії навчання використовують такі:

• середній квадрат відхилення СКВ (*MSE*):

$$\underline{q^2} = \frac{1}{N} \Sigma_{i=1}^{N} (y_i - \hat{y}_i(w))^2 \to \min_w;$$

• середня абсолютна процентна похибка (*MAPE)*:

$$\varepsilon^2_{\text{сапп}} = \frac{1}{N} \Sigma_{i=1}^{N} |y_i - \hat{y}_i(w)| \to \min_w,$$

де y_i — бажаний вихід НМ для i-го вхідного сигналу; $\hat{y}_i$ — фактичний вихід НМ для i-го вхідного сигналу; W — матриця ваг, $W = |w_{kj}|_{k=\underline{1, n}, j=\underline{1, m}}$.

2. Під час навчання задається зазвичай навчальна вибірка $L = (x_1, y_1)$, (x_2, y_2), (x_N, y_N), де x_i — вхідний вектор ($x \in R^{(n)}$), y_i — вихідний сигнал (бажаний вихід — мета навчання).

3. Для навчання застосовують різноманітні методи та алгоритми оптимізації, зокрема градієнтний метод та його модифікації [6—8], метод сполучених градієнтів (СГ), методи другого порядку (метод Ньютона), а також широко використовуються генетичні алгоритми (ГА) [9].

У разі відсутності вчителя (тобто коли мета невідома або не задана) використовують методи та алгоритми самонавчання. У такому випадку завдання полягає в автоматичному розбитті умовної вибірки L на кілька підмножин, які називаються кластерами і для яких встановлюються відповідні критерії та метрики класифікації.

Варто зазначити, що між штучним інтелектом (ШІ) та обчислювальним інтелектом (ОІ) є тісний зв'язок. Обидві концепції є ключовими у сфері комп'ютерних наук, але мають різні фокуси у розробленні інтелектуальних систем. Штучний інтелект полягає у створенні систем, здатних виконувати завдання, що потребують людського інтелекту, такі як навчання, розпізнавання мовлення, розуміння природної мови, ухвалення рішень і розпізнавання образів. Натомість обчислювальний інтелект спрямований на вирішення завдань за допомогою алгоритмів і моделей для розв'язання проблем, які зазвичай не мають точного розв'язку або є складними для традиційних обчислювальних методів. ОІ часто використовує еволюційні алгоритми, нечіткі системи та штучні нейронні мережі.

Взаємозв'язок між ШІ та ОІ

1. Перетин та інтеграція:

• використання технік ОІ у ШІ. Обчислювальний інтелект є важливою складовою сучасних систем ШІ. Наприклад, глибинне навчання, яке є підмножиною машинного навчання, використовує штучні нейронні мережі, що є частиною обчислювального інтелекту;

• методи ОІ у ШІ. Методи еволюційних алгоритмів і нечітких систем застосовуються в ШІ для розв'язання складних оптимізаційних завдань і моделювання невизначеностей.

2. Інструменти і методи:

• нейронні мережі. Штучні нейронні мережі є прикладом обчислювального інтелекту, проте активно використовуються у ШІ для вирішення завдань глибинного навчання;

• еволюційні алгоритми використовуються в ШІ для налаштування моделей та пошуку оптимальних рішень у великих просторах можливих розв'язків.

3. Застосування:

• ШІ-застосунки. Штучний інтелект широко застосовує обчислювальний інтелект для вирішення практичних завдань, як-то розпізнавання зображень, природна мова, робототехніка тощо;

• ОІ-застосунки. Обчислювальний інтелект доцільно використовувати у сценаріях, що потребують адаптивності та гнучкості: оптимізація складних систем, управління і аналіз великих обсягів даних.

Отже, штучний інтелект та обчислювальний інтелект переплітаються у багатьох сучасних технологіях, і часто складно встановити чітку межу між ними. Водночас ОІ забезпечує методи та інструменти, які підвищують здатність систем ШІ до оброблення й розуміння складних і нечітких даних. ШІ використовує ці методи для створення інтелектуальних систем, які можуть автоматично навчатися та адаптуватися до нових умов.

Термін «обчислювальний інтелект» близький за змістом до широковживаного у світових виданнях терміна «*soft computing*» [10, 11] —«м'які обчислення». Концепція м'яких обчислень була вперше запропонована Лотфі А. Заде (Lotfi A. Zadeh) у 1981 р. для аналізу «м'яких» даних. М'які обчислення — це складна комп'ютерна методологія, що охоплює нечітку логіку, генетичні алгоритми, нейрокомп'ютинг і ймовірнісні обчислення [11]. Ці складові не конкурують, а взаємодіють між собою, створюючи синергію. Основний принцип м'яких обчислень полягає у врахуванні неточності, невизначеності, часткової істини та апроксимації для досягнення робастності, зниження вартості рішень і кращої відповідності реальності. Чотири основні складові м'яких обчислень такі:

1) нечітка логіка — наближені обчислення, грануляція інформації, обчислення словами;

2) нейрокомп'ютинг — навчання, адаптація, класифікація, системне моделювання та ідентифікація;

3) генетичні обчислення — синтез, налаштування і оптимізація через систематизований випадковий пошук та еволюційні процеси;

4) ймовірнісні обчислення — управління невизначеністю, мережі довіри, хаотичні системи, прогнозування.

Традиційні комп'ютерні обчислення *(«hard computing»)* забезпечують високу точність, однак часто не відповідають вимогам реального світу. У

«м'яких» обчисленнях виділяють два основні класи завдань: перший стосується проблем, для яких неможливо отримати повну інформацію, другий — проблем, які є недостатньо чітко визначеними.

Кожна з компонент м'яких обчислень має власні внутрішні виклики, але їх інтеграція створює синергетичний ефект, що компенсує ці недоліки. Наприклад, нейронні мережі вирішують завдання апроксимації або класифікації через налаштування вагових коефіцієнтів, але знання в них розподілені серед численних зв'язків, що ускладнює інтерпретацію результатів для користувача, зумовлюючи потребу в експлікації знань. Натомість системи нечіткого виведення забезпечують прозорість завдяки явним продукційним правилам, що дає змогу створювати протоколи пояснень. Однак ця прозорість досягається завдяки тривалому творчому процесу попереднього здобуття знань та налаштування правил. Основні виклики нечітких систем полягають у складнощах автоматичного отримання знань із даних, автоматизованій редукції правил та налаштуванні функцій належності під час навчання.

Застосування генетичних алгоритмів потребує попереднього вивчення проблеми для вибору ймовірнісної моделі та формулювання цільової функції. Вибір конкретної компоненти м'яких обчислень як базової зазвичай визначає загальну архітектуру гібридної системи (під гібридною мається на увазі «м'яка» система).

Важливими технологіями обчислювального інтелекту стали штучні нейронні мережі (ШНМ), що моделюють нейронну систему мозку людини та високоорганізованих тварин. У 1958 р. відомий американський психолог і інформатик, один із піонерів штучного інтелекту Франк Розенблатт (Frank Rosenblatt) розробив ідею першої обчислювальної моделі штучного нейрона — перцептрона, що стала ключовою концепцією для ранніх досліджень у галузі нейронних мереж. Перцептрон Розенблатта був запрограмований на здатність до навчання, що давало йому можливість виконувати прості класифікаційні завдання.

У своїй книзі «Принципи нейродинаміки» [12] Ф. Розенблатт детально виклав концепцію перцептронів і нейродинамічні принципи, покладені в основу їх роботи. Він описав математичні моделі навчання, пов'язані з адаптацією нейронів для вирішення завдань розпізнавання образів і класифікації. Книга стала важливим внеском у розвиток штучного інтелекту, хоча деякі обмеження перцептронів (зокрема неможливість вирішувати нелінійно роздільні завдання) на деякий час загальмували розвиток цієї галузі, поки не з'явилися багатошарові нейронні мережі.

Перцептрон Розенблатта був тришаровою моделлю, що складалася з трьох типів нейронів: *S*-нейрони (сенсорні елементи), *A*-нейрони (асоціативні) та *R*-нейрони (нейрони, що реагують), які забезпечують вихідну реакцію нейронної мережі на вхідний сигнал. Структуру тришарового перцептрона, як модель зорового механізму мозку людини, зображено на рис. 2.

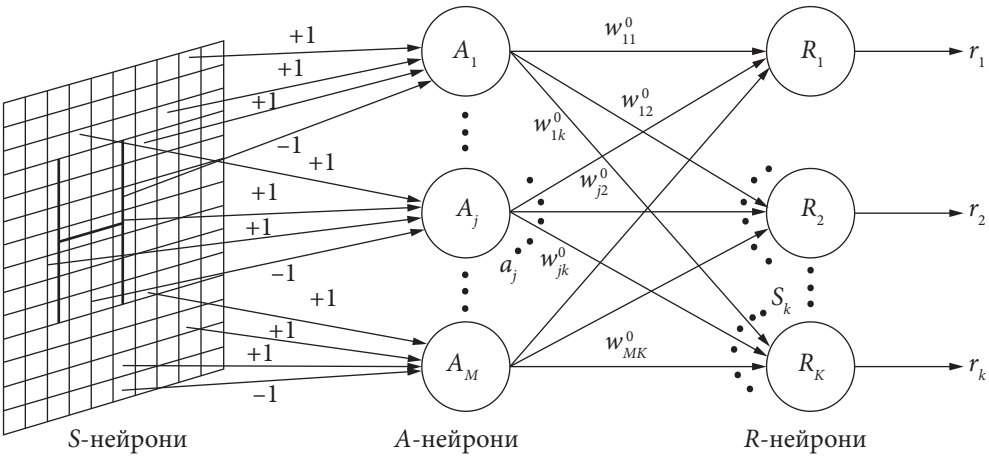


Рис. 2. Структура тришарового перцептрона

Під час проєктування зображення на сенсорне поле з *S*-нейронів, що представляють чутливі фотоелементи, *S*-нейрони, розміщені в комірках (*i*, *j*), на які потрапляють елементи зображення, спрацьовують і видають сигнал $S_{ij}(t) = 1$.

Сенсорні елементи з'єднані випадковими зв'язками з *A*-елементами аналогічно зоровому механізму людини. *A*-зв'язки (*i*, *j*) бувають двох типів: збудливі, їх вага $w_{ij} = 1$, і гальмівні $w_{ij} = -1$. Число вхідних збудливих зв'язків *x* та гальмівних *y* однакова для всіх *A*-нейронів.

Сумарний сигнал на вході асоціативного нейрона A_j визначається так:

$$g_i = \sum_{i \in P_j} w_{ij} S_i = \sum_{i \in P_j^з} S_i - \sum_{i \in P_j} S_i,$$

де P_j— множина вхідних зв'язків (сигналів нейрона A_j); $P_j^з$ — множина збудливих зв'язків у нейрона; $P_j^г$ — множина гальмівних зв'язків.

Очевидно: $|P_j^з| = x$, $|P_j^г| = y$.

Вихідний сигнал асоціативного нейрона A_j визначається так:

$$a_j = f(g_j),$$

де $f_1(g_i)$ — функція активації *A*-нейронів.

Тут найчастіше використовується два види функцій:

1) релейна $a_j = \begin{cases} 1, \text{ якщо } g_i \geq \theta, \text{ де } \theta \text{ — поріг;} \\ 0 \text{ у іншому випадку;} \end{cases}$

2) сигмоїдальна $f_1(x) = \dfrac{1}{1+e^{-x}}$.

A-нейрони пов'язані з *R*-нейронами «кожний з кожним». Позначимо через вагу зв'язку нейрона A_j з нейроном R_k.

Сумарний сигнал S_k на вході R_k визначається за умови:

$$S_k = \sum_{j=1}^{J} a_j w_{jk}^0.$$

Спрацьовує той нейрон R_k, на вході якого сигнал S_k максимальний. Нейрон, який спрацював, визначає відповідний клас об'єктів (образ). Якщо позначити r_k через вихідну реакцію нейрона R_k, то

$$r_k = \begin{cases} 1, \text{ якщо } S_k = \max\limits_j S_j, \\ 0 \text{ у іншому випадку.} \end{cases}$$

Ваги зв'язків $||w^0_{jk}||$ налаштовують так, щоб забезпечити необхідну класифікацію зображень. Для цього використовують відповідні алгоритми навчання.

Наведемо так званий алгоритми «із заохоченням — покаранням». У разі правильної реакції перцептрона ваги активних вхідних зв'язків нейрона, що спрацював, збільшуються і заохочуються так:

$$w_{jk}(t+1) = w_{jk}(t) + \Delta w.$$

У разі неправильної реакції, навпаки, караються:

$$w_{jk}(t+1) = w_{jk}(t) - \Delta w.$$

Це так званий α-алгоритм навчання. Інший, γ-алгоритм навчання, відрізняється тим, що в ньому ваги зв'язків змінюються так, що загальна сума ваг у кожного нейрона залишається незмінною:

$$\sum_{j=1}^{J}, w_{jk}(t+1) = \sum_{j=1}^{J}, w_{jk}(t),$$

$$w_{jk}^a(t+1) = w_{jk}^a(t) + \Delta w_1,$$

$$w_{jk}^m(t+1) = w_{jk}^m(t) - \Delta w_2,$$

де w_{jk}^a — вага активного зв'язку; w_{jk}^m — вага неактивного зв'язку.

Наприкінці 1960-х — на початку 1970-х років продовжився «бум» робіт з дослідження нейронних мереж. Було запропоновано нові архітектури, зокрема нейрон *Adaline (Adaptive Linear Neuron)* і нейронну мережу *Madaline (Multiple Adaline)*, розроблені Бернардом Уідроу та його колегою Тедом Гоффом у Стенфордському університеті, навчальні матриці Карла Штейнбуха (*Steinbuch Learning Matrices*), а також систему «Альфа» О.Г. Івахненка [13].

Так, нейрон *Adaline,* або адаптивний лінійний нейрон, був моделлю штучного нейрона, в якій навчання здійснювалося через мінімізацію середньоквадратичної похибки (*MSE*) між реальним виходом і бажаним результатом. На відміну від перцептрона *Adaline* використовував лінійну функцію активації замість порогової функції. Це означає, що *Adaline* міг виводити безперервні значення, що сприяло ефективнішому налаштуванню ваги ней-

рона. Навчання здійснювалося за допомогою методу градієнтного спуску, що коригувало ваги нейрона залежно від помилки на виході.

Нейронна мережа *Madaline (Multiple Adaline)* складалася з декількох *Adaline*-нейронів і була прикладом багатошарової нейронної мережі. *Madaline* використовувала алгоритм навчання, відомий як *Madaline Rule*, що давало можливість здійснювати адаптивне навчання, коригуючи ваги на основі вихідних сигналів. *Madaline* стала однією з перших багатошарових нейронних мереж, і це дало змогу подолати деякі обмеження одношарових моделей, таких як простий перцептрон. Її основною перевагою була здатність до оброблення складніших нелінійних задач.

Навчальні матриці Штейнбуха були розроблені німецьким інженером Карлом Штейнбухом у 1960-х рр. Це одна з перших моделей навчання, яка полягала в запам'ятовуванні шаблонів завдяки зберіганню відповідностей між вхідними та вихідними векторами в спеціальній матриці. У таких матрицях під час навчання зв'язки між елементами підлаштовувалися на основі відповідностей вхідних сигналів і бажаних виходів, даючи змогу реалізовувати базові алгоритми розпізнавання образів.

Система «Альфа» — це самонавчальна розпізнавальна система, розроблена українським вченим Олексієм Григоровичем Івахненком у 1960-х рр. [13]. Вона стала однією з перших реалізацій штучних нейронних мереж, здатних до самонавчання та адаптації, і використовувала принципи самоорганізації та самонавчання для розпізнавання образів і прогнозування. Система «Альфа» могла адаптуватися до нових даних без зовнішнього втручання, що було новаторським підходом на той час. Однією з ключових її особливостей було використання позитивних зворотних зв'язків, що сприяло підсиленню корисних сигналів і підвищенню точності розпізнавання. Система здатна розпізнавати складні образи та ситуації, що забезпечувало її ефективність для застосування в різних галузях, включно з технічною кібернетикою та автоматичним керуванням.

Ці інновації відіграли ключову роль у розвитку адаптивних систем і нейронних мереж. *Adaline* та *Madaline*, хоча і мали певні обмеження, стали фундаментом для сучасних підходів до машинного навчання та нейронних обчислень. Навчальні матриці Штейнбуха надихнули до пізніших розроблень, спрямованих на створення пам'яті для розпізнавання образів. Система «Альфа» стала важливою для розвитку штучного інтелекту та нейронних мереж, заклавши основи для подальших досліджень у цій сфері. Вона продемонструвала можливість створення машин, здатних до самонавчання та адаптації, що стало підґрунтям для сучасних технологій штучного інтелекту.

Багато вчених спрямували свої зусилля на вивчення потенційних можливостей перцептрона. Зокрема, відомий математик у галузі теорії груп, логіки та алгоритмів Павло Сергійович Новіков на початку 1960-х рр. довів низ-

ку теорем про збіжність процесу навчання перцептрона до бажаної реакції. Його результати стосувалися умов, за яких навчальний процес перцептрона буде збіжним і зможе досягати бажаних вихідних реакцій для різних класів завдань. Новіков досліджував питання збіжності вагових коефіцієнтів перцептрона, що є критичним для коректного функціонування навчальних алгоритмів. Він довів, що за певних умов процес налаштування ваг (параметрів) буде стабільно збігатися до стану, який забезпечує коректну класифікацію або розпізнавання образів відповідно до поставленого завдання. Це включало аналіз залежності збіжності від початкових умов, навчальної вибірки та навчальних правил. Робота Новікова допомогла створити теоретичний фундамент для моделей перцептронів, що сприяло глибшому розумінню математичних аспектів адаптації та навчання штучних нейронів.

Однак сподівання на широкі можливості перцептрона як моделі мозку не виправдалися. Подальші дослідження показали, що елементарний тришаровий перцептрон не володіє важливими властивостями людського інтелекту, такими як здатність до екстраполяції (узагальнення). Наприклад, перцептрон, навчений розпізнавати об'єкт в одній частині сенсорного поля, не міг правильно розпізнати його в іншій частині без додаткового навчання. Крім того, перцептрон не має інваріантності до змін масштабу та повороту зображень.

У 1971 р. професори Массачусетського технологічного інституту Марвін Мінський (Marvin Minsky) і Сеймур Пейперт (Seymour Papert) у своїй роботі показали, що тришаровий перцептрон не здатний вирішувати навіть просте логічне завдання — реалізувати логічну функцію *XOR* — виняткове «АБО», в якому бінарні об'єкти $X_1 = [0; 0]$ і $X_2 = [1; 1]$ мають бути віднесені до одного класу ($k = 2$), а $X_3 = [1; 0]$ і $X_4 = [0; 1]$ — до іншого ($k = 1$). Завдання *XOR* для розмірності $n = 2$ наведено на рис. 3.

У відповідь на критику Франк Розенблатт запропонував ускладнити структуру перцептрона, створивши його чотиришарову версію з двома шарами вагових зв'язків, що налаштовуються. Такий перцептрон міг легко розв'язувати задачу *XOR*. Проте виникла нова проблема: було неясно, як налаштовувати два шари ваг, щоб забезпечити збіжність навчання до бажаної реакції, що спричинило хвилю песимізму стосовно перспективи цих розробок з боку математиків. Через цю критику дослідження нейронних мереж були несправедливо призупинено на ціле десятиліття, що завдало шкоди розвитку цього напряму в галузі ШІ.

Лише через десять років, завдяки дослідженням Джеймса Андерсона (James A. Anderson), Джеффрі Гінтона (Geoffrey E. Hinton) та Рональда Вільямса (Ronald J. Williams), було запропоновано нейронну мережу зі зворотним поширенням помилки (*Back Propagation*) [15, 16]. Її структура є багатошаровим перцептроном (*Multilayer Perceptron, MLP*). Для цієї мережі розробили алгоритм зворотного поширення, який уможливлює налаштовування мере-

жі з довільною кількістю шарів і забезпечення збіжності до необхідної реакції (класифікації або прогнозування).

Було доведено теорему про універсальну апроксимацію нейронними мережами, яка стверджує: «Існує нейронна мережа зі зворотним поширенням помилки, здатна реалізувати будь-яке безперервне відображення: $y = F(x)$ за певних загальних припущень». Ця теорема обґрунтувала широкі можливості нейронних мереж зі зворотним поширенням і стимулювала різке зростання досліджень у цій галузі.

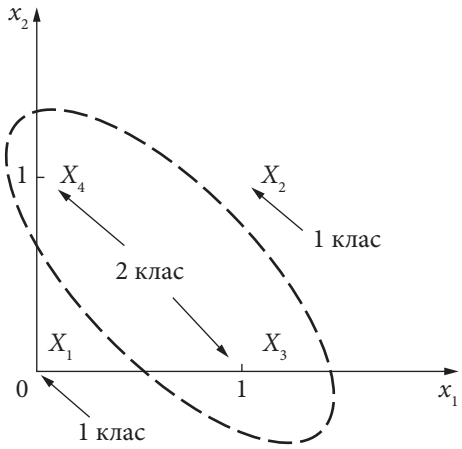


Рис. 3. Завдання *XOR* для розмірності $n = 2$

Окрім класичних нейронних мереж із послідовними зв'язками між шарами останнім часом створено нейронні мережі зі зворотними зв'язками (рекурентні нейронні мережі), такі як мережі Хопфілда та Хеммінга. Використання додатного зворотного зв'язку в таких мережах надає їм унікальної властивості: за певних умов їхня поведінка прагне до сталого стану (нерухомої точки), що називається атрактором. Атрактори представляють пам'ять системи, а нейронна мережа Хопфілда є моделлю асоціативної пам'яті людини. Коли на вхід мережі Хопфілда подається зашумлене або спотворене зображення об'єкта, вона за асоціацією «розпізнає» його і видає на виході неспотворене зображення — еталон, який є одним із атракторів.

Сьогодні нейронні мережі широко застосовують у прогнозуванні, класифікації, розпізнаванні образів, апроксимації функцій та управлінні технологічними процесами.

Перевагами нейронних мереж є здатність до навчання і налаштування вагових зв'язків між нейронами для отримання необхідної реакції, а також паралельно оброблення вхідної інформації, що забезпечує високу продуктивність нейронних мереж загалом, незважаючи на відносно низьку швидкість передання інформації через нейронні зв'язки та швидкість роботи самих нейронів.

Нейронні мережі розрізняють за архітектурою, функціями активації нейронів і алгоритмами налаштування (навчання) вагових зв'язків.

Важливою властивістю інтелекту є здатність до самоорганізації та самонавчання, коли «вчителя» у системі немає, а правильна класифікація об'єкта невідома (наприклад, під час розпізнавання образів). У такому разі система сама, аналізуючи різні об'єкти (зображення) у просторі ознак, розбиває їх на підмножини (кластери), за рівнем «подібності» або близькості об'єктів у просторі ознак. Для цього можуть використовуватися різні метрики близькості.

Нейронні мережі з самоорганізацією (зокрема, мережа Кохонена) використовують змагальне навчання і демонструють здатність до автоматичної класифікації об'єктів різної природи.

У завданнях навчання та самонавчання нейронних мереж налаштовуються ваги зв'язків $w = w'$ з учителем або без нього. При цьому структура мережі не змінюється. Однак у деяких задачах під час налаштування необхідно не лише коригувати ваги, а й змінювати структуру мережі. У таких випадках доцільно застосовувати моделі та методи самоорганізації.

До методів самоорганізації відносять, зокрема, метод індуктивного моделювання, так званий метод групового урахування аргументів О.Г. Івахненка (МГУА) [17]. Цей метод використовувався для відновлення невідомої функції прогнозу або класифікації $y = f(x_1, x_2, \ldots, x_n)$ за експериментальними даними. Метод ґрунтується на основних ідеях еволюції: схрещуванні (кросовері) батьківських пар, генерації нащадків, селекції (відборі кращих нащадків), перевірці умов зупинки.

Для відновлення невідомої функції n змінних у класі поліномів (так званих поліномів Колмогорова — Габора) (за іменами Андрія Колмогорова і Денеша Габора) виду

$$y = f(x_1, x_2, \ldots, x_n) = a_0 + \sum_{i=1}^{n} a_i x_i + \sum_{j \geq i} \sum_{i=1}^{n} a_{ij} x_i x_j + \sum_{k \geq j \geq i} \sum_{i=1}^{n} a_{ijk} x_i x_j x_k + \ldots$$

використовуються елементарні функції від кожної пари змінних. Такі функції називаються частковими описами:

$$y = \varphi(x_i, x_j) = \begin{cases} a_0 + a_i x_i + a_j x_j \text{ — лінійні;} \\ a_0 + a_i x_i + a_j x_j + a_{ii} x_i^2 + a_{jj} x_j^2 + a_{ij} x_i x_j \text{ — квадратні.} \end{cases} \quad (1)$$

Для кожного такого часткового опису (1) за навчальною вибіркою $L_{\text{навч}}$ з використанням методу найменших квадратів (МНК) знаходять оцінки невідомих коефіцієнтів $\{a_i\},\{a_{ij}\}$, а за перевірною вибіркою $L_{\text{перев}}$ — найкращу модель. Для цього використовують зовнішні критерії селекції:

1) регулярності:

$$\overline{\varepsilon}^2_{\text{пр}} = \frac{1}{N_{\text{пр}}} \sum_{i=1}^{N_{\text{пр}}} (y_i - \hat{y}_i(x))^2,$$

де y_i — реальне значення для i-ї точки; $\hat{y}_i(x)$ — вихід моделі; $N_{\text{пр}}$ — об'єм вибірки для перевірки;

2) незміщенності:

$$N_{\text{см}} = \frac{1}{N_{\text{пр}}} \sum_{i=1}^{N_{\text{пр}}} (y_i^* - y_i^{**})^2.$$

Далі застосовують процедуру селекції — відбору F кращих часткових описів за мінімумом критерію $\overline{\varepsilon}^2_{\text{пр}_i}$. Число F називається *свободою вибору*.

Вибрані найкращі часткові описи є входами наступного ряду процедури синтезу. На цьому перша ітерація методу закінчується.

Процес синтезу продовжується, аж поки не буде досягнуто мінімум критерію регулярності $\overline{\varepsilon}^2_{\text{пр}}(k) = \min_i \overline{\varepsilon}^2_{\text{пр}_i}(k)$, $\overline{\varepsilon}^2_{\text{пр}}(k) \le \overline{\varepsilon}^2_{\text{пр}}(k+1)$, де $\overline{\varepsilon}^2_{\text{пр}_i}(k)$ — значення критерію *y i*-ї моделі на *k*-й ітерації.

У такий спосіб визначають оптимальну модель мінімальної складності. Визначивши потрібну (оптимальну) модель, далі процедуру виконують у зворотному напрямі, починаючи з її зв'язків із попереднім рядом. На основі заміни змінних у кінці знаходять шукану модель у вихідних змінних $x_1, x_2, \ldots, x_n$. Наведений метод, на відміну від інших методів ідентифікації моделей складних систем, дає змогу автоматично відновити структуру шуканої моделі. Це забезпечується завдяки використанню полінома Колмогорова — Габора та принципу самоорганізації [17, 18]: зі збільшенням складності моделі *S* (*S* — число членів полінома Колмогорова — Габора) критерій регулярності $\overline{\varepsilon}^2_{\text{пр}}$ спочатку зменшується, досягає мінімуму, а потім залишається сталим (за відсутності шумів) або починає зростати.

Поліном Колмогорова — Габора — це математична конструкція, яка дає змогу наближати багатовимірні функції за допомогою суми функцій від одновимірних аргументів. Її теоретичні основи закладені в роботах радянського математика Андрія Колмогорова (1957 р.) і угорського математика Денеша Габора (1960 р.), котрі займалися проблемою універсальної апроксимації складних багатовимірних функцій. Згідно з теоремою Колмогорова про представлення функцій, будь-яку багатовимірну неперервну функцію можна наближено подати у вигляді суми одновимірних функцій, використовуючи допоміжні функції від однієї змінної. Це дає можливість замінити багатовимірний аналіз набором операцій з одновимірними функціями, що значно спрощує обчислення та апроксимацію.

Поліном Колмогорова — Габора є конкретним способом реалізації цього підходу, де апроксимація багатовимірної функції здійснюється за допомогою комбінацій одновимірних функцій. Теорема Колмогорова та ідея полінома Колмогорова — Габора стали основою для розвитку теорії нейронних мереж, оскільки демонструють, що навіть багатовимірні та складні функції можна наближати за допомогою сум і комбінацій одновимірних функцій, що є аналогом роботи багатошарових нейронних мереж.

Однією з суттєвих властивостей інтелекту людини є її здатність працювати з неповною і нечіткою інформацією в умовах невизначеності. Тому виникла потреба у створенні апарату формалізованого опису нечіткої та якісної інформації для прийняття рішень за умов невизначеності. Така проблема була вирішена Л. Заде, який у 1965 р. ввів поняття нечіткої множини (НЧМ, «*fuzzy set*») і розробив апарат операцій над нечіткими змінними [10].

Основною складовою НЧМ є її функція належності (ФН) $\mu_A(x)$ така, що $\mu_A(x) \in [0;1]$, де $\mu_A(x_i)$ — ступінь належності елемента до нечіткої множини A, що лежить між «0» і «1». У цьому випадку спостерігається принципова відмінність від традиційних множин, для яких ступінь належності елементів дорівнює або «1» (елемент x належить множині A), або «0» (елемент x не належить множині A).

Слід підкреслити, що ФН не є щільністю розподілу випадкової величини А, а ступінь належності $\mu_A(x_i)$— це не ймовірність появи значення x_i, а шанс появи такого значення за оцінкою особи, що приймає рішення і відображає його суб'єктивну оцінку («суб'єктивну імовірність»).

Подальшим розвитком теорії НЧМ стало введене Заде поняття лінгвістичної змінної (ЛЗ) та її використання для ухвалення рішень в умовах якісної та нечіткої інформації.

ЛЗ — це змінна (кількісна або якісна), що задається п'ятіркою кортежів [19]:

$$\text{ЛЗ} = <N, X, T, \mu, G>,$$

де N — ім'я змінної; $X = [x_{\min}, x_{\max}]$ — універсальна шкала; $T = \{T_j\}$ — множина базових термів або значень ЛЗ; μ — оператор, що ставить у відповідність кожному значенню T_j його ФН $\mu_{T_j}(x)$; G — генератор нових термів із базових з використанням приставок: «не», «дуже», «більш-менш».

При цьому ФН нових термів визначають за співвідношенням:

$$\mu_{\text{не}T_j}(x) = 1 - \mu_{T_j}(x);\ \mu_{\text{дуже}T_j}(x) = (\mu_{T_j}(x))^2;\ \mu_{\text{більш-менш}T_j}(x) = \sqrt{\mu_{T_j}(x)}.$$

Приклади

1. Нехай кількісна змінна «Дохід компанії» розглядається як ЛЗ, можливий дохід компанії змінюється в діапазоні $[1; 10^6]$ грн. Введемо базові терми: T_1 — низький, T_2 — низький-середній, T_3 — середній, T_4 — середній-високий, T_5 — високий. Тоді ЛЗ «Дохід компанії» задають так:

$$\text{ЛЗ «Дохід компанії»} = <N, X, \{T_j\}, \{\mu_{T_j}(x)\}, G>,$$

де N — дохід компанії; $X = [1; 10^6]$; $T = \{T_j\}$ — набір базових термів; а їх ФН $\mu_{T_j}(x)$ наведено на рис. 4.

2. Тепер розглянемо якісну змінну. Наприклад, враження від малюнка. На універсальній шкалі $X = [0; 1]$ введемо набір базових термів для ЛП «Враження»: T_1 — потворне, T_2 — неприємне, T_3 — приємне, T_4 — чудове, T_5 — прекрасне (наприклад, враження від «Сікстинської Мадонни» Рафаеля). ФН базових термів наведено на універсальній шкалі на рис. 5. Ці ФН відображають інформацію експерта про якісну змінну «враження», його

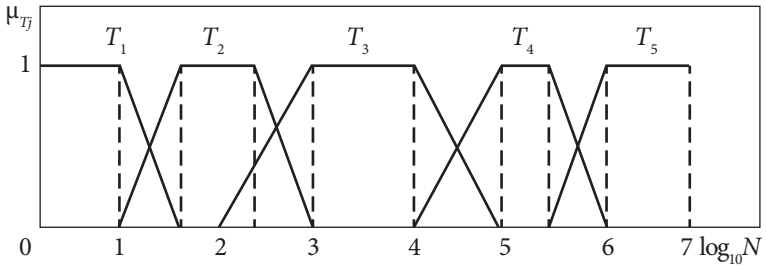


Рис. 4. Функція належності ФН

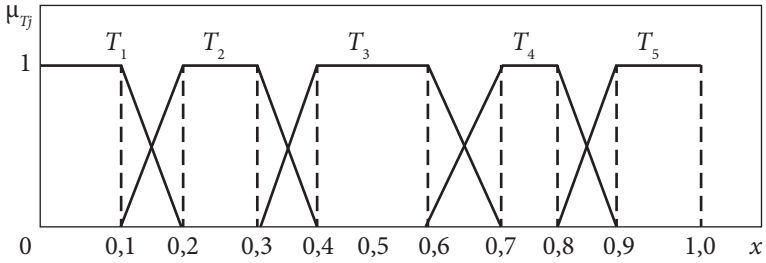


Рис. 5. ФН базових термів на універсальній шкалі

суб'єктивну оцінку відповідності цього змінного ступеня наявності певної ознаки у конкретної особи.

Лінгвістичні змінні використовують для ухвалення рішень в умовах якісної та недостовірної інформації в системах нечіткої логіки. Головним елементом таких систем є нечітка база правил (НБП), що відображає нечіткі знання експерта про відповідну предметну галузь. Нечіткі правила виведення мають такий вигляд:

$$R_1\text{: якщо } x_1 \in A_{11}, x_2 \in A_{21}, \ldots, x_n \in A_{n1}, \text{ то } z \in C_1,$$

$$R_k\text{: якщо } x_1 \in A_{1k}, x_2 \in A_{2k}, \ldots, x_n \in A_{nk}, \text{ то } z \in C_k,$$

де $x_1, x_2, \ldots, x_n$ — вхідні змінні; z — вихідна змінна; $\{A_{ij}\}$ і $\{C_k\}$— значення ЛЗ (термів).

Як наступний приклад розглянемо нечіткі правила в задачі аналізу ризику банкрутства корпорацій в умовах невизначеності [20]. Нехай для оцінки фінансового стану корпорацій використовуються такі фінансові показники: x_1 — коефіцієнт автономності; x_2 — коефіцієнт забезпеченості активів власними коштами; x_3 — коефіцієнт проміжної ліквідності; x_4 — коефіцієнт абсолютної ліквідності; x_5 — коефіцієнт оборотності активів; x_6 — рентабельність усього капіталу.

Ці показники розглядаються як лінгвістичні змінні зі значеннями: дуже низький (ДН), низький (Н), середній (С), високий (В) та дуже висо-

кий (ДВ). Нехай вихідна змінна «ризик банкрутства» набуває аналогічних значень: дуже високий (ДВ), високий (В), середній (С), низький (Н) і дуже низький (ДН).

Тоді можна записати такі правила:

• R_1: якщо x_1 — ДН, x_2 — ДН, x_3 — ДН, x_4 — ДН, x_5 — ДН, x_6 — ДН, то ризик банкрутства ДВ;

• R_k: якщо x_1 — С, x_2 — С, x_3 — С, x_4 — Н, x_5 — ДН, x_6 — ДН, то ризик банкрутства В;

• R_M: якщо x_1 — ДВ, x_2 — ДВ, x_3 — ДВ, x_4 — ДВ, x_5 — ДВ, x_6 — ДВ, то ризик банкрутства ДН.

Логічний висновок отримують відповідно до такої послідовності етапів [21]:

• фазифікація (запровадження нечіткості);

• логічний висновок, що складається з двох підетапів: визначення ступеня виконання умов правил і знаходження виходу кожного правила;

• композиція виходів правил;

• дефазифікація, тобто приведення до чіткості (знаходження чіткого виходу).

Основні алгоритми нечіткого логічного висновку [21]:

• алгоритм Мамдані;

• алгоритм Цукамото;

• алгоритм Сугено;

• алгоритм Ларсена.

Наведемо основні переваги і недоліки СНЛ [18]:

Основні переваги СНЛ:

• Можливість враховувати якісну і нечітку інформацію, працювати в умовах неповноти та невизначеності вихідних даних.

• Можливість враховувати нечіткі знання експертів, наприклад у вигляді бази нечітких правил.

Основні недоліки СНЛ:

• Для роботи СНЛ необхідно задати функції належності (ФН) лінгвістичних змінних у нечітких правилах. Це виконує експерт. Щоб спростити завдання, передбачається, що всі лінгвістичні змінні мають один клас ФН (наприклад, гауссівські, трикутні або трапецієподібні), а ФН різних термів відрізняються лише параметрами, які визначає експерт. Проте експерт може не володіти всією необхідною інформацією для цього.

• База нечітких правил, що формулюється експертом, може бути неповною або суперечливою.

Для усунення зазначених недоліків необхідно зробити СНЛ адаптивними і навчати їх на основі вхідної вибірки. Щоб використовувати накопичений арсенал методів і алгоритмів навчання, СНЛ реалізують у вигля-

ді нечітких нейронних мереж. У таких мережах є шар вхідних нейронів X_1, X_2, …, X_n, шар нейронів правил R_1, R_2, …, R_k та шар вихідних нейронів C_1, C_2, …, C_m. Зв'язки між шарами є нечіткими і задаються функціями належності з параметрами, що налаштовуються в процесі навчання, для чого використовують алгоритми навчання, розроблені для звичайних нейронних мереж. Отже, інтеграція технологій нейронних мереж і систем із нечіткою логікою дала змогу створити нову гібридну технологію — нечіткі нейронні мережі, що поєднують переваги обох підходів. Системи нечіткої логіки та нечіткі нейронні мережі (ННМ) детально розглянуто в розділах 3—5.

Системи нечіткої логіки та ННМ ефективно застосовують у різних задачах обчислювального інтелекту. Їх широке використання базується на теоремі *FAT (Fuzzy Approximation Theorem)* про універсальну апроксимацію для систем нечіткої логіки. Китайський дослідник у галузі нечіткої логіки Лі-Ксін Ванг (Li-Xin Wang) у своїй роботі «*Fuzzy Systems are Universal Approximators*» (1992 р.) довів, що системи нечіткої логіки з гауссівськими функціями належності є універсальними апроксиматорами, тобто здатні апроксимувати будь-яку неперервну функцію зі збільшенням кількості правил. Його роботи в цій сфері показали, що нечіткі системи можна налаштовувати для високоточної апроксимації, що зробило їх корисними для завдань прогнозування, класифікації та управління складними системами. Дещо пізніше (1993 р.) американський вчений з університету Південної Каліфорнії Барт Коско (Bart Kosko) показав, що нечіткі системи з трикутними функціями належності також є універсальними апроксиматорами. Його висновки доводять, що такі системи можуть наближати будь-які неперервні функції за відповідних умов, що розширило застосування нечіткої логіки в інженерії, економіці та інших сферах [23]. Ці результати про універсальну апроксимацію для звичайних нейронних мереж свідчать про глибокий зв'язок між властивостями звичайних і нечітких систем розпізнавання.

Однією з властивостей живих організмів, зокрема людського мозку, є здатність до саморозвитку, адаптації до змін середовища та самовдосконалення в процесі еволюції. Біологічна еволюція ґрунтується на таких механізмах:

- схрещування батьківських пар і передання генів нащадкам від обох батьків;
- мутації, спричинені випадковими чинниками;
- селекція — відбір кращих особин, гени яких переходять у наступне покоління.

Ці процеси повторюються багаторазово, сприяючи поступовому поліпшенню особин. Такі механізми було використано в еволюційному моделюванні (ЕМ) — важливій технології обчислювального інтелекту. Технологія

ґрунтується на використанні генетичних алгоритмів (ГА), які відтворюють механізми еволюції. ГА вирізняються методами вибору батьківських пар, кросовера, механізмами мутацій та методами селекції. Основна ідея ЕМ під час вирішення цього типу завдань полягає в заміні процесу побудови складної системи (наприклад, для розпізнавання образів або технічної діагностики) її еволюційним розвитком. Цей процес відбувається у прискореному масштабі часу, і для досягнення бажаного результату можуть знадобитися тисячі поколінь.

ЕМ також широко застосовують для структурного синтезу систем розпізнавання образів і класифікації, медичної діагностики, прогнозування та управління складними об'єктами і процесами. Мутації забезпечують різноманітність особин і унеможливлюють виродження популяцій. Зазначені технології ШІ широко використовують відповідні методи й алгоритми. Наприклад, нейронні мережі (НМ) та штучні нейронні мережі (ШНМ) застосовують алгоритми навчання та адаптації, зокрема градієнтний метод, метод сполучених градієнтів, стохастичну апроксимацію, рекурентний метод найменших квадратів (РМНК), сучасні алгоритми навчання Стефана Качмаржа (Stefan Kaczmarz), Дональда Марквардта (Donald Marquardt) та ін. Два останні методи є потужними інструментами для навчання нейронних мереж і застосовуються залежно від розміру завдання та вимог до точності й швидкості. Вони ґрунтовані на різних підходах до мінімізації похибки та корекції параметрів моделей і використовуються переважно для коригування вагових коефіцієнтів у нейронних мережах та інших системах машинного навчання.

Крім того, нейронні мережі використовують методи самонавчання і самоорганізації, такі як метод Кохонена, кластерний аналіз, метод *k*-середніх, метод Уорда, алгоритм «найближчого сусіда», нечіткі методи кластерного аналізу (наприклад, метод *k*-середніх, метод Густавсона — Кесселя), а також адаптивні нечіткі методи кластерного аналізу. Висвітленню цих методів присвячено наступні розділи підручника.

До перспективних напрямів ШІ також належить ройовий інтелект (*swarm intelligence*), що визначає колективну поведінку децентралізованих самоорганізованих систем і є сукупністю методів оптимізації. Термін був запроваджений у 1989 р. Джерардо Бені (Gerardo Beni) та Ван Цзіном (Jing Wang) у контексті досліджень систем клітинних роботів.

Системи ройового інтелекту зазвичай складаються з багатьох агентів (мультиагентна система), які локально взаємодіють один з одним та з навколишнім середовищем. Хоча кожен агент є відносно простим, їх взаємодія спричинює появу ройового інтелекту. Прикладом такої поведінки в природі може бути колонія мурах, рій бджіл, зграя птахів або риб. Нижче наведено приклади алгоритмів ройового інтелекту, які належать до кла-

су метаевристик і можуть мати варіації для одно- та багатокритеріальної оптимізації:

- алгоритми оптимізації колоній мурах (*ant colony optimization*);
- алгоритми оптимізації рою частинок (*particle swarm optimization*);
- бджолиний алгоритм (*bee's algorithm*);
- оптимізація пересування бактерій у пошуках поживних речовин (*bacterial foraging optimization*);
- стохастичний дифузійний пошук (*stochastic diffusion search*);
- світляковий алгоритм (*firefly algorithm*).

Технології і методи ШІ широко застосовують для вирішення багатьох прикладних задач. Наприклад, НМ і ШНМ, метод МГУА широко використовують для прогнозування нестаціонарних часових процесів в економіці та фінансовій сфері; НМ і МГУА — для класифікації та розпізнавання образів, технічної та медичної діагностики; нейронні мережі з самоорганізацією та методи кластерного аналізу — для автоматичної класифікації об'єктів за їхніми ознаками. ЕМ і ГА широко використовують для структурного синтезу складних систем розпізнавання образів та оптимізації інформаційно-комп'ютерних мереж.

Системи з нечіткою логікою та ШНМ ефективно застосовують для аналізу фінансового стану корпорацій, прогнозування ризику банкрутства, оцінки кредитоспроможності позичальників за умов невизначеності.

Отже, сучасні технології, алгоритми і методи ШІ та ОІ тісно взаємодіють, відбувається їх глибоке взаємопроникнення. Загалом вони відображають і технічно реалізують властивості й здібності людського інтелекту в різних галузях практичної діяльності.

СПИСОК ЛІТЕРАТУРИ

1. Haykin S. Neural Networks and Learning Machines. 3rd ed. Prentice Hall., 2008. 936 p.

2. Ivakhnenko A. G. Self-Organizing Methods in Modeling: GMDH Type Algorithms. Marcel Dekker, 1988. 327 p.

3. Klir G. J., & Yuan, B. Fuzzy Sets and Fuzzy Logic: Theory and Applications. Prentice Hall., 1995. 592 p.

4. Zadeh L. A. The concept of a linguistic variable and its application to approximate reasoning. *Information Sciences*. 1975. Vol. 8. P. 199—249.

5. Kaplan R. S., & Norton, D. P. The Balanced Scorecard: Translating Strategy into Action. Harvard Business Press, 1996. 322 p.

6. Everitt B. S., Landau S. Cluster Analysis. 5th ed. Wiley, 2011. 346 p.

7. Zgurovsky M., Zaychenko Y. The Fundamentals of Computational Intelligence: System Approach. Springer, 2016. 375 p.

8. Schalkoff R. J. Intelligent Systems: Principles, Paradigms, and Pragmatics. Jones & Bartlett, 2011. 608 p.

9. Alippi C. Neural Networks and Genetic Algorithms. Springer, 1997. 268 p.

10. Zadeh L. A. Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*. 1978. No. 1. P. 3—28.

11. Zadeh L. A. Fuzzy logic, neural networks, and soft computing. *Communications of the ACM*. 1994. No. 37(3). P. 77—84.

12. Rosenblatt F. Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms. Spartan Books, 1962. 460 p.

13. Müller J. A., Lemke F. Self-Organising Data Mining: An Intelligent Approach to Extract Knowledge from Data. Springer, 2000. 322 p.

14. Minsky M., Papert S. Perceptrons: An Introduction to Computational Geometry. MIT Press, 1969. 258 p.

15. Bodyanskiy Y., Zaychenko Y., Hamidov G. Hybrid Deep Learning Networks Based on Self-Organization and their Applications. Cambridge Scholars Publishing, 2024. 280 p.

16. Goodfellow I., Bengio Y., Courville A. Deep Learning. MIT Press, 2016. 775 p.

17. Ross T. J. Fuzzy Logic with Engineering Applications. 3rd ed. Wiley, 2010. 606 p.

18. Dubois D., Prade H. The three semantics of fuzzy sets. *Fuzzy Sets and Systems*. 1997. No. 90(2). P. 141—150.

19. Niven P. R. Balanced Scorecard Step-by-Step: Maximizing Performance and Maintaining Results. 2nd ed. Wiley, 2006. 384 p.

20. Kaufman L., Rousseeuw P. J. Finding Groups in Data: An Introduction to Cluster Analysis. Wiley, 2005. 342 p.

21. Zadeh L. A. Fuzzy sets. *Information and Control*. 1965. No. 8(3). P. 338—353.

22. Kosko B. Fuzzy systems as universal approximators. *IEEE Transactions on Computers*. 1994. No. 43(11). P. 1329—1333.

23. Ivakhnenko A. G. The method of self-organization of models of complex systems. *European Journal of Operational Research*. 1995. No. 82(3). P. 553—561.

РОЗДІЛ 1

НЕЙРОННІ МЕРЕЖІ

1.1. Нейронна мережа зі зворотним поширенням. Архітектура, функції

1.2. Градієнтний метод навчання нейронної мережі зі зворотним поширенням

1.3. Градієнтний алгоритм навчання для мережі з довільним числом шарів

1.4. Удосконалення градієнтного алгоритму навчання нейронної мережі ВР

1.5. Прискорення збіжності алгоритмів навчання нейронних мереж. Алгоритм спряжених градієнтів

1.6. Генетичний алгоритм навчання нейронної мережі

1.7. Нейронні мережі з радіально-базисними функціями (РБФ-мережі)

Висновки

Список літератури

1.1. Нейронна мережа зі зворотним поширенням. Архітектура, функції

Нейронна мережа зі зворотним поширенням (*Back Propagation*) є алгоритмом навчання багатошарових нейронних мереж, який використовують для коригування ваг за допомогою обчислення градієнта функції помилки. Цей алгоритм використовує градієнтний спуск, щоб мінімізувати функцію помилок. Після того, як дані проходять через мережу в прямому напрямку і обчислюються вихідні дані, алгоритм «поширюється в зворотному напрямку» для коригування ваг, зменшуючи помилку. Структура мережі складається з вхідного шару, прихованих шарів (одного чи декількох) та вихідного шару. Обчислення градієнта ґрунтується на ланцюжковому правилі для функцій багатьох змінних.

Метод *Back Propagation* є основою для сучасних алгоритмів глибокого навчання. Ідея нейронних мереж виникла в середині XX століття, але алгоритм зворотного поширення набув значного розвитку в 1980-х рр. Початкові згадки про градієнтні принципи навчання було висвітлено в роботах дослідників у 1940-х рр., але практичні застосування з'явилися лише з розвитком обчислювальної техніки та алгоритмів.

Американський вчений *Пол Вербос* (Paul Werbos) вперше дослідив цю концепцію в 1974 р., коли він розвинув фундаментальні ідеї зворотного поширення у своїй докторській дисертації. Девід Паркер (David Parker) з Великої Британії зробив подальший внесок у 1982 р., досліджуючи різні аспекти алгоритмів навчання. Джон Хопфілд (John Hopfield, США) та інші дослідники вивчали властивості багатошарових мереж упродовж 1980-х рр. Джеффрі Хінтон (Geoffrey Hinton) разом з Девідом Румельхартом (David Rumelhart) і Рональдом Вільямсом (Ronald Williams) (усі зі США) у 1986 р. опублікували знакову статтю, яка значно розширила розуміння та практичне застосування алгоритму.

Гнучкість алгоритму дає змогу застосовувати його для широкого кола завдань, зокрема класифікації, регресії, розпізнавання зображень та мовлення. Він є достатньо ефективним, щоб підтримувати розвиток глибокої нейронної мережі, і підтримує різні архітектури, що уможливлює його за-

стосування до різних багаторівневих мереж із будь-якою кількістю прихованих рівнів. Однак метод має низку недоліків:

• для нього характерною є проблема зникнення градієнта, коли в глибоких мережах значення градієнта можуть ставати дуже малими, сповільнюючи або зупиняючи навчання на попередніх рівнях;

• метод характеризується обчислювальною складністю, через яку навчання може потребувати багато часу, особливо для мереж із багатьма параметрами;

• метод чутливий до гіперпараметрів, таких як швидкість навчання, що впливає на його ефективність; а також вимоги до великих обсягів даних, щоб запобігти переобладнанню.

Для усунення недоліків алгоритму зворотного поширення було розроблено групу спеціальних оптимізованих методів, які забезпечують адаптивне регулювання швидкості навчання. Інші способи боротьби з проблемою зникнення градієнта полягають у використанні функцій активації і методів нормалізації, наприклад таких, як пакетна нормалізація. Зворотне поширення стало наріжним каменем сучасних досягнень глибокого навчання, заклавши основу для технологій розпізнавання зображень, голосових помічників та інших програм ШІ.

Далі покажемо, що зворотне поширення швидше належить до алгоритмів навчання, ніж до мережевої архітектури. Таку мережу правильніше називати мережею з прямим передаванням сигналів [1.1, 1.2]. Класичну трирівневу архітектуру Н-мережі наведено на рис. 1.1. Позначимо: R^d — d-вимірний простір. Вхідний вектор $X = \{x_1, x_2, \ldots, x_N\}$; вихідний вектор $Y = \{y_1, y_2, \ldots, y_M\}$. Нейронна мережа виконує функціональне перетворення, яке може бути подано як $Y = F(X)$, де $X = \{x_i\}$, $i = \overline{1, N}$; $Y = \{y_k\}$, $k = (\overline{1, M})$.

Прихований шар насправді може складатися з декількох шарів, проте можна вважати, що достатньо розглядати лише три шари для опису цього типу поведінки. Для нейронної мережі з N вхідними вершинами, H вершинами прихованого шару і M вихідними вершинами вихідні величини задаються рівнянням

$$y_k = g(\Sigma_{j=1}^{J}, W_{jk}^0 h_j), \quad k = \overline{1, M}. \tag{1.1}$$

де W_{jk}^0 — вихідна вага зв'язку від вершини j прихованого шару до вершини k вихідного шару; g — функція активації нейронів (яку буде визначено пізніше), що виконує відображення $R^j \to R^1$.

Вихідні сигнали вершин прихованого шару h_j, $j = 1, 2, \ldots, H$, задаються рівнянням:

$$h_j = \sigma\,(\Sigma_{i=1}^{N}, W_{ij}^I x_i + W_j^T), \quad j = \overline{1, J}. \tag{1.2}$$

де W_{ij}^I — вхідна вага зв'язку (i, j); W_j^T — величина порога (вага від вузла, який має постійний сигнал, що дорівнює 1, до вузла j); x_i — сигнал на виході i-го вхідного вузла; σ — так звана функція «сигмоїд», що задається таким рівнянням:

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{1.3}$$

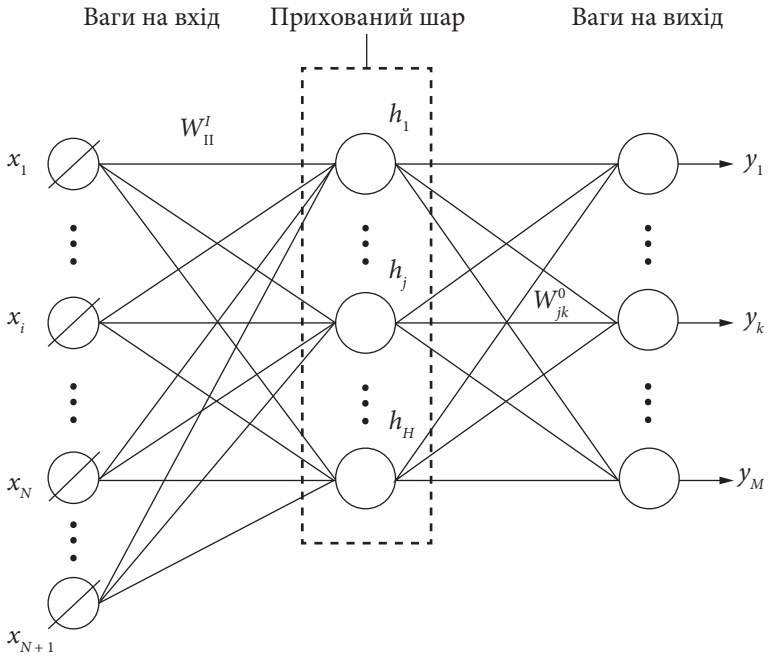


Рис. 1.1. Класична трирівнева архітектура Н-мережі

Функція g у рівнянні (1.1) може бути: а) сигмоїдальною $\sigma(x)$; б) $\tanh(x)$; в) релейною; г) релейно-пропорційною або іншою нелінійною функцією. У нашому викладі прийматимемо g або функцією виду $\sigma(x)$, або одиничною функцією, тобто релейною. Необхідно, щоб функція активації була нелінійною і мала обмежений вихід, тобто була обмеженою. Графіки функцій $\sigma(x)$, $\text{ReLU}(x)$ та $\tanh(x)$ наведено на рис. 1.2.

Нейронні мережі прямої дії. Дія нейронної мережі прямої дії визначається двома факторами:

- архітектурою мережі;
- величинами ваги.

Кількість вхідних і вихідних вузлів визначається апріорі і є фіксованою. Число прихованих вузлів є змінним і може налаштовуватися (регулюватися) користувачем. До цього часу це налаштування залишається поки «мистецтвом», хоча було запропоновано різні методи визначення кількості шарів та прихованих вузлів. Після визначення (завдання) архітектури мережі саме значення ваги визначають її поведінку.

Кажуть, що мережа «навчається», якщо ваги змінюються так, щоб досягти бажаної мети. Тут слід мати на увазі, що термін «навчання», запозичений з біологічної термінології, в мережі означає налаштування множини параметрів.

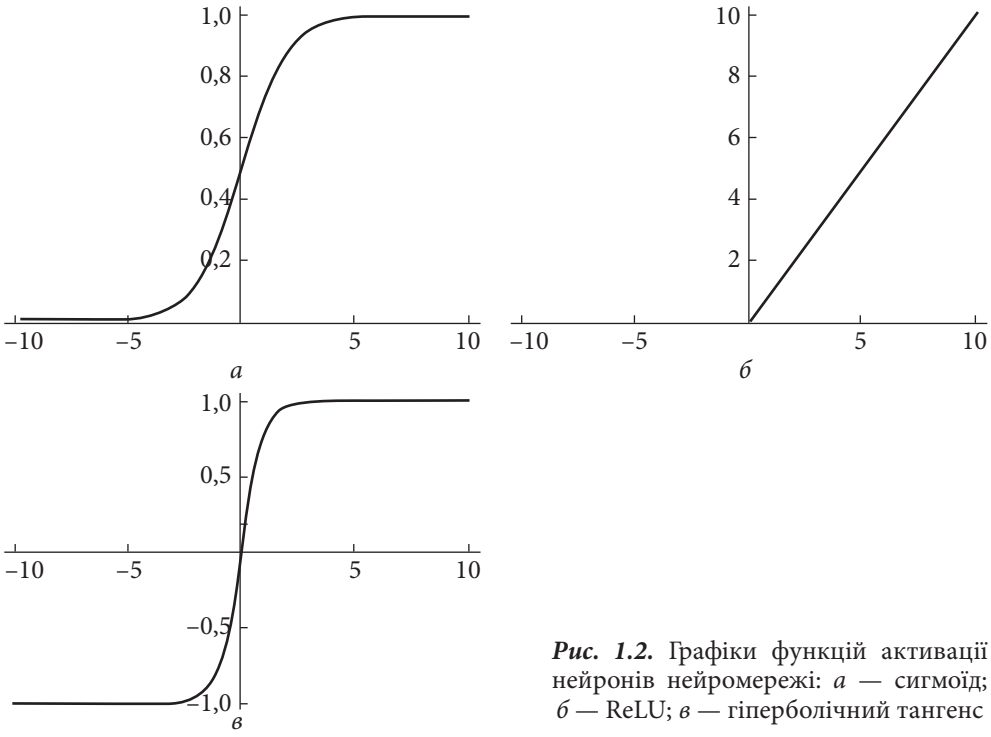


Рис. 1.2. Графіки функцій активації нейронів нейромережі: *а* — сигмоїд; *б* — ReLU; *в* — гіперболічний тангенс

Теорема про універсальну апроксимацію. Нейронна мережа *Back Propagation (НМ ВР)* є універсальним апроксиматором, для якого в 1989 р. *Курт Горнік* (Kurt Hornik) та інші вчені, відомі своїми дослідженнями в галузі нейронних мереж, довели теорему про універсальну апроксимацію [1.18]. Ця теорема є узагальненням теореми Колмогорова та Арнольда і стверджує, що багатошарові нейронні мережі за певних умов можуть апроксимувати будь-яку неперервну функцію. Більш строго вона формулюється так [1.18].

Нехай ϕ(.) — обмежена, непостійна, монотонно зростаюча неперервна функція; I_{m_0} — вимірний одиничний гіперкуб $[0,1]^{m_0}$, а простір неперервних на I_{m_0} функції позначається символом $C(I_{m_0})$.

Тоді для будь-якої функції $f \in C(I_{m_0})$ та довільного $\varepsilon > 0$ існує таке ціле число m_1 і множина дійсних констант $<a_i, b_i, w_{ij}>$, де $i = 1, \ldots, m_i; j = 1, \ldots, m_0$, що функція

$$F(x_1, x_2, \ldots, x_{m_0}) = \sum_{i=1}^{m_1} \alpha_i \varphi \left(\sum_{i=1}^{m_1} w_{ij} x_i + b_j\right) \tag{1.4}$$

є апроксимацією функції *f*, тобто набуває вигляду

$$\left|F(x_1, x_2, \ldots, x_{m_0}) - f(x_1, x_2, \ldots, x_{m_0})\right| < \xi, \tag{1.5}$$

для всіх $x_1, x_2, \ldots$, що належать вхідному простору.

Наведену теорему застосовують безпосередньо до мереж *Back Propagation* (багатошарові перцептрони) з таких причин:

1. Як функцію активації використовують обмежену монотонно зростаючу функцію «сигмоїд» $\varphi(x) = \frac{1}{1+e^{-x}}$.

2. НМ містить m_0 вхідних вузлів і один прихований шар, що складається з m_1 нейронів.

3. Нейрон прихованого шару i має синаптичні ваги $w_1, w_2, \ldots, w_{m_0}$ та поріг b_i.

4. Вихід нейронної мережі є лінійною комбінацією вихідних сигналів прихованих нейронів, зважених з вагами $\alpha_1, \alpha_2, \ldots, \alpha_m$.

Отже, теорема про універсальну апроксимацію є теоремою існування (*existence theorem*), тобто математичним доказом можливості апроксимації будь-якої неперервної функції за допомогою *НМ BP* навіть із одним прихованим шаром. Разом з тим, це теорема не дає відповіді на запитання, як побудувати мережу. Для виконання цього завдання у кожному конкретному випадку за заданої вибірки входів $\{X_i\}$ та бажаних виходів $\{d_i\}$, $\{(X_1, d_1), (X_2, d_2), \ldots, (X_n, d_n)\}$, необхідне навчання мережі з використанням одного з відомих алгоритмів.

1.2. Градієнтний метод навчання нейронної мережі зі зворотним поширенням

Першим алгоритмом навчання, розробленим для нейронної мережі зі зворотним поширенням (*Back Propagation, BP*) був градієнтний метод навчання. Нехай критерій навчання мережі, що має три шари (один прихований шар), задається формулою:

$$E(w) = \sum_{i=1}^{M} (d_i - y_i(w))^2 \to \min, \tag{1.6}$$

де d_i — бажане значення i-го виходу нейронної мережі; $y_i(w)$ — фактичне значення i-го виходу нейронної мережі *BP* для вагової матриці $W = [W^1; W^0]$.

Тобто критерій $E(w_j)$ становить собою середній квадрат похибки апроксимації. Нехай функції активації для нейронів прихованого шару [1.2]:

$$h_j = \sigma\left(\sum_{i=1}^{N}, x_i W_{ij}^I + W_{N+1,j}^I\right)$$

і нейронів вихідного шару [1.2]

$$y_k = \sigma\left(\sum_{j=1}^{J}, h_i W_{jk}^0\right)$$

є однаковими і сигмоїдальними:

$$\sigma(x) = \frac{1}{1+e^{-x}}.$$

Для такої функції похідна дорівнює:

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)).$$

Розглянемо градієнтний алгоритм навчання нейронної мережі.

1. Нехай $W(n)$ — поточне значення матриці ваги. Алгоритм має такий вигляд [1.6]:

$$W(n+1) = W(n) - \gamma_{n+1} \nabla_w E(W(n)),$$

де γ_n — розмір кроку на n-й ітерації, $\nabla_w E(W(n))$— градієнт $E(W(n))$.

2. На кожній ітерації спочатку навчаємо (коригуємо) вхідні ваги:

$$w_{ij}^I(n+1) = w_{ij}^I(n) - \gamma_{n+1} \frac{\partial E(W)}{\partial w_{ij}^I}; \tag{1.7}$$

$$\frac{\partial E(W)}{\partial w_{ij}^I} = \sum_{k=1}^{M} 2(d_k - y_k(W^0))\, y_k(W^0)(1 - y_k(W^0))\, h_j(W)(1 - h_j(W)) \cdot x_i.$$

3. Знаходимо (навчаємо) вихідні ваги [1.6]:

$$\frac{\partial E(W^0)}{\partial w_{ik}^0} = -2(d_k - y_k(W^0))\, y_k(W^0)(1 - y_k(W^0))\, h_j;$$

$$w_{ik}^0(n+1) = w_{ik}^0(n) - \gamma_{n+1} \frac{\partial E(W)}{\partial w_{jk}^0}, \tag{1.8}$$

де x_i, $i = \overline{1, N+1}$ — входи НМ; y_k, $k = \overline{1, M}$ — виходи НМ; h_j, $k = \overline{1, J}^{h_j}$ — виходи прихованого шару.

4. $n := n + 1$ і переходимо на наступну ітерацію.

Зауваження. Так званий алгоритм навчання з пам'яттю має вигляд [1.1, 1.6]:

$$W(n+1) = W(n) - \eta[(1 - \alpha) \nabla_w E(W(n)) + \alpha \nabla_w E(W(n-1))], \tag{1.9}$$

де η — швидкість навчання; $\alpha \in [0; 1]$ — параметр забування.

Отже, градієнтний метод є першим запропонованим алгоритмом навчання. Він простий у реалізації, однак має певні недоліки [1.1, 1.2]:

- має повільну збіжність;
- знаходить лише локальний екстремум.

1.3. Градієнтний алгоритм навчання для мережі з довільним числом шарів

Виконаємо виведення рекурентного співвідношення в алгоритмі *BP* для обчислення похідних помилки довільного шару. Розглянемо градієнтний метод навчання нейронної мережі *BP* із довільною кількістю шарів N. Нехай потрібно мінімізувати критерій

$$E(w) = \frac{1}{2} \sum_p \sum_j (y_{jp}^N - d_{jp})^2, \tag{1.10}$$

де y_{jp}^N — реальний вихід j-го нейрона вихідного шару N нейронної мережі при поданні на вхід p-го образу; d_{jp} — бажаний вихід.

Мінімізація здійснюється методом градієнтного спуску, що означає налаштування ваг у такий спосіб [1.6, 1.11]:

$$w_{ij}(t) = w_{ij}(t-1) + \Delta w_{ij};$$
$$\Delta w_{ij}^{N} = -\eta \frac{\partial E}{\partial w_{ij}^{(n)}}, \tag{1.11}$$

де w_{ij} — ваговий коефіцієнт зв'язку i-го нейрона $(N-1)$-го шару із j-м нейроном n-го шару; $0 < \eta < 1$ коефіцієнт швидкості навчання.

Далі:

$$\frac{\partial E}{\partial w_{ij}^{(n)}} = \frac{\partial E}{\partial y_j^{(n)}} \frac{dy_j^{(n)}}{ds_j^{(n)}} \frac{\partial s_j^{(n)}}{\partial w_{ij}^{(n)}}, \tag{1.12}$$

де y_j — вихід j-го нейрона n-го шару; s_j — зважений сумарний вихідний сигнал j-го нейрона.

Очевидно, що

$$s_j^{(n)} = \sum_{i=1}^{I} w_{ij}^{(n)} y_i^{(n-1)}. \tag{1.13}$$

Якщо функція активації j-го нейрона має вигляд

$$y_j = f(s_j),$$

то

$$\frac{dy_j}{ds_j} = f'(s_j).$$

В окремому випадку, якщо $f = \sigma$ — «сигмоїд», то маємо:

$$\frac{\partial y_j}{\partial s_j} = f'(s_j) = y_j(1 - y_j). \tag{1.14}$$

Третій співмножник (1.12) має вигляд $\dfrac{\partial s_j^{(n)}}{\partial w_{ij}^{(n)}} = y_i^{(n-1)}$.

Що ж до першого співмножника (1.12) $\dfrac{\partial E}{\partial y_j}$, то він легко розкладається через виходи нейронів наступного $(n+1)$-го шару у такий спосіб [1.1, 1.6]:

$$\frac{\partial E}{\partial y_j^{(n)}} = \sum_k \frac{\partial E}{\partial y_k^{(n+1)}} \frac{dy_k^{(n+1)}}{ds_k^{(n+1)}} \frac{\partial s_k^{(n+1)}}{\partial y_k^{(n)}} = \sum_k \frac{\partial E}{\partial y_k^{(n+1)}} \frac{dy_k^{(n+1)}}{ds_k^{(n+1)}} w_{jk}^{(n+1)}. \tag{1.15}$$

Ввівши нову змінну, отримаємо рекурентні формули для розрахунку величини $\delta_j^{(n)}$ n-го шару через величини $\delta_j^{(n+1)}$ наступного шару (рис. 1.3):

$$\delta_j^{(n)} = \frac{\partial E}{\partial y_j^{(n+1)}} \frac{dy_j^{(n)}}{ds_j^{(n)}}, \tag{1.16}$$

$$\delta_j^{(n)} = \sum_{k=1}^{K} \delta_j^{(n+1)} w_{jk}^{(n+1)} \frac{dy_j^{(n)}}{ds_j^{(n)}}. \tag{1.17}$$

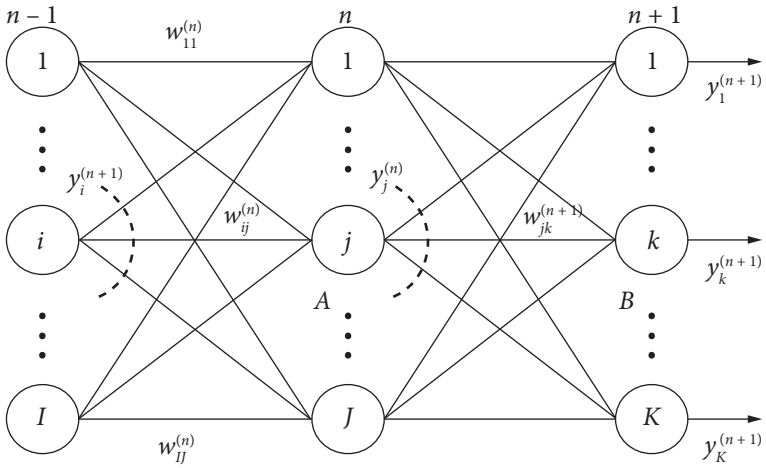


Рис. 1.3. Рекурентна формула для розрахунку величини $\delta_j^{(n)}$ n-го шару через величини $\delta_j^{(n+1)}$ наступного шару

Для вихідного шару $n = N$ маємо [1.1, 1.6]:

$$\delta_i^{(N)} = (y_i^{(N)} - y_i)\frac{dy_i^{(N)}}{ds_i^{(N)}}. \tag{1.18}$$

Тепер можна записати алгоритм градієнтного спуску (1.12) у вигляді:

$$\Delta w_{ij}^{(N)} = -\eta\delta_j^{(n)} y_i^{(n-1)}. \tag{1.19}$$

Іноді для надання процесу корекції ваги деякої інерційності з метою згладжування різких стрибків під час руху по поверхні використовується значення Δw_{ij} на попередній ітерації $(t-1)$. У такому разі $\Delta w_{ij}(t)$ має вигляд:

$$\Delta w_{ij}^{(n)}(t) = \eta(\mu\Delta w_{ij}^{(n)}(t-1) - (1-\mu)\delta_j^{(n)} y_i^{(n-1)}(t)), \quad \mu \in [0; 1]. \tag{1.20}$$

Градієнтний алгоритм навчання для мережі з довільним числом шарів. Повний алгоритм навчання НМ за допомогою процедури зворотного поширення складається з таких кроків.

Нехай на входи НМ подано один із можливих образів: $x = \{x_i\}_{i=\overline{1,I}}$.

1. Припустимо, що $y_i^{(0)} = x_i$, $i = \overline{1, I}$.

2. Розраховуємо послідовно значення виходів для n-го шару ($n = 1, 2, \ldots, N$):

$$s_j^{(n)} = \Sigma_{i=1}^{I} y_i^{(n-1)} w_{ij}^{(n)}, \tag{1.21}$$

$$y_j^{(n)} = f(s_j^{(n)}). \tag{1.22}$$

3. Розраховуємо $\delta_i^{(N)}$ для нейронів вихідного шару за формулою (1.18). Визначаємо $\Delta w_{jk}^{(N)}$.

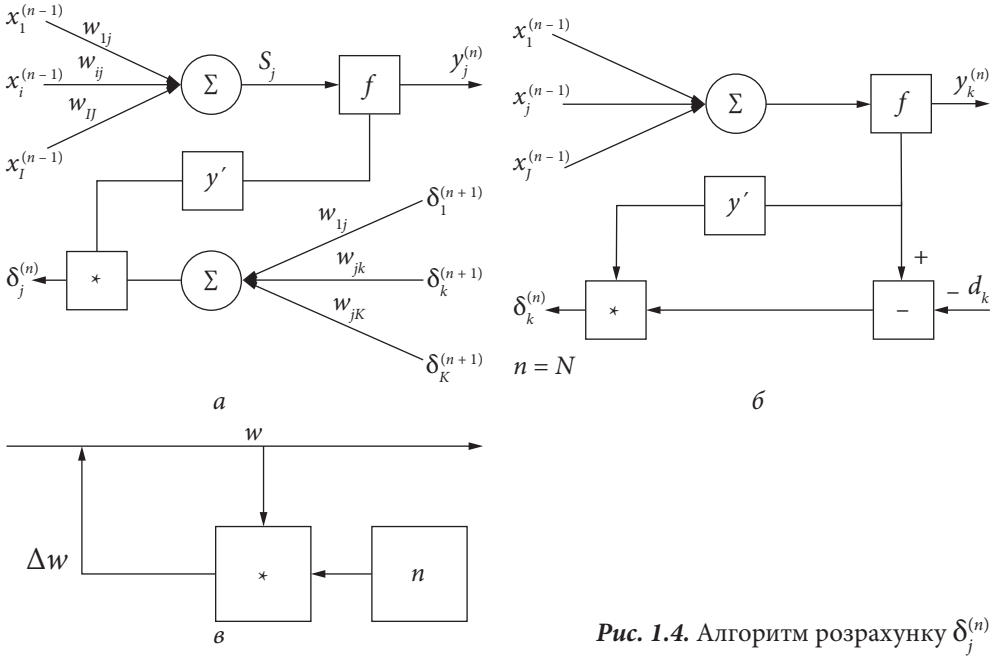


Рис. 1.4. Алгоритм розрахунку $\delta_j^{(n)}$

4. Використовуючи рекурентну формулу (1.17), розраховуємо $\delta_k^{(n)}$ через $\delta_k^{(n+1)}$ і $\Delta w_{ij}^{(n)}$ для всіх попередніх шарів $n = N - 1, N - 2, \ldots, 1$ за формулою (1.19).

5. Коригуємо ваги в НМ згідно з процедурою:

$$w_{ij}^{(n)}(t) = w_{ij}^{(n)}(t-1) + \Delta w_{ij}^{(n)}(t). \qquad (1.23)$$

На цьому перша ітерація (t) закінчується.

6. Обчислюємо $E = E(w(t))$.

Якщо $E(w(t)) < \bar{\varepsilon}^2$, то — СТОП. Інакше переходимо на крок 1 ($t+1$)-ї ітерації.

Алгоритм розрахунку $\delta_j^{(n)}$ проілюстровано на рис. 1.4.

1.4. Удосконалення градієнтного алгоритму навчання нейронної мережі ВР

У процесі реалізації градієнтного алгоритму навчання нейронної мережі *ВР* можуть виникнути складнощі, властиві градієнтним алгоритмам оптимізації.

1. Якщо стартова точка знаходиться далеко від точки мінімуму функції $E(w_{ij})$, то необхідно рухатися з малим кроком і процес пошуку може затягуватися. Для його прискорення має сенс збільшити величину кроку. Ознакою такої ситуації є сталість знака $\Delta E(t-1) < 0$, $\Delta E(t) < 0$ (рис. 1.5, *а*) [1.1, 1.6].

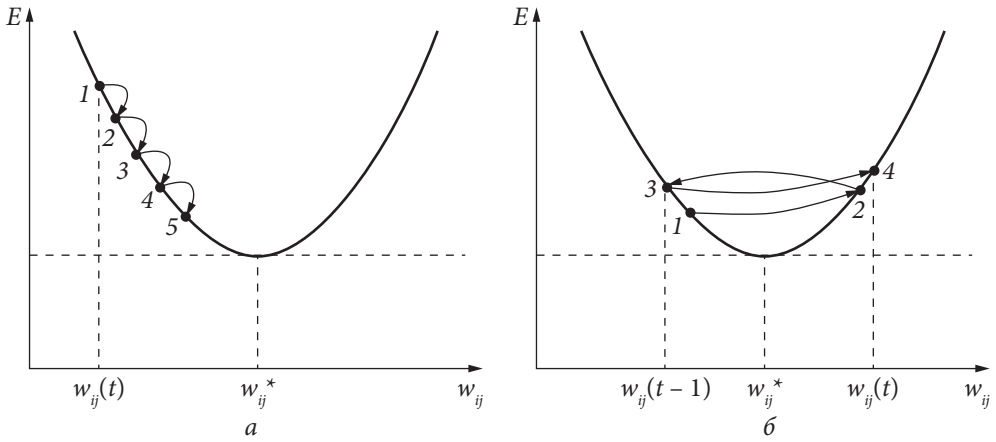


Рис. 1.5. Ілюстрація роботи алгоритму градієнтного спуску за різної величини кроку

2. Якщо стартова точка знаходиться на околиці точки мінімуму і величина кроку велика, то відбудеться «перескакування» через точку і виникає явище «осциляції». У цьому разі доцільно поступово зменшувати $\eta(t)$. Ознакою такої ситуації є зміна знака ΔE, тобто $\Delta E(t-1) \cdot \Delta E(t) < 0$ (рис. 1.5, *б*).

Градієнтний метод із корекцією кроку навчання (метод відкату). Для подолання зазначених труднощів, пов'язаних з використанням градієнтного методу, було розроблено градієнтний метод з корекцією кроку навчання.

Тут величина кроку $\eta(t)$ на $(t+1)$-й ітерації описується рекурентним виразом [1.6]:

$$\eta(t+1) = \begin{cases} \eta(t) \cdot u, \text{ якщо } \Delta E(t-1) \cdot \Delta E(t) > 0, \text{ за умови } \Delta E(t-1) < 0; \\ \eta(t) \cdot d \text{ в інакшому випадку,} \end{cases} \quad (1.24)$$

де $u > 1$; $0 < d < 1$.

Рекомендується вибирати u, $d \approx 1$. Корекцію кроку можна здійснювати, якщо виконано кілька послідовних кроків, наприклад $t-2$, $t-1$.

Метод з вибиванням із локальних мінімумів (*shock BP*). Цей метод використовують у разі багатоекстремальної залежності $E(w)$ за умови потреби пошуку глобального мінімуму (рис. 1.6).

У випадку, якщо ми «застрягли» в локальному мінімумі w_{ij}^* і помилка E упродовж тривалого часу не змінюється, то має сенс зробити великий крок у випадковому напрямку ξ, щоб «вискочити» з цієї «пологої пастки» і потрапити у сферу притягання іншого мінімуму w_{ij}^0. Тоді

$$w_{ij}(t+1) = w_{ij}(t) + x\xi,$$

де ξ рівномірно розподілена в інтервалі $[-1; +1]$; x — розмір кроку.

Метод із векторним кроком навчання (*Super SAB*). Основний недолік класичного градієнтного методу полягає в тому, що він використовує однаковий крок навчання $\eta(t)$ у всіх напрямках. Тобто метод не враховує тієї обставини, що за різними компонентами w_{ij} ми можемо знаходитися на різній відстані від точки мінімуму (тобто за одними компонентами далеко, а за іншими — близько).

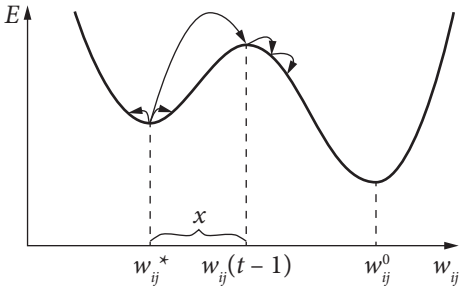


Рис. 1.6. Пошук глобального мінімуму

Тому дослідники з Португалії *Луїс Б. Алмейда* (Luís B. Almeida) та *Тереза Алмейда да Сілва* (Teresa Almeida da Silva) розробили метод з векторним кроком пошуку, який вони назвали *Super SAB* [1.6]. Цей метод є вдосконаленням алгоритму зворотного поширення помилки, де крок навчання адаптується автоматично, що дає змогу пришвидшити процес пошуку мінімуму функції помилки. Пошук відбувається за виразом

$$w_{ij}(t) = w_{ij}(t-1) - \eta_{ij}(t)\delta_j^{(n)} y_i^{(n-1)}; \tag{1.25}$$

$$\eta_{ij}(t) = \begin{cases} \eta_{ij}(t-1)u, \text{ якщо } \dfrac{\partial E(t)}{\partial w_{ij}} \dfrac{\partial E(t-1)}{\partial w_{ij}} \geq 0 \text{ і } \dfrac{\partial E(t-1)}{\partial w_{ij}} < 0, \\ \eta_{ij}(t-1)d, \text{ якщо } \dfrac{\partial E(t)}{\partial w_{ij}} \dfrac{\partial E(t-1)}{\partial w_{ij}} < 0, \end{cases} \tag{1.26}$$

де $u > 1$; $0 < d < 1$.

Зміна ваги відбувається відповідно до виразу [1.6]:

$$w_{ij}(t) = \begin{cases} w_{ij}(t-1) - \eta_{ij}(t)\delta_j^{(n)} y_i, \text{ якщо } \dfrac{\partial E(t)}{\partial w_{ij}} \dfrac{\partial E(t-1)}{\partial w_{ij}} \geq 0 \text{ і } \dfrac{\partial E(t-1)}{\partial w_{ij}} < 0; \\ w_{ij}(t-1) \text{ в іншому випадку.} \end{cases} \tag{1.27}$$

Автономний градієнтний метод з апроксимацією рельєфу квадратичною функцією. Нехай ми знаходимося в точці $w(t)$. Обчислюємо градієнт $\nabla E(w(t))$ і антиградієнт $-\nabla E(w(t))$ і робимо два пробних кроки:

$$w_1 = w(t) + \nabla E(w);$$
$$w_2 = w(t) - \nabla E(w).$$

Обчислюємо $E(w)$, $E(w_1)$ і $E(w_2)$. Далі, припускаючи, що $E(w)$ може бути апроксимована параболою, знаходимо за трьома точками w, w_1 і w_2 точку мінімуму. Основний недолік методу полягає в тому, що функція $E(w)$ може мати значно складніший вигляд, тому цю процедуру апроксимації доводиться повторювати багаторазово, що потребує великих обчислювальних витрат.

1.5. Прискорення збіжності алгоритмів навчання нейронних мереж. Алгоритм спряжених градієнтів

Як показано вище, алгоритм навчання мереж типу *Back Propagation* — це реалізація класичного методу якнайшвидшого спуску. Цей алгоритм навчання відносно простий у реалізації та застосуванні, що пояснює його широке використання в галузі нейромереж. Однак він має два істотні недоліки:

- має повільну збіжність;
- ефективний лише під час пошуку точок локального мінімуму.

Тому було розроблено інші, ефективніші методи навчання, які є альтернативою методу градієнтного спуску: метод спряжених градієнтів і метод, заснований на генетичній оптимізації.

Метод спряжених градієнтів. Метод спряжених градієнтів (МСГ) сприяє збільшенню швидкості збіжності порівняно з методом якнайшвидшого спуску. Однак, як і метод якнайшвидшого спуску, він є методом локальної оптимізації [1.6]. У нейронних мережах цільова функція, яку потрібно мінімізувати, — це середня похибка на всій множині навчальних зразків. Вона дорівнює

$$E(W) = \sum_{t=1}^{T} \sum_{k=1}^{M} (d_{tk} - y_{tk}(W))^2, \tag{1.28}$$

де $t = 1, 2, \ldots, T$ — множина навчальних зразків.

Для тришарової мережі з N вхідних вузлів, включно з пороговим вузлом, J прихованих вузлів та M вихідних вузлів вектор ваг W містить $NJ + MJ$ компонент. У формулі (1.28) M — кількість вихідних вузлів; $\{d_{t1}, d_{t2}, \ldots, d_{tM}\}$ — бажаний вихід для навчального зразка t, а $y_t(W) = \{y_{t1}(W), y_{t2}(W), \ldots, y_{tm}(W)\}$ — реакція (вихідний сигнал мережі) на зразок t.

Алгоритм, побудований на базі МСГ, як і найбільш загальний алгоритм спряжених напрямів, набув широкого застосування у сфері оптимізації у зв'язку з його здатністю розв'язувати широкий клас задач, для яких він забезпечує збіжність до оптимального результату після здійснення кінцевої кількості кроків. Це поліпшення є дуже важливим порівняно з методом якнайшвидшого спуску, який потрубує нескінченного числа ітерацій для пошуку мінімуму функції f.

Спряжені напрямки. Назва походить від використання спряжених векторів. У векторному просторі розмірності D множина векторів $\{P_1, P_2, \ldots, P_D\}$ утворює множину спряжених напрямків щодо матриці A, якщо [1.6]:

$$P_i^T A P_j = 1 \text{ для } i = j, \tag{1.29}$$

де A — додатно визначена матриця розміром $D \times D$.

Вектори, що задовольняють вираз (1.29), називають A-спряженими. Виникає запитання: як саме алгоритм, побудований на базі МСГ, дає можливість досягати збіжності за кінцеве число кроків та під час вирішення яких завдань.

Припустимо, що потрібно мінімізувати таку функцію:

$$F(W) = (b - AW)^T (b - AW), \tag{1.30}$$

де b і W — D-вимірні вектори, а матрицю $A^{D \times D}$ визначено вище. Отже, маємо квадратичну функцію. Припустимо, що ми шукаємо ітераційно оптимальний вектор W^*, який мінімізує $E(W)$, і починаємо пошук з початкової точки W_0. Вибираємо ненульовий вектор p_1, який є напрямом пошуку на наступній ітерації; при цьому не важливо, як було обрано W_0 і p_1.

Знаходимо W_1 як вектор:

$$W_1 = W_0 + \alpha p_1, \tag{1.31}$$

де скаляр α вибирається так, щоб мінімізувати $E(W_0 + \alpha p_1)$. Оптимальний напрямок, в якому потрібно рухатися на наступній ітерації, — це напрямок, в якому слід зробити лише один крок до точки оптимального рішення W^*, що утворює A-спряжену пару з вектором p_1.

Оптимальний напрямок — це $W^* - W_1$, тому умова, що $(W^* - W_1)$ є A-спряженим напрямком, еквівалентна твердженню, що має виконуватися умова: $(W^* - W_1)^T A p_1 = 0$.

Звичайно, у цій точці оптимальне рішення W^* невідоме, інакше не виникало б потреби застосовувати алгоритм. Однак ця умова важлива з такої причини. У D-вимірному просторі є рівно $D - 1$ незалежних векторів, які утворюють A-спряжену пару з вектором p_1. Отже, нам знадобиться лише кінцева кількість напрямків, щоб знайти оптимальне рішення.

Алгоритм спряжених напрямків систематично конструює множину A-спряжених векторів. Через максимум D кроків алгоритм знайде оптимальний напрямок і збіжність буде забезпечена.

Тут ми обминули важливе питання розв'язання задачі одновимірної мінімізації за скаляром α. Для завдань у формі (1.30) така мінімізація виконується безпосередньо і утворює частину класичного алгоритму СГ, хоча для більш загальних проблем це завдання аж ніяк не тривіальне.

У розглянутій задачі навчання Н-мережі рівняння (1.30) не існує в явній формі, і, зокрема, ми не маємо явного виразу для матриці A, хоча градієнт помилки ∇E може виконувати цю роль. Зауважимо, що у рівнянні (1.30) AW — множник, пропорційний градієнту функції $E(W)$. Це дає змогу використовувати вектор градієнта $E(W)$ для визначення напрямку руху.

Зазначимо, що алгоритм, побудований на базі МСГ, подібно до методу градієнтного спуску, забезпечує знаходження лише локально оптимальних рішень, а для проблем оптимізації з критерієм загального виду кінцева збіжність не гарантується. Окрім того, цей алгоритм забезпечує значне прискорення збіжності порівняно з методом якнайшвидшого спуску.

Опис алгоритму, побудованого на основі МСГ:

1. Вважаємо $k = 1$. Потрібно ініціалізувати ваговий вектор $W(k)$ та обчислити градієнт $G(k) = \text{grad}E(W(k))$. Нехай вектор початкового напрямку має вигляд: $p(k) = -\frac{G(k)}{||G(k)||}$; $(k + 1)$-*та ітерація.*

2. Необхідно знайти скаляр α, який мінімізує $E(W + \alpha p)$. Для виконання цього завдання використаємо метод Фібоначчі (Леонардо Фібоначчі (Leonardo Fibonacci)), або золотого перерізу. В результаті визначаємо $W(k + 1)$ за рівнянням:

$$W(k + 1)=W(k) + \alpha p(k). \tag{1.32}$$

3. Якщо $E(W(k + 1)) < \varepsilon_{\text{доп}}$, де $\varepsilon_{\text{доп}}$ — допустима точність знаходження мінімуму, то рішення отримано. Інакше — необхідно визначити новий напрямок: $G(k) = \text{grad}E(W(k + 1))$.

4. Якщо $\text{mod}_D(k) = 0$, де D — розмірність простору ваг W, то новий вектор напрямку матиме вигляд

$$p(k + 1) = -\frac{G(k + 1)}{||G(k + 1)||}. \tag{1.33}$$

Інакше приймаємо:

$$\beta(k) = \frac{G(k + 1)^T G(k + 1)}{G(k)^T G(k)} \tag{1.34}$$

і обчислюємо новий вектор напрямку:

$$p_{k+1} = -\frac{-G(k + 1) + \beta_p(k)}{||-G(k + 1) + \beta_p(k)||}.$$

5. Необхідно замінити $p(k)$ на $p(k + 1)$ і $G(k)$ на $G(k + 1)$. Здійснюємо перехід до кроку 1 наступної ітерації. Вибір $\beta(k)$ згідно з (1.34) забезпечує умову, що $p(k + 1)$ буде спряженим до $p(k)$.

1.6. Генетичний алгоритм навчання нейронної мережі

Цей алгоритм є алгоритмом глобальної оптимізації. У ньому використовують такі механізми [1.2, 1.6]:

1. Схрещування батьківських пар (cross-over), генерація нащадків.

2. Мутація (дія випадкових впливів).

3. Природний добір найкращих (селекція).

Цільовою функцією навчання є мінімізація середньоквадратичної похибки:

$$E(w) = \frac{1}{M} \Sigma_{k=1}^{M} (d_k - y_k(w))^2;$$

$$W = [W_I, W_O];$$

$$W_I = ||w_{ij}^I||;$$

$$W_O = ||w_{ij}^O||.$$

На початку роботи алгоритму задають початкову популяцію.

Будь-яку популяцію з N особин представляють відповідними вагами: $[W_I(0), \ldots, W_i(0), \ldots, W_N(0)]$.

Обчислюємо індекс придатності (*Fitness Index*) і оцінюємо якість прогнозування:

$$FI(W_i) = C - E(W_i) \to \max, \tag{1.35}$$

де C — константа.

Схрещування батьківських пар. Для вибору батьків використовують імовірнісний механізм. Позначимо P_i — ймовірність вибору i-го батька [1.6]:

$$P_i = \frac{FI(W_i(O))}{\sum_{i=1}^{N} FI(W_i(O))}.$$

Потім здійснюють схрещування вибраних пар. Можна застосовувати різні механізми схрещування, наприклад такий.

Для першого нащадка беруть непарні компоненти вектора першого батька, а парні компоненти — вектора другого батька. Для другого нащадка, навпаки, парні компоненти беруть із вектора першого з батьків, а непарні компоненти —з вектора другого з батьків. Це можна записати так:

$$W_i(0) \oplus W_k(0) = W_i(1) \cup W_k(1);$$

$$W_i = [w_{ij}]_{j=\underline{1,R}};$$

$$w_{ij}(1) = \begin{cases} w_{kj}(0), \text{ якщо } j = 2m - 1, \text{ непарне;} \\ w_{ij}(0), \text{ якщо } j = 2m, \text{ парне;} \end{cases}$$

$$w_{kj}(1) = \begin{cases} w_{ij}(0), \text{ якщо } j = 2m - 1; \\ w_{kj}(0), \text{ якщо } j = 2m; \end{cases}$$

$$m = \overline{1, \frac{R}{2}}.$$

Всього береться $N/2$ батьківських пар і генерується N нащадків.

Дія мутацій. Мутації виникають унаслідок впливу на ваги випадкового шуму. Мутації відіграють важливу роль в ГА; вони забезпечують необхідну різноманітність популяції і виключають можливість її виродження [1.11]: $w'_{ij}(n) = w_{ij}(n) + \xi(n)$, де $\xi(n) = ae^{-\alpha n}$, $a = \text{const} \in [-1; 1]$, $\alpha > 0$, — показник згасання мутацій, n — номер ітерації.

Селекція. Можна використати різні механізми селекції:

1. Повна заміна старої популяції на нову.

2. Вибір N найкращих з усіх особин $N_{\text{батьк}} + N_{\text{нащад}} = 2N$ за критерієм максимуму FI.

Вказані ітерації повторюються до виконання однієї з таких умов зупинки:

1) за кількістю ітерацій $n \geq n_{\text{зад}}$;

2) після досягнення заданої величини фітнес-індексу $\max FI(W_i(n)) \geq FI_{\text{зад}}$;

3) якщо протягом m ітерацій FI для найкращої особини не змінюється.

Основна перевага методу полягає в тому що він забезпечує вирішення завдання *глобальної оптимізації*, тобто за числа ітерацій n дає змогу знайти глобально оптимальне рішення.

Основні недоліки методу:

1. Низка параметрів визначається експериментально, наприклад, N — розмір популяції; a — показник згасання мутацій.

2. Реалізація генетичного алгоритму пов'язана з великим обсягом обчислень.

1.7. Нейронні мережі з радіально-базисними функціями (РБФ-мережі)

Багатошарові нейронні мережі типу *Back Propagation* дають можливість здійснювати глобальну апроксимацію функцій кількох змінних, перетворюючи множину вхідних змінних на множину вихідних змінних $y \in R^M$ [1.2, 1.11]. Завдяки сигмоїдальному характеру активаційної функції реалізується глобальна апроксимація: нейрон, який активувався (після перевищення сумарним сигналом певного порогового значення), залишається активним за будь-якого значення, що перевищує цей поріг. Отже, значення функції у довільній точці простору визначається сумарною дією багатьох нейронів, що пояснює термін «глобальна апроксимація».

Інший спосіб відображення вхідної множини у вихідну передбачає адаптацію декількох окремих функцій апроксимації до очікуваних значень, причому така адаптація виконується лише в обмеженій ділянці багатовимірного простору. Завдяки цьому підходу відображення всієї множини даних є сумою локальних перетворень. У цьому разі приховані нейрони слугують локальними базисними функціями. Такі функції активуються (набувають ненульових значень) тільки в обмежених ділянках простору даних, тому цей підхід називають «локальною апроксимацією».

Суть радіальних нейронних мереж. Нейронні мережі, які ґрунтуються на радіально-базисних функціях (РБФ-мережі), є особливим сімейством, у якому приховані нейрони реалізують функції, що радіально змінюються навколо обраного центру і набувають ненульових значень лише побли-

зу цього центру. Такі функції, позначені як $\varphi(x) = [\varphi_1(x), \varphi_2(x), \ldots, \varphi_K(x)]^T$, називають радіальними базисними функціями. У цих мережах роль прихованого нейрона полягає у відображенні радіального простору навколо окремої точки або групи точок, які утворюють кластер. Вихідний нейрон, об'єднуючи сигнали від усіх прихованих нейронів, забезпечує відображення всього багатовимірного простору.

Мережі радіального типу є природним доповненням до сигмоїдальних мереж. У багатовимірному просторі сигмоїдальний нейрон становить собою гіперплощину, яка поділяє простір на дві категорії (класи), де виконується одна з двох умов:

$$\Sigma_{j=1}^{N} w_{ij} x_j > 0, \text{ якщо } x \in A_1, \text{ або } \Sigma_{j=1}^{N} w_{ij} x_j < 0, \text{ якщо } x \in A_2.$$

Зі свого боку радіальний нейрон є гіперсферою, яка здійснює сферичний поділ простору навколо центральної точки. Саме тому він є природним доповненням сигмоїдального нейрона, оскільки у разі кругової симетрії даних дає змогу помітно зменшити кількість нейронів, необхідних для поділу різних класів.

Найпростіша нейронна мережа радіального типу функціонує за принципом багатовимірної інтерполяції, а саме відбувається відображення p різних вхідних векторів ($i = 1, 2, \ldots, p$) з вхідного N-вимірного простору в множину з p раціональних чисел ($i = 1, 2, \ldots, p$). Для реалізації цього процесу необхідно використовувати p прихованих нейронів радіального типу і задати таку функцію відображення $F(x)$, для якої виконується умова інтерполяції [1.2, 1.5]:

$$F(x_j) = d_i. \tag{1.36}$$

Використання p прихованих нейронів, що з'єднуються зв'язками з вагами з вихідними лінійними нейронами, означає формування вихідних сигналів мережі підсумовуванням зважених значень відповідних базисних функцій.

Розглянемо радіальну мережу з одним виходом і p навчальними парами $((X_1, d_1), \ldots, (X_p, d_p))$. Припустимо, що координати кожного з центрів вузлів мережі визначаються одним з векторів x_i, $i = 1, p$. У цьому разі взаємозв'язок між вхідними і вихідними сигналами мережі може бути визначений системою рівнянь, лінійних щодо ваг w, яка в матричній формі має вигляд $w^T\varphi(x) = d$.

Математичну основу функціонування радіальних мереж становить теорема американського математика Томаса М. Ковера (Thomas M. Cover, 1965 р.) про розпізнаваність образів, яка стверджує, що в просторі високої розмірності довільну сукупність точок, які належать до двох класів, можна лінійно розділити з більшою ймовірністю, ніж за їх проєкції в простір з меншою розмірністю. Ця теорема відіграє важливу роль у розумінні пове-

дінки нейронних мереж, зокрема одношарових перцептронів, і виявилася фундаментальною для завдань класифікації.

Якщо вектор радіальних функцій $\varphi(x) = [\varphi_1(x), \varphi_2(x), \ldots, \varphi_K(x)]^T$ у N-вимірному вхідному просторі позначити $\varphi(x)$, то цей простір є нелінійно розділюваним на два просторові класи X^+ і X^- у разі, якщо існує такий вектор ваг w, що виконуються умови [1.6, 1.11]:

$$w^T\varphi(x) > 0, x \in X^+; \tag{1.37}$$

$$w^T\varphi(x) < 0, x \in X^-. \tag{1.38}$$

Межа між цими класами визначається рівнянням $w^T\varphi(x) = 0$. Доведено, що кожна множина образів, випадково розміщених у багатовимірному просторі, є φ-розділюваною з імовірністю 1 за умови відповідно великої розмірності K цього простору. Насправді це означає, що застосування досить великої кількості прихованих нейронів, які реалізують радіальні функції $\varphi_i(x)$, гарантує виконання завдання класифікації під час побудови лише двошарової мережі: прихований шар має реалізувати вектор $\varphi(x)$, а вихідний шар може складатися з єдиного лінійного нейрона, що виконує підсумовування вихідних сигналів від прихованих нейронів з ваговими коефіцієнтами, заданими вектором w.

Приймемо, що координати кожного з центрів вузлів мережі визначаються одним з векторів x_i, тобто $c_i = x_i$. У цьому разі взаємозв'язок між вхідними та вихідними сигналами мережі може бути визначений системою рівнянь, лінійних щодо ваг w_i, яка в матричній формі має вигляд:

$$\begin{bmatrix} \varphi_{11} & \varphi_{12} & \ldots & \varphi_{1p} \\ \varphi_{21} & \varphi_{22} & \ldots & \varphi_{2p} \\ \ldots & \ldots & \ldots & \ldots \\ \varphi_{p1} & \varphi_{p2} & \ldots & \varphi_{pp} \end{bmatrix} \begin{bmatrix} w_1 \\ w_{2p} \\ \ldots \\ w_p \end{bmatrix} = \begin{bmatrix} d_1 \\ d_{2p} \\ \ldots \\ d_p \end{bmatrix}, \tag{1.39}$$

де $\varphi_{ij} = (||x_j - x_i||)$ визначає радіальну функцію з центром у точці x_i з вхідним вектором x_j.

Якщо позначити матрицю з елементів φ_{ij} як Φ і ввести позначення векторів $w = [w_1, w_2, \ldots, w_p]^T$, $d = [d_1, d_2, \ldots, d_p]^T$, то можна записати систему рівнянь у редукованій матричній формі:

$$\Phi^T w = d. \tag{1.40}$$

Доведено, що для низки радіальних функцій у разі $x_1 \neq x_2 \neq \ldots \neq x_p$ квадратна інтерполяційна матриця Φ є невласною і при цьому додатно визначеною. Тому існує рішення рівняння (1.40), а саме:

$$w = \Phi^{-1}d, \tag{1.41}$$

що дає змогу визначити вектор ваги вихідного нейрона мережі.

Теоретичне розв'язання проблеми, подане виразом (1.41), не може вважатися абсолютно істинним через суттєве обмеження загальних властивостей мережі, що випливають із попередньо зроблених припущень. За дуже великої кількості навчальних вибірок і такої самої кількості радіальних функцій проблема з математичного погляду стає нескінченною (неналежно структурованою), оскільки кількість рівнянь починає перевищувати число ступенів свободи фізичного процесу, що моделюється рівнянням (1.39). Це означає, що результатом такої надмірної кількості вагових коефіцієнтів стане адаптація моделі до різного роду шумів або нерегулярностей, характерних для навчальних вибірок. Як наслідок, гіперплощина, що інтерпретує ці дані, не буде гладкою, а узагальнювальні можливості залишаться дуже слабкими. Щоб їх посилити, слід зменшити кількість радіальних функцій і отримати додаткову інформацію з надлишкового обсягу даних, що сприятиме регуляризації задачі та поліпшенню її зумовленості.

Радіальна нейронна мережа. Використання в розкладанні P базисних функцій, де P — кількість навчальних зразків, неприпустиме також із практичного погляду, оскільки зазвичай кількість цих зразків дуже велика і в результаті обчислювальна складність алгоритму, що навчається, стає надмірною. Розв'язання системи рівнянь (1.40) розмірністю $(P \times P)$ за великих значень P стає складним, оскільки дуже великі матриці (за винятком ортогональних), як правило, мають порядковий характер, а коефіцієнт порядку може сягати навіть 10020. Тому так само, як і для багатошарових мереж, потрібно зменшити кількість ваг, що призводить до зменшення кількості базисних функцій у рази. Тому шукають субоптимальне рішення у просторі меншої розмірності, яке з достатньою точністю апроксимує розв'язання. Якщо обмежитися K базових функцій, то рішення апроксимації можна подати у вигляді

$$F(x) = \sum_{i=1}^{K} W_i \varphi(||x - c_i||), \tag{1.42}$$

де $K < P$, а c_i $(i = 1, 2, \ldots, K)$ — множина центрів, які потрібно визначити. В особливому випадку, якщо прийняти $K = P$, можна отримати точне рішення $c_i = x_i$.

Завдання апроксимації полягає у підборі відповідної кількості радіальних функцій $\varphi(||x - c_i||)$ та їх параметрів, а також у такому доборі ваг W_i $(i = 1, 2, \ldots, K)$, щоб рішення рівняння (1.42) було найближчим до точного. Тому проблему підбору параметрів радіальних функцій та значень ваг w_i мережі можна звести до мінімізації цільової функції, яка за використання евклідової метрики записується у формі [1.11, 1.14]:

$$E = \sum_{i=1}^{p} [\sum_{i=1}^{K} W_i \varphi(||x - c_i||) - d_i]^2. \tag{1.43}$$

У цьому рівнянні K — кількість радіальних нейронів, p — кількість зразків вибірки (x_i, d_i), де x_i — вхідний вектор, d_i — відповідна йому очікувана величина. Позначимо $d = [d_1, d_2, \ldots, d_p]^T$ — вектор очікуваних значень,

$w = [w_1, w_2, \ldots, w_p]^T$ — вектор ваг мережі, G — радіальна матриця, або так звана матриця англійського математика Джорджа Гріна (1828 р.) [1.11]:

$$G = \begin{bmatrix} \varphi(||x_1 - c_1||) & \varphi(||x_1 - c_2||) & \ldots & \varphi(||x_1 - c_K||) \\ \varphi(||x_2 - c_1||) & \varphi(||x_2 - c_2||) & \ldots & \varphi(||x_2 - c_K||) \\ \ldots & \ldots & \ldots & \ldots \\ \varphi(||x_p - c_1||) & \varphi(||x_p - c_2||) & \ldots & \varphi(||x_p - c_K||) \end{bmatrix}.$$

Матриця Гріна (або функція Гріна) у контексті штучного інтелекту, особливо в машинному навчанні та обчислювальній фізиці, є інструментом для розв'язання лінійних диференціальних рівнянь, які описують різноманітні процеси, зокрема поширення сигналів, дифузію, теплопровідність, хвильові процеси тощо. Завдяки цим властивостям матриця Гріна дає можливість знижувати обчислювальні витрати і збільшувати точність у складних завданнях моделювання й прогнозування, що є важливим аспектом сучасних методів штучного інтелекту.

У разі обмеження K базисними функціями матриця G стає прямокутною з кількістю рядків, як правило, значно більшою, ніж число стовпців ($p >> K$).

Якщо припустити, що параметри радіальних функцій відомі, то оптимізаційна задача (1.44) зводиться до розв'язання системи лінійних рівнянь щодо ваг w [1.14]:

$$G(w) = d. \tag{1.44}$$

З урахуванням прямокутності матриці G можна визначити вектор ваг w, використовуючи операцію псевдоінверсії матриці G, тобто:

$$w = G^+ d, \tag{1.45}$$

де $G^+ = (G^T)^{-1} G^T$ позначає псевдоінверсію прямокутної матриці G. Псевдоінверсія прямокутної матриці — це узагальнення операції обернення, яка застосовується до матриць, що не є квадратними або не мають оберненої матриці. Для прямокутної матриці G псевдоінверсія, також відома як псевдообернена матриця Мура — Пенроуза (Еліакім Гастінгс Мур (Eliakim Hastings Moore), Роджер Пенроуз (Roger Penrose)) (позначається як G^+), використовується для знаходження найкращого наближеного розв'язання системи лінійних рівнянь. Її часто застосовують у задачах регресії, оброблення сигналів і машинного навчання.

В обчислювальній практиці псевдоінверсію розраховують із застосуванням декомпозиції *SVD (Singular Value Decomposition)* або сингулярного розкладання — методу факторизації матриці, який розкладає довільну (необов'язково квадратну) матрицю G на добуток трьох матриць: унітарної матриці, яка містить ліві сингулярні вектори матриці G; діагональної матри-

ці, що містить сингулярні значення матриці *G*; транспонованої (або спряженої) унітарної матриці, яка містить праві сингулярні вектори матриці *G*.

У розглянутому вище рішенні використовувалося подання базових функцій матрицею Гріна, що залежить від евклідової норми вектора $||x - t_i||$. Якщо взяти до уваги, що багатовимірна функція може мати різний масштаб по кожній осі, то доцільно уточнити норму масштабування, увівши визначення евклідової метрики вагових коефіцієнтів у вигляді матриці *Q*:

$$||x_1||_Q^2 = (Qx)^T(Qx) = x^TQ^TQX. \tag{1.46}$$

Матриця масшабування за *N*-вимірного вектора *x* має вигляд

$$Q = \begin{bmatrix} Q_{11} & Q_{12} & \dots & Q_{1N} \\ Q_{21} & Q_{22} & \dots & Q_{2N} \\ \dots & \dots & \dots & \dots \\ Q_{N1} & Q_{N2} & \dots & Q_{NN} \end{bmatrix}.$$

Позначивши добуток матриць Q^TQ матрицею кореляції *C*, у загальному випадку матимемо:

$$||x||_Q^2 = \Sigma_{i=1}^N \Sigma_{j=1}^N C_{ij} x_i x_j. \tag{1.47}$$

Якщо матриця масштабування *Q* має діагональний вигляд, то $||x||_Q^2 = \Sigma_{i=1}^N C_{ii}^N x_i^2$. Це означає, що норма масштабування вектора *x* розраховується згідно зі стандартною формулою Евкліда, з використанням індивідуальної шкали для кожної змінної x_i. При $Q = 1$ зважена евклідова метрика зводиться до класичної (немасштабованої) метрики: $||x||_Q^2 = ||x||^2$.

Найчастіше як радіальну функцію застосовують функцію Гаусса. У разі розміщення її центра в точці c_i вона може бути визначена у скороченій формі:

$$\varphi(x) = \varphi(||x - c_{ij}||) = \exp(-\frac{||x - c_i||^2}{2\sigma_i^2}). \tag{1.48}$$

У цьому виразі σ_i — параметр, від значення якого залежить ширина функції. За гауссової форми радіальної функції з центром у точці c_i і зваженої матриці масштабування Q_i, пов'язаної з *i*-ю базовою функцією, отримуємо узагальнену форму функції Гаусса, що використовується як радіальна базисна функція (*RBF — Radial Basis Function*) і є спеціальною функцією, що моделює розподіл значень навколо центральної точки з певним радіусом [1.14]:

$$\varphi(x) = \varphi(||x - c_i||_{Q_i}) = \exp[-(x - c_i)^T Q_i^T Q_i(x - c_i)] =$$
$$= \exp[-\frac{1}{2}(x - c_i)^T C_i(x - c_i)], \tag{1.49}$$

де матриця відіграє роль скалярного коефіцієнта стандартної багатовимірної функції Гаусса, заданої виразом (1.49).

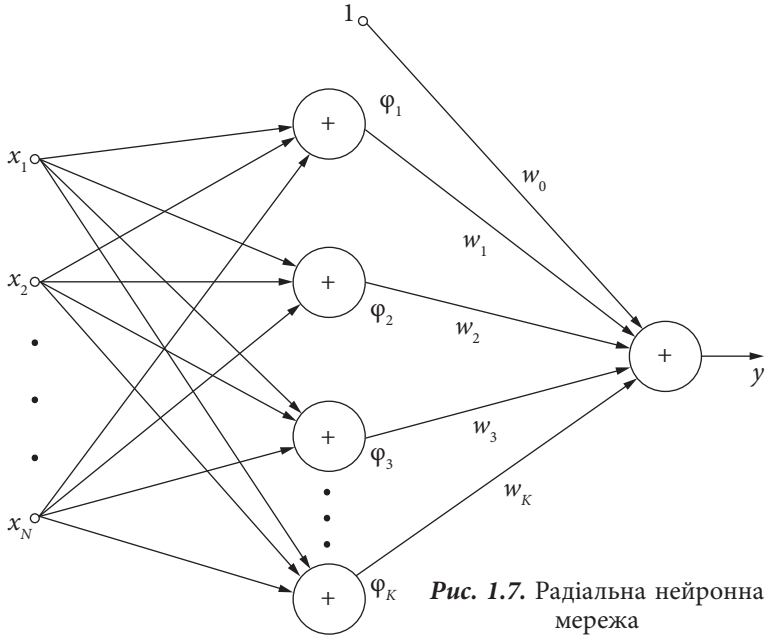


Рис. 1.7. Радіальна нейронна мережа

Отримане рішення, що зображує функцію апроксимації в багатовимірному просторі у вигляді зваженої суми локальних базисних радіальних функцій (вираз (1.42)), може бути інтерпретоване радіальною нейронною мережею, зображеною на рис. 1.7 (для спрощення ця мережа має лише один вихід), у якій φ_i визначається залежністю (1.48) або (1.49).

Ця мережа має двошарову структуру, де тільки прихований шар виконує нелінійне перетворення, реалізоване нейронами з радіально-базисними функціями. Вихідний нейрон зазвичай є лінійним, і його роль полягає у зваженому підсумовуванні сигналів, що надходять від нейронів прихованого шару. Вага , як і у випадку з сигмоїдальними функціями, становить собою поріг для постійного зсуву функції.

Архітектура радіальної мережі подібна до багатошарової архітектури сигмоїдальної мережі з одним прихованим шаром. Роль прихованих нейронів виконують радіально-базисні функції, які відрізняються своєю формою від сигмоїдальних функцій. Незважаючи на подібність, ці мережі принципово різняться. Радіальна мережа має фіксовану структуру з одним прихованим шаром і лінійними вихідними нейронами, тоді як сигмоїдальна мережа може містити кілька шарів, а вихідні нейрони можуть бути як лінійними, так і нелінійними. Радіальні функції можуть мати різну структуру [1.11, 1.14]. Нелінійна радіальна функція кожного прихованого нейрона визначається параметрами c_i і s_i, тоді як у сигмоїдальній мережі зазвичай використовуються стандартні активаційні функції з однаковим для всіх

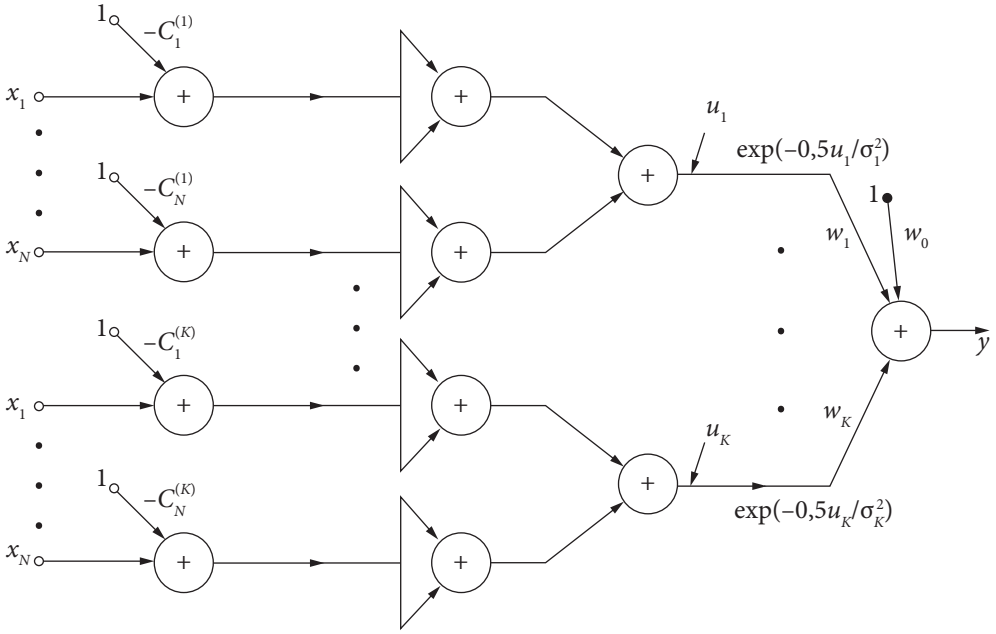


Рис. 1.8. Схема РБФ-мережі (*RBF*) з радіальною функцією виду (1.48) за класичного розуміння евклідової метрики

нейронів параметром σ. Аргументом радіальної функції є евклідова відстань від зразка x до центра c_i, тоді як у сигмоїдальній мережі це скалярний добуток векторів $w^T x$.

Значні відмінності між цими мережами можна побачити у разі детального порівняння їх структур. Сигмоїдальна мережа має багатошарову структуру, де конфігурація нейронів повторюється в кожному шарі. Кожен нейрон у сигмоїдальній мережі виконує підсумовування сигналів із наступною активацією. Структура радіальної мережі відрізняється. На рис. 1.8 зображено детальну схему РБФ-мережі (*RBF*) з радіальною функцією типу (1.48) за класичним розумінням евклідової метрики. З рисунка видно, що перший шар складається з нелінійних радіальних функцій, параметри яких (центри c_i і коефіцієнти s_i) уточнюються під час навчання. Перший шар не містить лінійних ваг у тому сенсі, що характерно для сигмоїдальної мережі.

Ще складнішою виявляється детальна структура мережі, що реалізує масштабовану радіальну функцію Q у вигляді, визначеному виразом (1.48). Така мережа, наведена на рис. 1.9, називається гіперрадіальною базисною функцією, або *HRBF (Hyper Radial Basis Function)* [1.11]. Вона являє собою узагальнення радіально-базисної функції (*RBF*), яке дає змогу моделювати складніші просторові залежності, зокрема в багатовимірних задачах інтерполяції й апроксимації даних. *HRBF* використовують для розв'язання

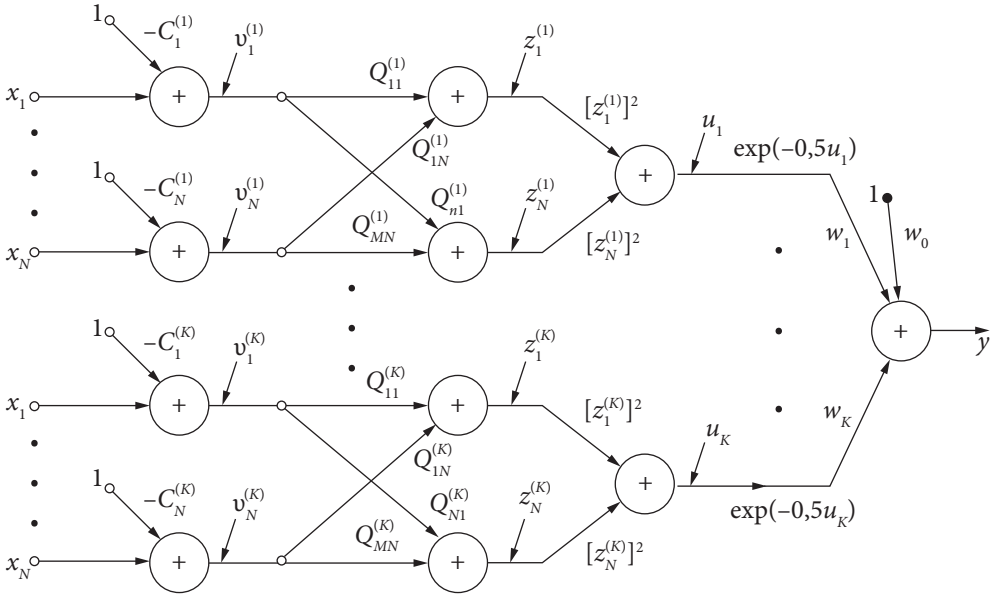


Рис. 1.9. Структура мережі, що реалізує масштабовану радіальну функцію Q у вигляді метрики, визначеної виразом (1.47)

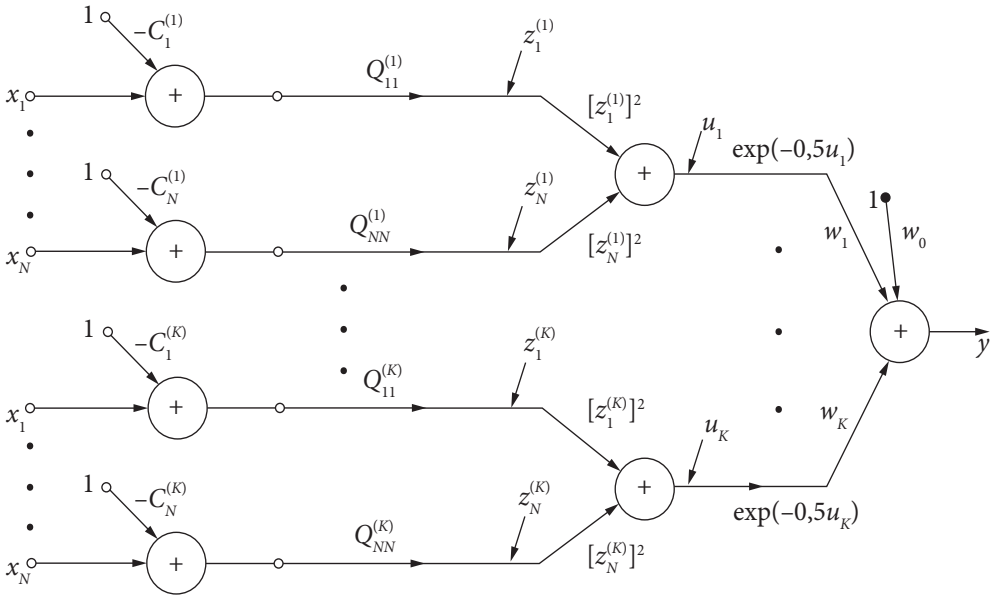


Рис. 1.10. Схема структури спрощеної радіальної мережі *HRBF* з діагональними матрицями

задач, де класична *RBF* виявляється недостатньо точною або ефективною, особливо в задачах, що потребують адаптації до даних з більшою кількістю змінних чи специфічною поведінкою. Тобто *HRB*-функції поєднують переваги радіально-базисних функцій з можливістю працювати в складних, високорозмірних просторах, забезпечуючи високу точність і адаптивність.

Радіальний нейрон у цій мережі має особливо складну структуру, що містить суматори сигналів, аналогічні застосовуваним у сигмоїдальній мережі, та показові функції активації з параметрами, що підлягають уточненню у процесі навчання. Ваги k-го радіального нейрона прихованого шару — це елементи матриці $Q(k)$, які відіграють роль системи масштабування. Вони вводять додаткові ступені свободи мережі, що дає змогу краще наблизити вихідний сигнал мережі $y = f(x)$ до очікуваної функції $d(x)$.

У багатьох практичних додатках матриця масштабування $Q(k)$ має діагональну форму, в якій лише діагональні елементи набувають ненульових значень. У такій системі немає кругового перемішування сигналів, що відповідають різним компонентам вектора x, а елемент відіграє роль індивідуального коефіцієнта масштабування для i-го компонента вектора x k-го нейрона.

На рис. 1.10 наведено детальну схему структури спрощеної радіальної мережі HRBF з діагональними матрицями масштабування $Q(k)$.

Слід зазначити, що у мережах *HRBF* роль коефіцієнтів виконують елементи матриці Q, які уточнюються у процесі навчання.

Методи навчання радіальних нейронних мереж

Описані раніше методи підбору ваг w_i вихідного шару радіальної мережі *RBF* були засновані на припущенні, що параметри самих базисних функцій відомі, у зв'язку з чим матриці Гріна вважаються визначеними, і отже, завдання зводиться до розв'язування надлишкової системи лінійних рівнянь виду (1.42). Практично такий підхід можливий тільки в абсолютно нереальному випадку. $K = p$, за якого центри $c_i = x_i$ відомі заздалегідь, а значення параметра σ_i можна легко підібрати експериментальним шляхом за умови дотримання певного компромісу між монотонністю та точністю відображення. Насправді завжди $K << p$, тому процес навчання мережі *RBF* з урахуванням обраного типу радіальної базової функції зводиться до:

- добору центрів c_i та параметрів σ_i форми базисних функцій;
- добору ваг нейронів вихідного шару.

Тоді проблема уточнення ваг нейронів вихідного шару значно спрощується. Відповідно до формули (1.45) вектор ваг w можна визначити за один крок, здійснивши псевдоінверсію матриці G : $w = G^+d$. Матриця G, що має p рядків і K стовпців, становить собою реакції нейронів прихованого шару на чергові збурення x_i ($i = 1, 2, ..., p$). Псевдоінверсію матриці G розрахову-

ють з використанням розкладання за власними значеннями, відповідно до яких маємо [1.11]:

$$G = USV^T. \tag{1.50}$$

Матриці U і V є ортогональними і мають розмірності $(p \times p)$ і $(K \times K)$ відповідно, тоді як S — псевдодіагональна матриця з розмірністю $(p \times K)$. При цьому $K < p$, а діагональні елементи $s_1 \geq s_2 \geq ... \geq s_K \geq 0$. Припустимо, що тільки r перших елементів s_i мають значну величину, а іншими можна знехтувати. Тоді кількість стовпців ортогональних матриць U і V може бути зменшено до r. Отримані в такий спосіб редуковані матриці U_r і V_r мають вигляд [1.14]:

$$U_r = [u_1\ u_2 \dots\ u_r],$$
$$V_r = [v_1\ v_2 \dots\ v_r],$$

а матриця $S_r = \text{diag}[s_1, s_2, \dots, s_r]$ стає повністю діагональною (квадратною).

Цю матрицю описує залежність (1.50) у такій формі:

$$G \cong U_r S_r V_r^T. \tag{1.51}$$

Псевдообернена до G матриця в цьому випадку визначається виразом

$$G^+ = V_r S_r^{-1} U_r^T, \tag{1.52}$$

в якому $S_r^{-1} = \left[\frac{1}{s_1}, \frac{1}{s_2}, \dots, \frac{1}{s_r}\right]$, а вектор ваг мережі, яку навчають, задається формулою

$$w = V_r S^{-1} U_r^T d. \tag{1.53}$$

Перевагою формули (1.53) є її простота. Вихідні ваги мережі підбираються за один крок простим перемноженням відповідних матриць, причому деякі з них (U_r, V_r) є ортогональними і за своєю природою добре впорядкованими (коефіцієнт порядку дорівнює 1).

Беручи до уваги (1.53), що визначає значення ваг вихідного шару, головною проблемою навчання радіальних мереж залишається підбір параметрів нелінійних радіальних функцій, особливо центрів c_i.

Одним із найпростіших, хоч і не найефективнішим, способом визначення параметрів базисних функцій вважається випадковий вибір. У цьому рішенні центри c_i базисних функцій вибираються випадково на основі рівномірного розподілу. Такий підхід припустимий стосовно класичних радіальних мереж за умови, що рівномірний розподіл навчальних даних добре відповідає специфіці завдання.

Під час вибору гауссової форми радіальної функції визначається значення стандартного відхилення s_i, що залежить від розкиду обраних випадковим чином центрів c_i, а саме:

$$\varphi(||x - c_{ij}||^2) = \exp\left(-\frac{||x - c_i||^2}{d^2 / K}\right) \tag{1.54}$$

для $i = 1, 2, \ldots, K$, де d —максимальна відстань між центрами c_i. З виразу (1.54) випливає, що стандартне відхилення гауссівської функції, яке характеризує ширину кривої, встановлюється таким, що дорівнює $\sigma = d / \sqrt{2K}$ як у разі випадкового вибору, так і завжди для всіх базових функцій.

Стандартне відхилення функції σ пропорційне максимальному розкиду центрів і зменшується зі збільшенням їх кількості.

Серед багатьох спеціалізованих методів і алгоритмів підбору центрів розглянемо кілька найважливіших: метод поділу на кластери, що самоорганізовується, гібридний алгоритм і метод навчання з учителем.

Застосування процесу самоорганізації для уточнення параметрів радіальних функцій

Досить хороші результати уточнення параметрів радіальних функцій можна отримати за допомогою алгоритму самоорганізації. Процес самоорганізації навчальних даних автоматично розподіляє простір на так звані області Вороного, що визначають групи даних [1.11]. Приклад такого поділу двовимірного простору даних на сфери впливу окремих радіальних функцій показано на рис. 1.11.

Дані, згруповані в межах кластера, представлені центральною точкою, що визначає середнє значення всіх його елементів. Центр кластера надалі ототожнюється з центром відповідної радіальної функції. Отже, кількість радіальних функцій відповідає кількості кластерів і може коригуватися алгоритмом самоорганізації.

Поділ даних на кластери можна здійснити за допомогою однієї з версій алгоритму Лінде — Бузо — Грея (Linde — Buzo — Gray algorithm (алгоритм був розроблений у 1980 р. трьома дослідниками — Єсом Лінде (Yves Linde, Франція) та А. Бузо (A. Buzo) і Робертом Греєм (Robert M. Gray, обидва зі США)), скорочено *LBG algorithm*) [1.14] (*K-means*). Це алгоритм кластеризації, який застосовується для векторного квантування, особливо в оброблені сигналів. Він також відомий як алгоритм *K*-середніх (*K*-means), оскільки використовує подібну методику кластеризації за допомогою розподілу даних на кластери на основі їх близькості до центрів кластерів.

У прямій (онлайн) версії цього алгоритму уточнення центрів здійснюється після надходження кожного нового вектора x із множини навчальних даних. У накопичувальній версії (офлайн) центри уточнюються одночасно після обробки всіх елементів

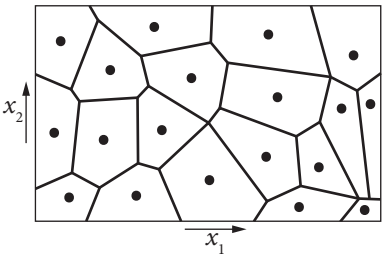


Рис. 1.11. Поділ двовимірного простору даних на сфери впливу окремих радіальних функцій

множини. В обох випадках виконується зазвичай випадковий попередній вибір центрів із застосуванням рівномірного розподілу.

Якщо навчальні дані становлять собою неперервну функцію, початкові значення центрів зазвичай розміщують у точках, що відповідають максимальним і мінімальним значенням функції. Дані про ці центри та їх найближче оточення згодом видаляються з навчальної множини, а центри, що залишились, рівномірно розподіляються у сфері, утвореній рештою елементів множини.

У прямій версії після надходження k-го вектора x_k, що належить навчальній множині, обирається центр, найближчий до x_k згідно з обраною метрикою. Цей центр уточнюється відповідно до алгоритму Кохонена — *WTA (Winner Takes All)* [1.18].

$$c_i(k+1) = c_i(k) + \eta[x_k - c_i(k)], \tag{1.55}$$

де η — коефіцієнт навчання, що має мале значення (зазвичай $\eta << 1$), причому зменшується у часі.

Інші центри не змінюються. Усі навчальні вектори x надаються кілька разів, зазвичай у випадковій послідовності до моменту стабілізації значень центрів.

Також застосовується різновид алгоритму, згідно з яким значення центру-переможця уточнюється відповідно до формули (1.55), а один або кілька найближчих до нього центрів відсуваються у протилежному напрямку [1.11], і цей процес реалізується відповідно до виразу

$$(k+1) = c_i(k) - \eta[x_k - c_i(k)]. \tag{1.56}$$

Така модифікація алгоритму дає можливість віддалити центри, розміщені близько один до одного, що забезпечує найкраще обстеження всього простору даних ($\eta_1 < \eta$).

У накопичувальній версії надаються всі навчальні вектори, і кожен із них зіставляється з якимось центром. Множина векторів, приписаних тому самому центру, утворює кластер, новий центр якого визначається як середнє відповідних векторів:

$$c_i(k+1) = \frac{1}{N} \sum_{j=1}^{N_i} x_j(k). \tag{1.57}$$

У цьому виразі N — кількість векторів $x(k)$, приписаних у k-му циклі до i-го центру. Усі центри уточнюються паралельно. Процес подання множини векторів x та уточнення значень центрів повторюється багаторазово до стабілізації їхніх значень. На практиці найчастіше застосовують прямий алгоритм, що демонструє дещо кращу збіжність. Однак жоден алгоритм не гарантує абсолютну збіжність до глобально оптимального рішення, а забезпечує лише локальну оптимізацію, яка залежить від початкових умов

і параметрів процесу навчання. У разі невдалого вибору початкових умов деякі центри можуть опинитися в зонах з малою кількістю навчальних даних або їх відсутністю, що призводить до загальмування або зупинення процесу модифікації центрів.

Одним із способів вирішення цієї проблеми є одночасне коригування розміщення великої кількості центрів з фіксацією значення η для кожного з них. Центр, найближчий до поточного вектора x, модифікується найсильніше, тоді як інші — обернено пропорційно їх відстані до цього поточного вектора.

Інший підхід полягає у використанні зваженої міри відстані від кожного конкретного центру до поданого вектора x. Зважена норма робить «фаворитами» ті центри, які рідше ставали «переможцями». Обидва підходи не гарантують 100 %-ї оптимальності рішення, оскільки є фактично процедурами збурення в процесі локальної оптимізації [1.11].

Проблема також полягає у виборі коефіцієнта навчання η. За використання постійного значення η воно має бути дуже малим для гарантованої збіжності алгоритму, що суттєво збільшує час навчання. Адаптивні методи підбору η, зокрема зменшення зі зростанням номера ітерації k, дають змогу зробити його значення залежним від часу. Найвідомішим представником цієї групи є алгоритм Даркена — Муді (Кріс Даркен (Chris Darken), Джон Муді (John Moody)) (Darken — Moody Algorithm) [1.11], що використовує самоналаштування для вдосконалення процесу навчання нейронних мереж, зокрема мереж з радіально-базисними функціями (*RBF*-мереж). Його основна мета — автоматичне налаштування коефіцієнта навчання η на кожному етапі, що сприяє поліпшенню збіжності і стабільності алгоритму.

Алгоритм Даркена — Муді часто використовують для навчання *RBF*-мереж та інших моделей, де необхідне точне налаштування параметрів у процесі навчання. Його також застосовують у завданнях кластеризації та оброблення сигналів, де потрібно автоматично налаштовувати параметри для досягнення оптимальних результатів.

Для алгоритму Даркена — Муді запишемо:

$$\eta(k) = \eta_0 / (1 + k/T). \tag{1.58}$$

Коефіцієнт T позначає постійну часу, підбирається індивідуально для кожної задачі. При $k < T$ значення η є практично незмінним, але за умови $k > T$ воно поступово зменшується до нуля. Незважаючи на те, що адаптивні методи підбору η є найпрогресивнішими порівняно з випадком використання сталого значення η, їх також не можна вважати найкращим рішенням, особливо у разі моделювання динамічних процесів.

Після фіксації розміщення центрів виконують налаштування параметрів σ_j, що відповідають конкретним базисним функціям. Параметр s_j радіальної функції впливає на форму і розмір зони, в якій значення цієї функ-

ції відрізняється від нуля (точніше, перевищує певне порогове значення δ). Підбір σ_j має здійснюватися так, щоб зони охоплення всіх радіальних функцій покривали весь простір вхідних даних, при цьому допускається лише незначне перекриття між будь-якими двома зонами. За такого підходу до вибору значень σ_j відображення функції, реалізоване радіальною мережею, буде відносно монотонним.

Найпростішим способом вибору σ_j для *j*-ї радіальної функції є використання евклідової відстані між *j*-м центром c_j та його найближчим сусідом. В іншому підході, що враховує ширше сусідство, на значення σ_j впливає відстань між *j*-м центром c_j та його *P* найближчими сусідами. У цьому випадку значення σ_j визначають за формулою

$$\sigma_j = \sqrt{1 / P \sum_{k=1}^{P} ||C_j - C_k||^2}). \quad (1.59)$$

Насправді значення *P* зазвичай лежить в інтервалі [3, 5].

У ході виконання будь-якого завдання ключова проблема, що визначає якість відображення, полягає у попередньому підборі кількості радіальних функцій (прихованих нейронів). Як правило, при цьому керуються загальним принципом: чим більша розмірність вектора *x*, тим більша кількість радіальних функцій потрібна для отримання задовільного рішення. Докладний опис процесу добору кількості радіальних функцій наведено далі.

Імовірнісний алгоритм підбору параметрів радіальних функцій

Вимоги до кількості прихованих нейронів можна пом'якшити, застосувавши мережу типу *HRBF*, що реалізує радіальне відображення з використанням евклідової метрики. Коефіцієнти $Q_{jk}^{(i)}$ матриці масштабування Q_i пов'язані з відповідними компонентами вектора *x*, що визначають *i*-ту радіальну функцію (див. рис. 1.9). Вони являють собою ще одну групу параметрів, що підлягають підбору і полегшують апроксимацію навчальних даних радіальною мережею. Завдяки збільшенню кількості параметрів, які підбирають, необхідну точність можна досягти *HRBF*-мережею за меншого числа нейронів.

На рис. 1.12 наведено зразкову залежність похибки класифікації 10-вимірних навчальних даних для радіальної мережі *RBF* та для мережі *HRBF* з діагональною матрицею *Q*. Окрім помітного зниження рівня похибки класифікації для мережі *HRBF,* кількість прихованих нейронів було зменшено зі 160 (мережа *RBF*) до 110 (мережа *HRBF)*.

Оптимальні значення центрів та коефіцієнтів Q_{ij} для кожної базової функції можна підібрати за допомогою модифікованого алгоритму, що змінює одночасно характеристики центрів і матрицю *Q*.

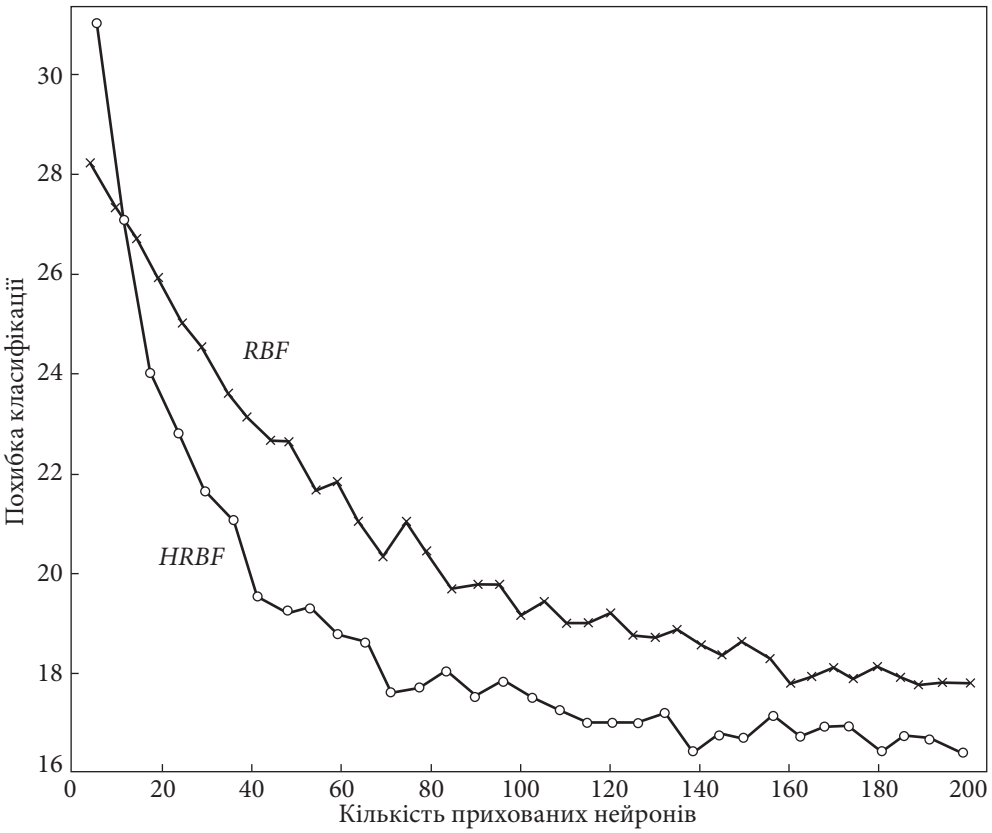


Рис. 1.12. Зразкова залежність (у відсотках) похибки класифікації 10-вимірних навчальних даних

Одним з таких алгоритмів, розроблених для мережі *HRBF* з діагональною матрицею, що масштабує Q, є ймовірнісний алгоритм [1.11]. За умови рівномірного розподілу навчальних даних x і використання діагональної матриці масштабування Q процес адаптації центрів та елементів матриці Q_i описується такими рекурентними співвідношеннями:

$$c_i(k+1) = \frac{c_i(k) + \alpha_k[\varphi_i(x_k)[x_k - c_i(k)]]}{(1-\alpha_k) + \alpha_k\varphi_i(x_k)}; \tag{1.60}$$

$$F_i(k+1) = \frac{F_i(k) + \alpha_k[\varphi_i(x_k)[x_k - c_i(k)][x_k - c_i(k)]^T - F_i(k)]}{(1-\alpha_k) + \alpha_k\varphi_i(x_k)}, \tag{1.61}$$

де $\alpha_k = \frac{\alpha_0}{k}$ позначає коефіцієнт навчання, що змінюється в часі, а α_0 — константа, підбирається з інтервалу [0, 1] (найчастіше значення α_0 перебуває в межах [0,5, 0,8]).

За цим методом радіальну функцію $\varphi(x)$ визначають так:

$$\varphi(x) = \exp(-\frac{1}{2}[x_k - c_i(k)]^T F_i^{-1}[x_k - c_i(k)]), \quad (1.62)$$

де $F_i = \text{diag}[F_{11}, F_{22}, \ldots, F_{NN}]$. Її значення відповідає умовній імовірності того, що вектор x належить до кластера з центром c_i. За такого визначення матриці F_i вона пов'язана з матрицею масштабування Q_i, яка використовується мережею *HRBF* (див. рис. 1.10), і описується співвідношенням

$$Q_i = \sqrt{\frac{1}{2} F_i^{-1}}. \quad (1.63)$$

Якщо обидві матриці мають діагональну структуру, то $Q_{jj} = \sqrt{\frac{1}{2F_{jj}}}$.

У залежностях (1.60) і (1.61) на кожному етапі виконання алгоритму одночасно відбувається адаптація як центрів, так і матриці ваг F, причому уточнюються параметри всіх радіальних функцій мережі. Це є відмінною особливістю методу, що розглядається, від описаного раніше алгоритму K-середніх, відповідно до якого уточнювалося значення лише одного центру — переможця в конкурентній боротьбі. Наведені формули можна застосовувати і для мережі *RBF* за умови, що $F_i = 1$ і що відповідно до виразу (1.61) уточнюються параметри центру, який має найбільше значення функції $F_i(x)$.

Гібридний алгоритм навчання радіальних мереж

За гібридного алгоритму процес навчання поділяється на два етапи [1.11]:

1) підбір лінійних параметрів мережі (ваги вихідного шару) у разі використання методу псевдоінверсії;

2) адаптація нелінійних параметрів радіальних функцій (центру c_i і ширини σ_i цих функцій).

Обидва етапи тісно пов'язані між собою. У разі фіксації конкретних значень центрів і ширини радіальних функцій (у перший момент це будуть початкові значення) за один крок з використанням декомпозиції *SVD (Singular Value Decomposition)* підбираються значення лінійних ваг вихідного шару. Така фіксація параметрів радіальних функцій дає змогу віднайти значення самих функцій $F_i(x_k)$ для $i = 1, 2, \ldots, K$ і $k = 1, 2, \ldots, p$, де i — номер радіальної функції, k — номер чергової навчальної пари (x_k, d_k).

Чергові значення x_k генерують у прихованому шарі сигнали, що описуються векторами:

$$\varphi_k = [1, \varphi_1(x_k), \varphi_2(x_k), \ldots, \varphi_K(x_k)],$$

де 1 позначає одиничний сигнал поляризації. Їм відповідає вихідний сигнал мережі $y_k = \varphi_k w$, причому вектор w містить ваги вихідного шару:

$$w = [w_0, w_1, \ldots, w_K]^T.$$

За наявності p навчальних пар отримуємо таку систему рівнянь:

$$\begin{bmatrix} 1 & \varphi_1(x_1) & \varphi_2(x_1) & \dots & \varphi_K(x_1) \\ 1 & \varphi_1(x_2) & \varphi_2(x_2) & \dots & \varphi_K(x_2) \\ \dots & \dots & \dots & \dots & \dots \\ 1 & \varphi_1(x_p) & \varphi_2(x_p) & \dots & \varphi_K(x_p) \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \dots \\ w_K \end{bmatrix} = \begin{bmatrix} y_1 \\ y_{2p} \\ \dots \\ y_p \end{bmatrix}, \tag{1.64}$$

яку можна записати у векторній формі:

$$Gw = y. \tag{1.65}$$

У разі застосування гібридного методу на етапі підбору вихідних ваг вектор Gw замінюється вектором очікуваних значень $d = [d_1, d_2, \dots, d_p]^T$, і утворена при цьому система рівнянь $Gw = d$ роз'язується за один крок з використанням алгоритму розрахунку псевдоінверсії [1.11]:

$$w = G^+ d. \tag{1.66}$$

В алгоритмі розрахунку псевдоінверсії застосовується декомпозиція *SVD*, що дає змогу отримати поточне значення вектора w відповідно до формули (1.53).

На другому етапі за фіксованих значень вихідних ваг сигнали збудження пропускаються мережею до вихідного шару, що уможливлює обчислення похибки для послідовності векторів x_k. Далі відбувається повернення до прихованого шару (зворотне поширення). З використанням отриманої похибки визначається вектор градієнта цільової функції щодо конкретних центрів c_{ij} і ширини σ_{ij}. Для наступного викладу припустимо, що використовується модель мережі типу *HRBF* з діагональною формою матриці, що масштабує Q. Це означає, що кожна радіальна функція визначається у загальному вигляді:

$$\varphi_i(x_k) = \exp(-\frac{1}{2} u_{ik}), \tag{1.67}$$

де сумарний сигнал нейрона u_{ik} описується виразом:

$$u_{ik} = \sum_{j=1}^{N} \frac{(x_{jk} - c_{ij})^2}{\sigma_{ij}^2}. \tag{1.68}$$

За наявності p навчальних пар цільову функцію можна записати у вигляді:

$$E = \frac{1}{2} \sum_{k=1}^{p} [y_k - d_k]^2 = \frac{1}{2} \sum_{k=1}^{p} [\sum_{i=1}^{K} w_i \varphi(x_k) - d_k]^2. \tag{1.69}$$

У результаті диференціювання цієї функції отримуємо [1.11]:

$$\frac{\partial E}{\partial c_{ij}} = \sum_{k=1}^{p} [(y_k - d_k) w_i \exp(-\frac{1}{2} u_{ik}) \frac{(x_{jk} - c_{ij})}{\sigma_{ij}^2}, \tag{1.70}$$

$$\frac{\partial E}{\partial \sigma_{ij}} = \Sigma_{k=1}^{p}[(y_k - d_k)w_i \exp(-\frac{1}{2}u_{ik})\frac{(x_{jk} - c_{ij})^2}{\sigma_{ij}^3}. \quad (1.71)$$

Застосовуючи градієнтний метод якнайшвидшого спуску, можна виконати уточнення центрів та ширини радіальних функцій згідно з формулами:

$$c_{ij}(n+1) = c_{ij}(n) - \eta\frac{\partial E}{\partial c_{ij}}, \quad (1.72)$$

$$\sigma_{ij}(n+1) = \sigma_{ij}(n) - \eta\frac{\partial E}{\partial \sigma_{ij}}. \quad (1.73)$$

Уточнення нелінійних властивостей радіальної функції завершує черговий цикл навчання. Багаторазове повторення обох етапів забезпечує повне та швидке навчання мережі, особливо якщо початкові значення параметрів радіальних функцій близькі до оптимальних.

Насправді виділені етапи різною мірою впливають на адаптацію параметрів. Як правило, алгоритм *SVD* працює з більшою швидкістю (він за один крок знаходить локальний мінімум функції). Для вирівнювання цієї диспропорції одне уточнення лінійних параметрів зазвичай супроводжується декількома циклами адаптації нелінійних параметрів.

Приклад використання радіальної мережі

Нейронні мережі з радіальними базисними функціями застосовують як для виконання завдань класифікації чи апроксимації функцій багатьох змінних, так і для прогнозування, тобто у прикладних сферах, де сигмоїдальні мережі краще зарекомендували себе впродовж багатьох років. Вони виконують ті самі функції, що й сигмоїдальні мережі, однак реалізуються на базі інших методів оброблення даних, пов'язаних з локальними відображеннями. Завдяки цій особливості забезпечується значне спрощення та прискорення процесу навчання.

Як приклад розглянемо апроксимацію тривимірної функції, що описується формулою:

$$d = f(x_1, x_2) = 3(1 - x_1)^2 \cdot \exp(-x_1^2 - (x_2 + 1)^2) -$$

$$- 10(\frac{x_1}{5} - x_1^3 - x_2^5) \cdot \exp(- x_1^2 - x_2^2) - \frac{1}{3}\exp(- (x_1 + 1)^2 - x_2^2).$$

Нехай змінні перебувають у межах $-3 \le x_1 \le$ та $-3 \le x_2 \le 3$. Навчальні дані мають рівномірний розподіл в областях визначення змінних x_1 і x_2. Загалом для навчання використовувалося 625 екземплярів навчальних даних у вигляді пар даних ($[x_1, x_1]$, d). Для вирішення задачі було побудовано ме-

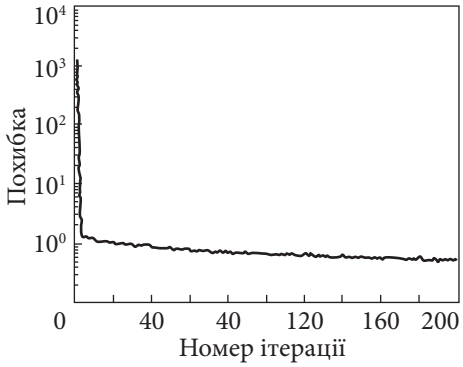


Рис. 1.13. Крива зміни похибки зі збільшенням кількості ітерацій

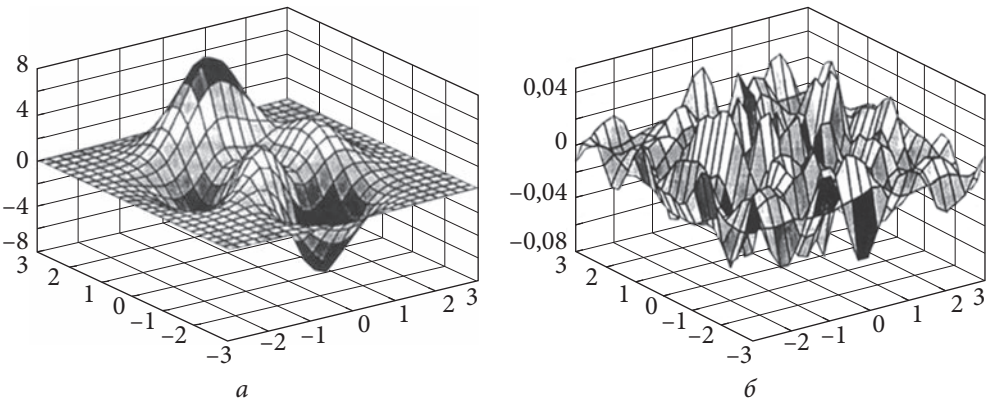


Рис. 1.14. Результати відновлення тривимірної функції $F(x_1, x_2)$ радіальною мережею *RBF: а* — відновлена поверхня, *б* — похибка відновлення даних (характеристика навченості мережі)

режу зі структурою: два входи для x_1 і x_2 відповідно, 36 радіальних нейронів гауссівського типу та один вихідний лінійний нейрон, що відповідає значенню *d* функції) (структура 2-36-1). Застосовували гібридний алгоритм навчання з випадковим вибором початкових значень параметрів мережі. На рис. 1.13 зображено графік навчання радіальної мережі *RBF* для прикладу відновлення тривимірної функції (крива зміни похибки зі збільшенням кількості ітерацій).

З графіка видно, що прогрес у зменшенні похибки досить великий, особливо у початковій стадії процесу.

Результати відновлення тривимірної функції $F(x_1, x_2)$ радіальною мережею *RBF* наведено на рис. 1.14.

Максимальна похибка відновлення не перевищила рівня 0,06, що становить близько 1 % очікуваного значення [1.14]. Порівняння швидкості навчання та навчальних властивостей радіальної мережі з аналогічними показниками багатошарового перцептрона однозначно засвідчить про пе-

ревагу першої. Вона швидше навчається і набагато менш чутлива до початкових значень параметрів як базисних функцій і ваг вихідного нейрона.

Методи вибору кількості базисних функцій

Вибір кількості базисних функцій, кожній з яких відповідає один прихований нейрон, є основною проблемою, що виникає під час коректного розв'язання задачі апроксимації. Як і у випадку сигмоїдальних мереж, занадто мала кількість нейронів не дає можливості досягти достатньої точності узагальнення на множині навчальних даних, тоді як надто велика їх кількість може збільшити похибку на тестовій множині. Оптимальна кількість нейронів залежить від багатьох факторів, як-от розмірність завдання, обсяг навчальних даних та, передусім, просторова структура апроксимованої функції. Зазвичай кількість базисних функцій K є певною часткою обсягу навчальних даних p, причому фактична величина цієї частки залежить від розмірності вектора x та розкиду очікуваних значень d_i, відповідних вхідним векторам для $i = 1, 2, \ldots, p$.

Евристичні методи. Через неможливість апріорного визначення точної кількості прихованих нейронів застосовують адаптивні методи, які дають можливість додавати або видаляти нейрони у процесі навчання. Розроблено багато евристичних методів для реалізації цих операцій [1.11]. Зазвичай навчання мережі починається з будь-якої попередньо прийнятої кількості нейронів, а надалі контролюють зменшення середньоквадратичної похибки та зміну параметрів мережі. Якщо середня зміна ваг після певної кількості навчальних циклів замала, то додаються дві базисні функції (два нейрони) з центрами, що відповідають найбільшій та найменшій похибці адаптації. Після цього навчання розширеної структури продовжують. Одночасно контролюють абсолютні значення ваг окремих нейронів. Коли значення ваги менше за початково встановлений поріг δ, відповідні нейрони видаляють з мережі. Додавання і видалення нейронів може виконуватися на будь-якому етапі навчання до досягнення необхідної точності.

Інший підхід до керування кількістю прихованих нейронів на початку 1990-х рр. запропонував американський вчений і дослідник у галузі машинного навчання та штучного інтелекту Дж. Платт (John Platt) [1.11]. Цей підхід, що поєднує елементи самоорганізації з навчанням під наглядом, застосовується в моделях, де необхідно динамічно змінювати кількість нейронів залежно від просторової структури даних та вимог до точності апроксимації. Після подання кожного навчального зразка обчислюється евклідова відстань між ним і центром найближчої наявної радіальної функції. Якщо ця відстань перевищує заданий поріг TM(k), створюється новий центр радіальної функції (тобто додається нейрон), після чого мережа навчається за

допомогою градієнтних методів (навчання з учителем). Додавання нейронів триває до досягнення необхідного рівня точності.

Важливим аспектом цього методу є вибір значення ТМ(k), яке визначає рішення щодо розширення мережі. Зазвичай ТМ(k) змінюється експоненційно з часом (залежно від кількості ітерацій), від значення $ТМ_{min}$ на початку процесу до $ТМ_{max}$ наприкінці. Недолік цього підходу полягає в тому, що він не дає можливості зменшити кількість нейронів навіть тоді, коли деякі з них виявляються непотрібними (через невдало вибрані центри) або дублюють функцію інших нейронів. Крім того, метод дуже чутливий до вибору параметрів навчання, особливо значень $ТМ_{min}$ та $ТМ_{max}$.

Порівняння радіальних і сигмоїдальних мереж

Радіальні нейронні мережі належать до категорії мереж із навчанням під наглядом, так само як і багатошаровий перцептрон. Порівняно з багатошаровими мережами, які використовують сигмоїдальні функції активації, вони мають деякі специфічні властивості, що забезпечують простіше відображення характеристик моделювання процесу.

Сигмоїдальна мережа, де ненульове значення сигмоїдальної функції поширюється від певної точки в просторі до нескінченності, вирішує завдання глобальної апроксимації заданої функції. Водночас радіальна мережа, побудована на функціях, що мають ненульові значення лише в певній ділянці навколо своїх центрів, забезпечує апроксимацію локального типу, сфера якої, як правило, є більш обмеженою. Отже, слід розуміти, що здатність радіальних мереж до узагальнення дещо слабша ніж сигмоїдальних, особливо на межах області навчальних даних. Завдяки глобальному характеру сигмоїдальної функції багатошарові мережі не мають вбудованого механізму ідентифікації області даних, на яку найсильніше реагує конкретний нейрон. Через фізичну неможливість пов'язати зону активності нейрона з відповідною областю навчальних даних у сигмоїдальних мережах складно визначити початкову позицію для навчання. З огляду на полімодальність цільової функції, досягнення глобального мінімуму в такій ситуації стає надзвичайно складним навіть за наявності вдосконалених методів навчання.

Радіальні мережі вирішують цю проблему ефективніше. Найчастіше використовувані в практиці радіальні функції гауссового типу за своєю природою мають локальний характер і набувають ненульових значень тільки в ділянці навколо певного центру. Це дає змогу легко встановити залежність між параметрами базисних функцій і просторовим розміщенням навчальних даних у багатовимірному просторі, що полегшує вибір задовільних початкових умов для навчання з учителем. Застосування алгорит-

мів навчання за початкових умов, близьких до оптимальних, значно збільшує ймовірність успішного навчання радіальних мереж.

Вважається [1.11, 1.14], що радіальні мережі краще виконують такі класифікаційні завдання, як виявлення пошкоджень у системах, розпізнавання образів тощо. Їх використання для прогнозування складних часових процесів, як-от щомісячні коливання зайнятості населення в масштабах країни, економічні тренди тощо, також дає досить добрі результати, порівнянні або навіть кращі від отриманих за допомогою сигмоїдальних мереж.

Важливою перевагою радіальних мереж є спрощений алгоритм навчання. Завдяки наявності лише одного прихованого шару і тісному зв'язку активності нейрона з відповідною областю простору навчальних даних початкова точка навчання розміщена набагато ближче до оптимального рішення, ніж у багатошарових мережах. Крім того, можна розділити етап налаштування параметрів базисних функцій і налаштування ваг мережі (гібридний алгоритм), що спрощує і прискорює процес навчання. Прискорення навчання стає ще більшим завдяки процедурі формування оптимальної структури мережі з погляду узагальнювальної здатності. Для багатошарових мереж це досить трудомістке завдання, яке зазвичай потребує багаторазового повторення навчання або додаткового навчання. Для радіальних мереж, особливо заснованих на ортогоналізації, формування оптимальної структури є природним етапом процесу навчання і не потребує додаткових зусиль.

Висновки

У цьому розділі докладно описано і охарактеризовано одну з перших і найбільш розвинених технологій штучного інтелекту — нейронні мережі. Розглянуто нейронну мережу зі зворотним поширенням помилки (*Back Propagation*), описано її архітектуру та властивості. Сформульовано теорему про універсальну апроксимацію для нейронних мереж *Back Propagation*, яка обґрунтовує універсальність і застосування цього класу мереж у завданнях прогнозування, апроксимації та класифікації.

Детально проаналізовано основні алгоритми навчання нейронних мереж: градієнтний спуск, спряжені градієнти та генетичний алгоритм, описано їхні властивості та недоліки.

Розглянуто інший клас нейронних мереж — мережі з радіально-базисними функціями (РБФ). Наведено архітектуру цих мереж і основи їх теорії. Описано алгоритм їх навчання — гібридний алгоритм, проаналізовано його властивості та здійснено порівняння з класичним алгоритмом градієнтного спуску. Визначено перспективні галузі застосування РБФ-мереж. Також розглянуто складніший варіант цього класу мереж — мережі з гіпер-РБФ

та їхні властивості. Проведено порівняльний аналіз нейронних мереж *Back Propagation* і РБФ-мереж. Наведено приклад застосування РБФ-мережі для завдання апроксимації функцій. Описано сучасні методи визначення кількості нейронів у задачах, що мають важливе практичне значення.

СПИСОК ЛІТЕРАТУРИ

1.1. Engelbrecht A.P. Introduction to Computational Intelligence. Wiley, 2007. 628 p.

1.2. Zgurovsky M., Zaychenko Yu. The Fundamentals of Computational Intelligence: System Approach. Springer, 2016. 375 p.

1.3. Poole D.L., Mackworth A.K. Artificial Intelligence: Foundations of Computational Agents. Cambridge University Press, 2010. 662 p.

1.4. Alippi C. Neural Networks and Genetic Algorithms. Springer, 1997. 268 p.

1.5. Haykin S. Neural Networks: A Comprehensive Foundation. Prentice Hall, 1998. 842 p.

1.6. Schalkoff R.J. Intelligent Systems: Principles, Paradigms, and Pragmatics. Jones & Bartlett, 2011. 608 p.

1.7. Dennis A., Wixom B.H., Roth R.M. Systems Analysis and Design. Wiley, 2015. 498 p.

1.8. Kohonen T. Self-Organizing Maps. Springer, 2001. 362 p.

1.9. Luger G.F. Artificial Intelligence: Structures and Strategies for Complex Problem Solving. Pearson, 2005. 896 p.

1.10. Minsky M., Papert S. Perceptrons: An Introduction to Computational Geometry. MIT Press, 1969. 258 p.

1.11. Osowski S. Neural Networks for Information Processing. Springer, 1994. 300 p.

1.12. Rosenblatt F. Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms. Spartan Books, 1962. 460 p.

1.13. Rutkowska D., Pilinski M., Rutkowski L. Neural Networks, Genetic Algorithms, and Fuzzy Systems. Springer, 1997. 368 p.

1.14. Haykin S. Neural Networks and Learning Machines. Pearson, 2008. 936 p.

1.15. Bezdek J.C. On the relationship between neural networks, pattern recognition and intelligence. *Int. J. Approximate Reasoning.* 1992. Vol. 6. P. 85–107.

1.16. Bezdek J.C. What is computational intelligence? In: *Computational Intelligence: Imitating Life*. IEEE Press, 1994. P. 1–12.

1.17. Engelbrecht A. Computational Intelligence: An Introduction. 2nd ed. John Wiley & Sons, 2007. 630 p.

1.18. Hornik K., Stinchcombe M., White H. Multilayer feedforward networks are universal approximators. *Neural Networks.* 1989. Vol. 2. P. 359–366.

1.19. Jones T. Artificial Intelligence: A Systems Approach. Infinity Science Press, 2008. 518 p.

1.20. Kosko B. Neural Networks and Fuzzy Systems. Prentice Hall, 1992. 449 p.

1.21. Marks R.J. Intelligence: Computational versus Artificial. *IEEE Transactions on Neural Networks.* 1993. No. 4(5). P. 737–739.

1.22. Russell S.J. Human Compatible: Artificial Intelligence and the Problem of Control. Viking, 2019. 352 p.

РОЗДІЛ 2

НЕЙРОННІ МЕРЕЖІ ЗІ ЗВОРОТНИМИ ЗВ'ЯЗКАМИ І САМО-ОРГАНІЗАЦІЄЮ

2.1. Нейронна мережа Хопфілда та її застосування

2.2. Нейронна мережа Хемінга.
Архітектура і алгоритм роботи

2.3. Самоорганізаційні (самоорганізовані)
нейронні мережі. Алгоритми навчання Кохонена

2.4. Застосування нейронних мереж Кохонена

Висновки

Список літератури

2.1. Нейронна мережа Хопфілда та її застосування

Відродження інтересу до нейронних мереж значною мірою пов'язане з роботами вже згадуваного вище у тексті Джона Хопфілда (John Hopfield, США)[1] — американського вченого, відомого як винахідника асоціативної нейронної мережі, репрезентованої у 1982 р. Ця мережа під назвою «мережа Хопфілда» стала однією з перших моделей, яка продемонструвала практичне використання властивостей нейроноподібних систем.

Роботи Хопфілда відкрили шлях до розуміння того, як можна використовувати мережі з нейроноподібних елементів для обчислювальних задач. Його підхід став потужним стимулом для досліджень у багатьох наукових галузях, поставивши перед дослідниками дві ключові цілі:

1) глибше зрозуміти, як працює мозок;

2) використати властивості нейронних мереж для вирішення задач, що не піддаються розв'язанню традиційними методами.

Ідея рекурентності

Нейронна мережа Хопфілда є прикладом **динамічної системи зі зворотними зв'язками**, в якій вихід однієї операції використовується як вхід для наступної. Бінарну версію мережі Хопфілда наведено на рис. 2.1.

Мережі, що працюють як системи зворотного зв'язку, називаються *рекурентними нейронними мережами*. Кожна пряма операція в такій мережі визначається як ітерація. Як і інші нелінійні динамічні системи, рекурентні мережі можуть демонструвати широкий спектр поведінкових режимів:

• *стійкість (збіжність)*. Система сходиться до фіксованої (нерухомої) точки. Це означає, що у разі введення нерухомої точки як вхідного значення вихід залишається таким самим. У такому стані система залишається зафіксованою;

• *періодичні цикли.* Система повторює певні стани через визначений інтервал;

• *хаотична поведінка.* Система демонструє непередбачувані й нестабільні зміни.

[1] Дж. Хопфілд — член Національної академії наук США, з 2024 р. — лауреат Нобелівської премії з фізики за видатні досягнення в галузі нейронних мереж.

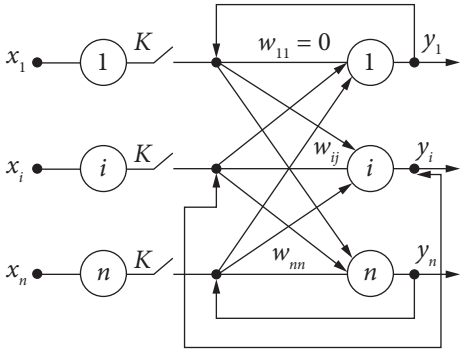


Рис. 2.1. Бінарна мережа Хопфілда

Було доведено, що мережі Хопфілда здебільшого є стійкими. У загальному випадку мережа може мати кілька фіксованих точок. Те, до якої саме точки мережа збігається, залежить від вихідної точки, обраної для початкової ітерації. Інакше кажучи, поведінка системи залежить від початкових умов.

Ця здатність до самостійної збіжності та стійкості дає змогу застосовувати мережі Хопфілда для вирішення завдань оптимізації, пошуку локальних мінімумів та збереження асоціативних патернів.

Атрактори та їхні басейни притягання в мережі Хопфілда. Нерухомі точки нейронної мережі, до яких вона сходиться в процесі ітерацій, називаються *атракторами*. Множина точок (векторів), що притягуються до конкретного атрактора під час ітерацій, утворює *басейн притягання* цього атрактора.

У мережі Хопфілда множина нерухомих точок виступає як її *пам'ять*. Це означає, що мережа здатна діяти як *асоціативна пам'ять*, де вхідні вектори, що перебувають у басейні притягання певного атрактора, пов'язані (асоційовані) з ним.

Наприклад, атрактор може бути *бажаним образом* (збереженим у пам'яті мережі). Басейн притягання цього атрактора може містити зашумлені або частково відсутні версії цього образу. Отже, мережа має здатність «згадувати» повний бажаний образ, навіть якщо на вхід подаються його неповні або змінені версії.

Ця властивість мережі Хопфілда уможливлює її використання для завдань розпізнавання образів, корекції помилок і реконструкції інформації.

Бінарні мережі Хопфілда

Бінарну мережу Хопфілда зображено на рис. 2.1. Вхідні та вихідні вектори складаються з «–1» та «+1» (замість «–1» може бути використано 0). Є симетрична вагова матриця, що складається з цілих чисел з нулями по діагоналі $W = ||w_{ij}||$. Вхідний вектор X множать на вагову матрицю, використовуючи звичайне матрично-векторне множення. Він подається на входи відповідних нейронів, і визначається вихідний вектор. Однак тільки одна компонента вихідного вектора $Y = [y_i]$ додається до сигналів зворотного зв'язку на кожній ітерації. Ця компонента, яка може бути обрана випадко-

во або вибирається по черзі, замінює відповідну компоненту вхідного вектора *i*, отже, утворює вхідний вектор для наступної ітерації. Процес триває до того часу, доки вхідні й вихідні вектори не стануть однаковими (тобто доки не буде досягнуто нерухомої точки). Алгоритм цього процесу, який називають процедурою асинхронної корекції, описано далі.

Опис алгоритму асинхронної корекції

У перший момент ключ K замкнутий, вхідний вектор x подається з вагами $W = ||w_{ij}||$ на входи нейронів і знаходиться сумарний сигнал на вході нейрона $S_j(x)$. Далі ключ K розмикається і виходи нейронів подаються на їх входи.

Опишемо ітерацію алгоритму асинхронної корекції [2.1].

1. Обчислити компоненти вихідного вектора y_j, $j = 1, 2, \ldots, n$, за формулою:

$$y_j = T(\Sigma_{i=1}^{n} w_{ij} x_i), \tag{2.1}$$

де $T(x) = -1$, якщо $x < 0$, $T(x) = 1$, якщо $x > 0$.

2. Виконати асинхронну корекцію, тобто:

- розпочати з вхідного вектора $(x_1, x_2, \ldots, x_n)$;
- знайти y_j за формулою (2.1);
- замінити $(x_1, x_2, \ldots, x_n)$ на $(y_1, x_2, x_3, \ldots, x_n) = Y$ і подати Y на вхід X;
- повторити процес, щоб знайти y_2, y_3 і т. д. і замінити ними відповідні входи мережі.

3. Повторити кроки 2, 3, доки вектор $Y = (y_1, y_2, \ldots, y_n)$ не припинить змінюватися.

Було доведено, що якщо змінюється стан хоча б одного з виходів, то кожен крок зменшує енергію зв'язків *E:*

$$E = \frac{1}{2} \Sigma_{i=1}^{n} \Sigma_{j=1}^{n} w_{ij} x_i x_j \tag{2.2}$$

так, що забезпечується збіжність мережі до нерухомої точки (атрактор).

Асинхронна корекція, разом із наявністю нулів на діагоналі матриці W, забезпечує зменшення енергетичної функції (2.2) з кожною ітерацією. Це є ключовим фактором для забезпечення збіжності мережі до нерухомої точки. Якщо допустити, щоб увесь вектор одночасно коригувався на кожній ітерації, то в результаті можна отримати мережу, в якій термінальні стани атракторів будуть представлені періодичними циклами, а не нерухомими точками.

Асинхронна корекція виключає таку поведінку, даючи можливість коригувати лише одну змінну за ітерацію, що гарантує стабільність системи та її збіжність до нерухомої точки.

Зразки поведінки мережі Хопфілда

Одним із ключових аспектів, який визначає поведінку мережі Хопфілда, є її *вагова матриця*. Це порушує важливе питання: *як правильно визначити вагову матрицю*?

Відповідь: потрібно задати певні вагові вектори, які називаються *еталонними екземплярами*. Ці екземпляри виконують роль бажаних станів, які мережа має зберігати як фіксовані точки (атрактори). Однак не завжди ці екземпляри автоматично стають фіксованими точками.

Для того щоб еталонні екземпляри були атракторами, вагова матриця має бути задана у спеціальному вигляді, який гарантує, що ці екземпляри стануть нерухомими точками системи. Вибір правильної вагової матриці є ключовим етапом під час проєктування мережі Хопфілда, оскільки саме вона визначає динамічну поведінку системи. Враховуючи зазначене, вагову матрицю слід задати у вигляді:

$$w_{ij} = \begin{cases} \Sigma_{k=0}^{N}(x_{kj} - 1) \cdot (x_{kj} - 1) \text{ для } i \neq j; \\ 0 \text{ для } i = j, \end{cases} \tag{2.3}$$

де N — число заданих екземплярів, а $X_k = \{x_{k_1}, x_{k_2}, ..., x_{k_n}\}$ — k-й екземпляр.

Якщо екземпляри векторів утворюють множину *ортогональних векторів*, то за умови, що вагова матриця задається, як показано вище, можна гарантувати, що кожен екземпляр стане нерухомою точкою. Проте ортогональність не є необхідною умовою для того, щоб екземпляри гарантували утворення нерухомих точок. У загальному випадку навіть неортогональні екземпляри можуть стати фіксованими точками за відповідного вибору вагової матриці.

Застосування мережі Хопфілда

Мережа Хопфілда має широкий спектр застосувань, зокрема для *розпізнавання образів*. Однак кількість образів, які така мережа може успішно розпізнати, невелика через її обмежену пам'ять.

У межах дослідження було виконано експеримент, в якому мережу навчали розпізнавати чотири літери алфавіту. Вихідні образи продемонстровано на рис. 2.2.

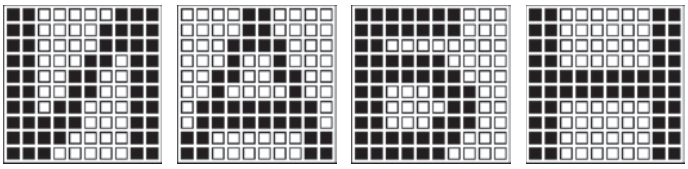


Рис. 2.2. Вихідні образи

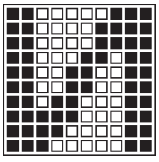
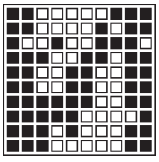
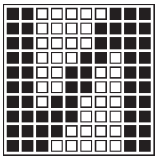
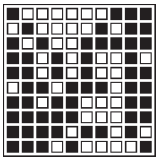
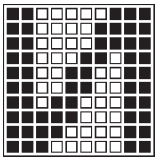
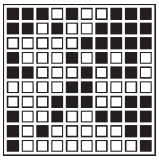
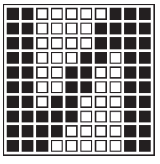
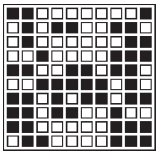
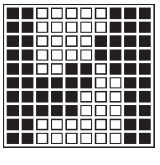
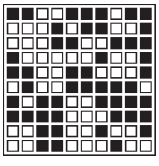
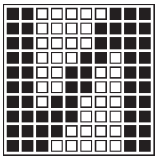
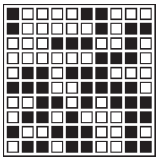
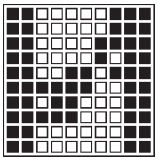
Методологія дослідження

1. Зашумленість кожного образу поступово збільшували.
2. Зашумлені образи подавалися на вхід мережі Хопфілда.
3. Результати роботи мережі аналізували й записували.

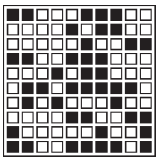
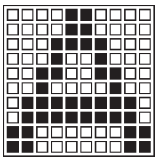
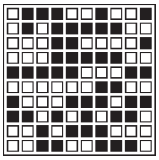
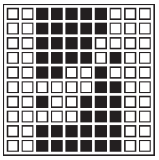
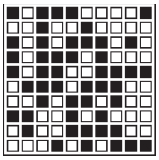
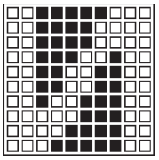
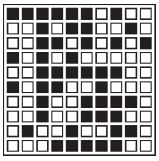
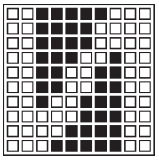
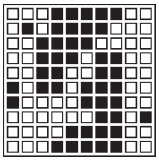
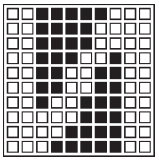
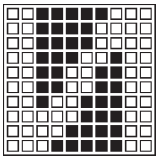
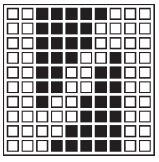
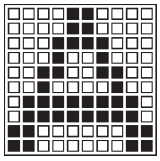
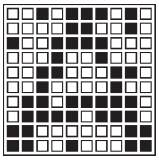
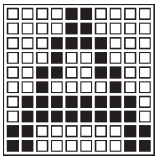
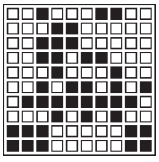
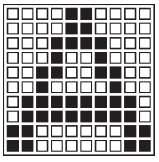
Результати роботи мережі для різних рівнів зашумленості наведено у табл. 2.1 [2.1]. Це демонструє здатність мережі Хопфілда до корекції вхідних даних і відновлення образів навіть за наявності шуму.

Таблиця 2.1

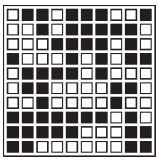
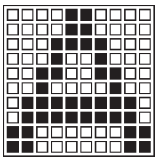
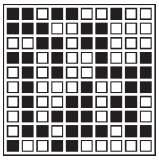
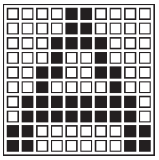
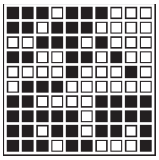
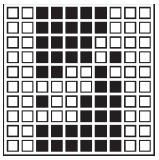
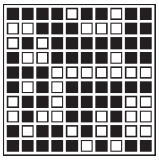
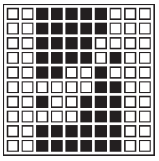
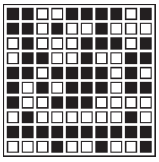
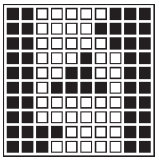
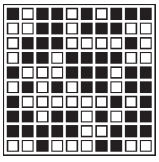
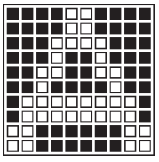
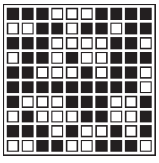
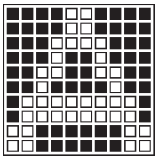
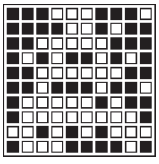
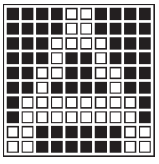
Результати роботи мережі Хопфілда

Образ, що тестується	Зашумлення образу, %	Вигляд зміненого образу	Результат розпізнавання
	10		
	20		
	30		
	35		
	40		
	45		

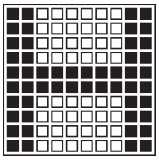
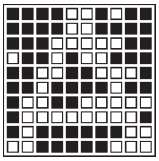
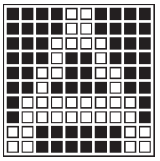
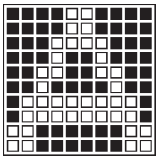
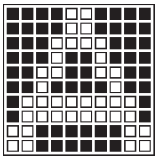
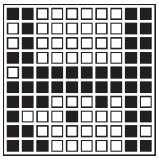
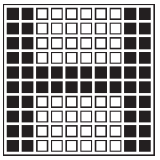
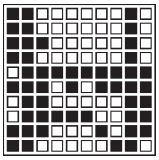
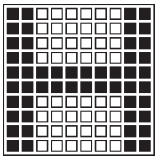
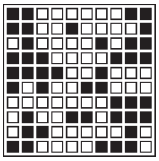
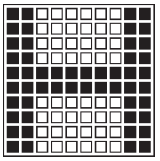
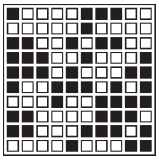
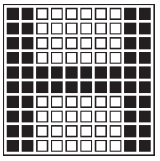
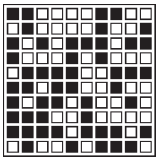
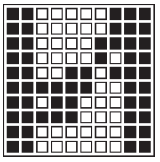
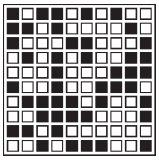
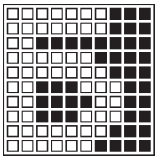
Продовження табл. 2.1

Образ, що тестується	Зашумлення образу, %	Вигляд зміненого образу	Результат розпізнавання
	50		
	60		
	70		
	80		
	90		
	100		
	10		
	20		

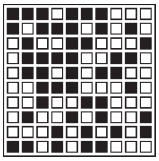
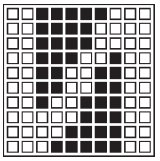
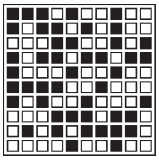
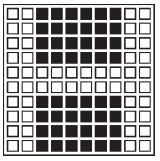
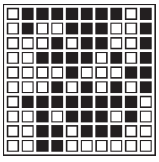
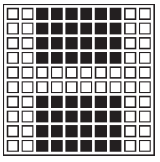
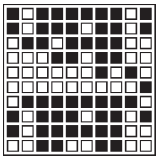
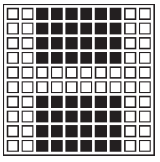
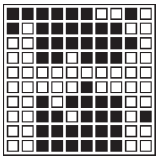
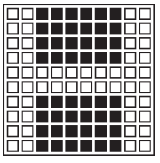
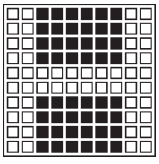
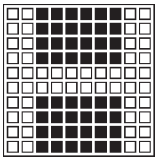
Продовження табл. 2.1

Образ, що тестується	Зашумлення образу, %	Вигляд зміненого образу	Результат розпізнавання
	30		
	40		
	45		
	50		
	60		
	65		
	70		
	80		

Продовження табл. 2.1

Образ, що тестується	Зашумлення образу, %	Вигляд зміненого образу	Результат розпізнавання
	90		
	100		
	10		
	20		
	30		
	40		
	45		
	50		

Закінчення табл. 2.1

Образ, що тестується	Зашумлення образу, %	Вигляд зміненого образу	Результат розпізнавання
	55		
	60		
	70		
	80		
	90		
	100		

Отже, нейронна мережа Хопфілда ефективно виконує завдання розпізнавання образів для експериментів з рівнем спотворення (0—40 %). У цьому діапазоні всі зразки розпізнаються без помилок (іноді з'являються незначні перешкоди для 40 %-го зашумлення).

За 45—60 % зашумлення образи розпізнаються нестабільно, часто виникає «переплутування» і на виході нейронної мережі з'являється зовсім інший еталон або його негатив. Починаючи з 60 % на виході системи з'являється негатив тестованого образу, іноді частково спотворений (при 60—70 %).

Ефект «перехресних асоціацій»

Ускладнимо завдання і навчимо нейронну мережу Хопфілда розпізнаванню ще одного прикладу, показаного на рис. 2.3.

Літера «П» дуже подібна до вже наявних у пам'яті нейронної мережі літер «И» та «Н». Тепер нейронна мережа Хопфілда не може розпізнати жодну з цих літер навіть у неспотвореному стані. Замість правильно розпізнаної літери вона видає зображення (для перекручування образу від 0 до 50 %), наведене на рис. 2.4. Воно дещо нагадує кожну з літер — «И», «Н», «П» і не є правильною інтерпретацією жодної з них.

У разі зашумлення образу на 50—60 % на виході нейронної мережі з'являється спочатку подане на рис. 2.4 зображення в злегка спотвореному вигляді, а потім його негатив (тут не наведено).

Починаючи з 65 %-го зашумлення образу на виході нейронної мережі стабільно з'являється негатив зображення.

Описана поведінка нейронної мережі відома як ефект «перехресних асоціацій». При цьому символи «А» і «Б» розпізнаються безпомилково за спотворення до 40 %. За 45—65 % на виході нейронної мережі з'являються зображення позитива або негатива, їх злегка зашумлені інтерпретації, а також зображення, подібне до негатива літери «Б» (але дуже спотвореного), або ж негатив тестованого образу. За спотворення 70 % і більше нейронна мережа стабільно розпізнає в образі, що тестується, його негатив.

Проведені експериментальні дослідження виявили такі недоліки нейронної мережі Хопфілда [2.1, 2.2]:

1) наявність перехресних асоціацій, коли кілька еталонів виявляються рдібними один до одного (як, наприклад, в експериментах з літерою «П»);

2) зважаючи на обмеження ємності запам'ятовування, кількість атракторів (еталонів), що запам'ятовуються, становить усього $(0{,}15 \div 0{,}3)n$, де n — розмірність матриці ваг W. Ці фактори істотно обмежують можливості практичного застосування нейронної мережі Хопфілда.

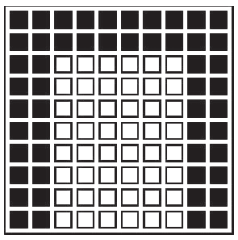


Рис. 2.3. Літера «П»

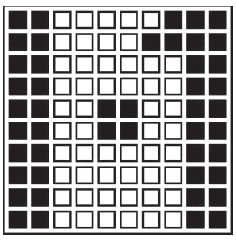


Рис. 2.4. Розпізнавання літери «П»

2.2. Нейронна мережа Хеммінга. Архітектура і алгоритм роботи

Якщо немає необхідності, щоб мережа в явному вигляді видавала зразок, тобто достатньо, наприклад, отримувати номер (або клас) зразка, асоціативну пам'ять успішно реалізує нейронна мережа американського математика Річарда Хеммінга (Richard Wesley Hamming). Його роботи мали значний вплив на розвиток інформатики й телекомунікацій.

Мережа Хеммінга характеризується меншими потребами пам'яті та обсягів обчислень порівняно з мережею Хопфілда, про що очевидно свідчить її структура (рис. 2.5).

Мережа складається з двох шарів. Перший і другий шари мають m нейронів, де m — число зразків. Нейрони першого шару мають по n синапсів, з'єднаних із входами мережі (утворюють фіктивний нульовий шар). Нейрони другого шару сполучені між собою синаптичними зв'язками. Єдиний синапс з додатним зворотним зв'язком для кожного нейрона зв'язаний з його аксоном.

Ідея роботи мережі полягає в знаходженні відстані Хеммінга від тестованого образу до всіх образів. Відстань Хеммінга — це кількість відмінних бітів у двох бінарних векторах. Мережа має вибрати зразок з мінімальною відстанню Хеммінга до невідомого вхідного сигналу, внаслідок чого буде активізовано лише один вихід мережі, що відповідає цьому зразку.

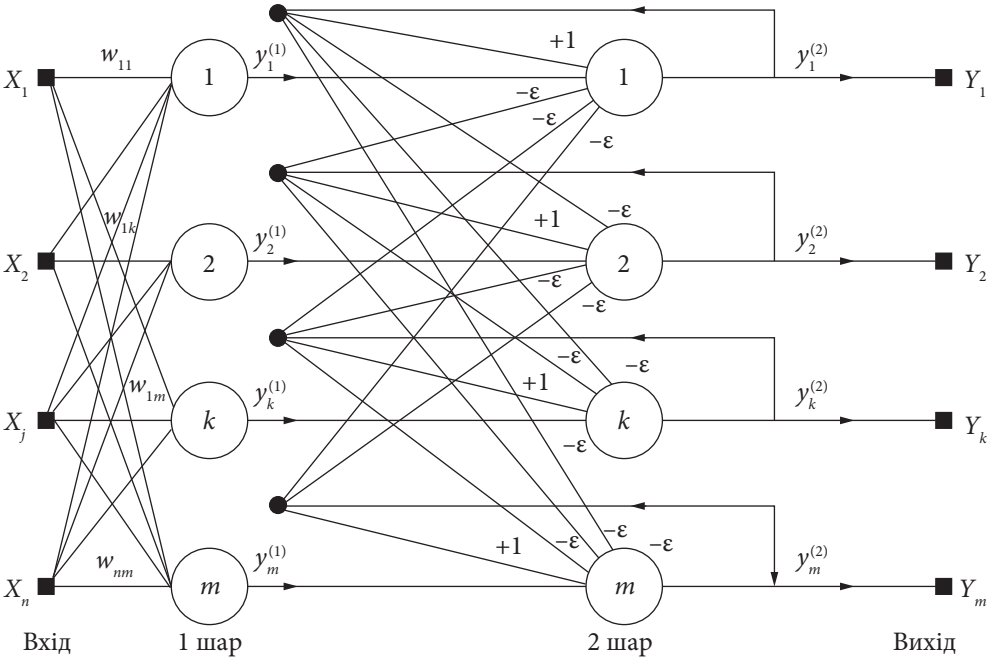


Рис. 2.5. Структура нейронної мережі Хеммінга

На стадії ініціалізації ваговим коефіцієнтам першого шару w_{ik} і порога активації функції T_k надають значення:

$$w_{ik} = \frac{x_i^k}{2}, \quad i = \overline{1, n}; \quad k = \overline{1, m}; \tag{2.4}$$

$$T_k = \frac{n}{2}, \quad k = \overline{1, m}; \tag{2.5}$$

де x_i^k — i-й елемент k-го еталонного зразка.

Вагові коефіцієнти гальмівних синапсів у другому шарі вважають такими, що дорівнюють певній величині –ε, де $0 < \varepsilon < 1/m$. Синапс нейрона, зв'язаний з його аксоном, має вагу +1.

Алгоритм функціонування мережі Хеммінга такий [1.2]:

1. На вхід мережі подають невідомий вектор $X = \{x_i: i = \overline{1, n}\}$, виходячи з якого розраховують стани нейронів першого шару (верхній індекс у дужках вказує номер шару) за формулою:

$$y_j^{(1)} = f(s_j^{(1)} = f(\Sigma_{i=1}^n w_{ij} x_i + T_j), \quad j = \overline{1, m}. \tag{2.6}$$

Після цього отриманими значеннями ініціалізують значення аксонів другого шару:

$$y_j^{(2)} = y_j^{(1)}, \quad j = \overline{1, m}. \tag{2.7}$$

Покласти $p = 0$ і перейти на $(p + 1)$-ту ітерацію.

2. Обчислюють нові стани нейронів другого шару, подавши їх виходи на входи:

$$s_j^{(2)}(p + 1) = y_j(p) - \varepsilon \Sigma_{k=1}^m y_j^{(2)}(p), \quad k \neq j, \quad j = \overline{1, m}, \tag{2.8}$$

та значення їх виходів (аксонів):

$$y_j^{(2)}(p + 1) = f[s_j^{(2)}(p + 1)], \quad j = \overline{1, m}. \tag{2.9}$$

Активаційна функція має вигляд порога, при цьому величина порога має бути досить великою, щоб будь-які можливі значення аргументів не призводили до насичення.

3. Перевіряють, чи змінилися виходи нейронів другого шару за останню ітерацію. Якщо так, то переходять до кроку 2, інакше — закінчують.

З опису алгоритму видно, що роль першого шару дуже умовна; використавши один раз на 1-му кроці значення його вагових коефіцієнтів, мережа більше не повертається до нього. Тому перший шар можна взагалі виключити з мережі, замінивши його на матрицю вагових коефіцієнтів.

Відзначимо переваги нейронної мережі Хеммінга:

- невеликі витрати пам'яті;
- швидкість роботи мережі;

• надзвичайно простий алгоритм роботи;

• ємність мережі не залежить від розмірності вхідного сигналу (як у мережі Хопфілда) і точно дорівнює кількості нейронів *m*.

Експериментальні дослідження мереж Хопфілда і Хеммінга

Постановка експерименту. Проведено порівняльні експериментальні дослідження нейронних мереж Хопфілда і Хеммінга у завданні розпізнавання зображень символів. Для навчання мережі використовують вхідну вибірку символів (1, 7, e, q, p), після чого на вхід по черзі подають зашумлені образи з цієї вибірки і відбувається їх розпізнавання [2.1]. Рівень шумів варіювався від 0 до 50 %. Результати розпізнавання символів наведено на рис. 2.6. Кожна частина рисунка містить чотири зображення (зліва направо, згори вниз): початкове зображення — еталон; зашумлене зображення; результат мережі Хеммінга; результат мережі Хопфілда.

Аналіз результатів

Результати експериментів з мережами Хопфілда і Хеммінга наведено у табл. 2.2. Елемент таблиці — результат розпізнавання мережею (Хеммінга, Хопфілда) символа за вказаного рівня шуму, а саме: 0 — не розпізнано; 0,5 — розпізнано з дефектами; 1 — розпізнано.

Мережа Хеммінга загалом впоралася добре (за винятком символа «e» розпізнавала все правильно аж до рівня шумів 40 %), а ось у мережі Хопфілда результати розпізнавання значно гірші, у розпізнаванні символів з подібним елементом («e», «p», «q» — округла частина) виникли труднощі — рівень розпізнавання становив менш як 30 %, що зумовлено ефектом «перехресних асоціацій».

Таблиця 2.2

Результати експериментів з мережами Хопфілда і Хеммінга

Символи	Розпізнавання, %, мережею Хеммінга / мережею Хопфілда					
	0	10	20	30	40	50
«1»	(1 / 1)	(1 / 0,5)	(1 / 1)	(1 / 0,5)	(1 / 0,5)	(0 / 0)
«7»	(1 / 1)	(1 / 0,5)	(1 / 0,5)	(1 / 1)	(1 / 0,5)	(0 / 0)
«e»	(1 / 1)	(1 / 0,5)	(1 / 1)	(1 / 1)	(0 / 0)	(0 / 0)
«q»	(1 / 0,5)	(1 / 0,5)	(1 / 0)	(1 / 0)	(1 / 0)	(0 / 0)
«p»	(1 / 1)	(1 / 1)	(1 / 0)	(1 / 0)	(1 / 0)	(0 / 0)

а

б

Рис. 2.6. Результати розпізнавання символів на основі еталону та зображень різного ступеня спотворення: *а* — розпізнавання символа «1», *б* — символа «7» (див. також с. 103, 104)

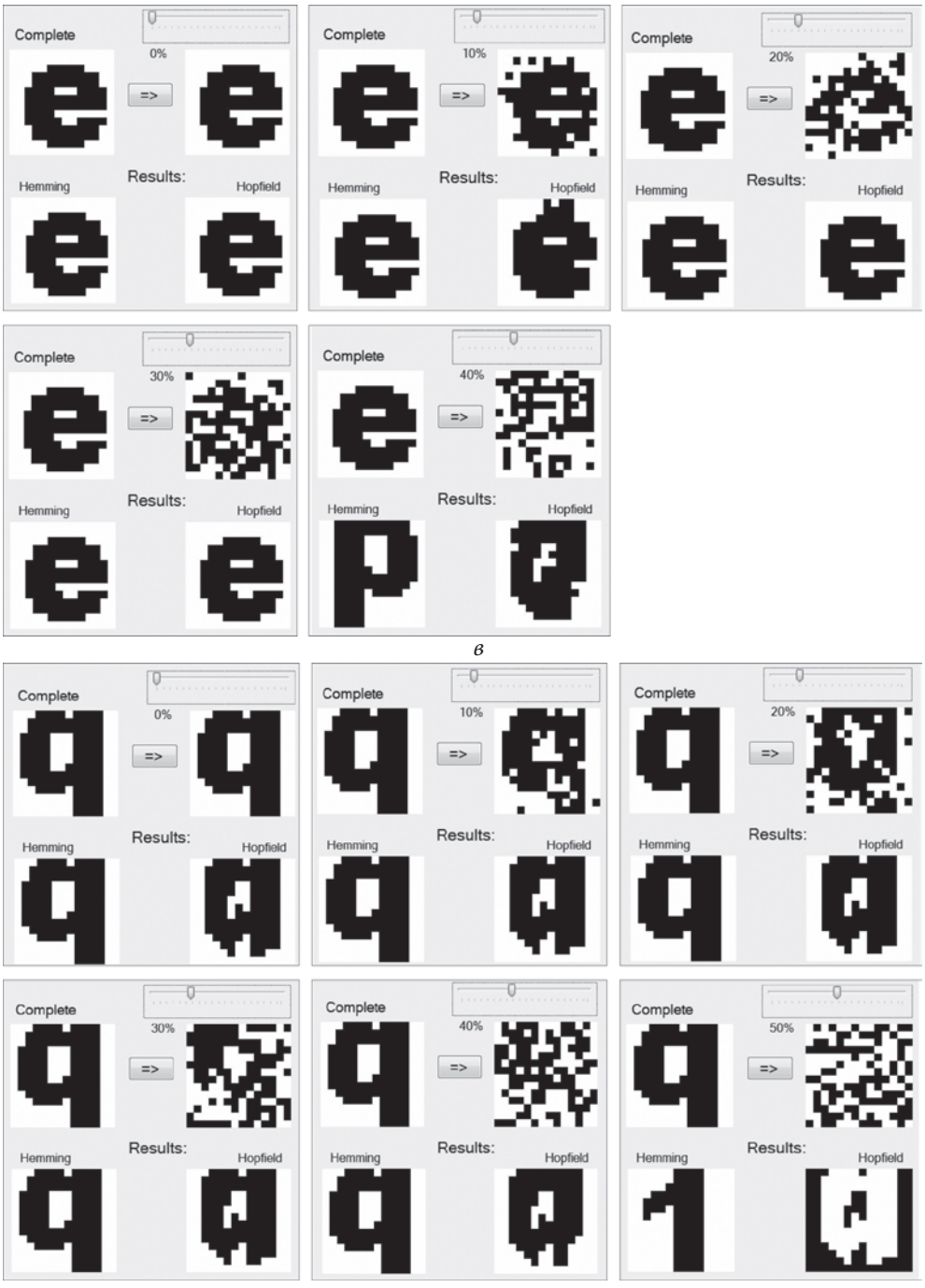


Рис. 2.6. *Продовження*: *в* — символа «e», *г* — символа «q»

д

Рис. 2.6. *Закінчення*: *д* — символа «р»

2.3. Самоорганізаційні (самоорганізовані) нейронні мережі. Алгоритми навчання Кохонена

Навчання на основі збігів. Закон навчання Хебба

У 1949 р. канадський психолог Дональд Олдинг Хебб (Donald Olding Hebb)[2] *опублікував книгу «Організація поведінки»* («Organizational Behaviour»), у якій запропонував правдоподібний механізм навчання на клітинному рівні в мозку [2.6, 2.12]. Основна ідея Д. Хебба полягає в тому, що якщо вхідний сигнал нейрона, який надходить через синаптичні зв'язки, зумовлює активацію нейрона, то ефективність цього входу, з точки зору його здатності стимулювати нейрон у майбутньому, має зростати.

Д. Хебб припустив, що ця зміна ефективності відбувається саме в синапсі, через який сигнал надходить на вхід нейрона. Подальші дослідження підтвердили припущення Хебба, хоча згодом було відкрито й інші механізми біологічного навчання на клітинному рівні. Як визнання заслуг ученого,

[2] Д. Хебб — видатний канадський фізіолог і нейропсихолог, чиї дослідження значно вплинули на розуміння функціонування біологічних нейронних мереж. Він також розробив перший алгоритм навчання штучних нейронних мереж.

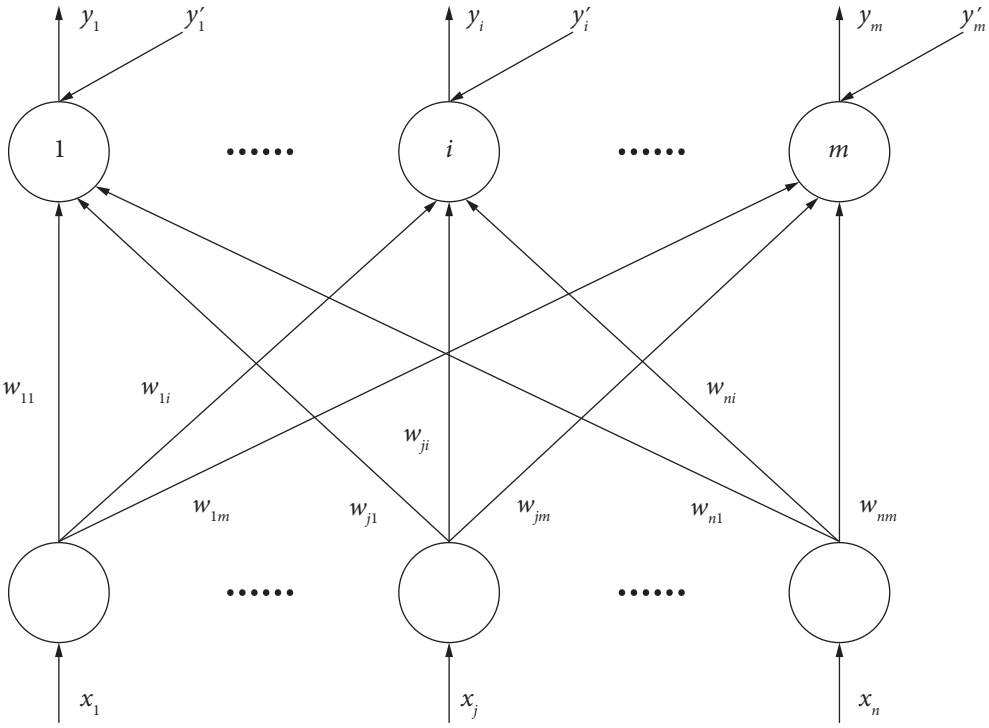


Рис. 2.7. Архітектура нейронної мережі, що складається з лінійних асоціаторів

цей механізм було названо *законом навчання Хебба* [2.12]. Закон навчання Хебба належить до класу законів змагального навчання і є основою для побудови багатьох моделей нейронних мереж.

Лінійні асоціативні елементи. На рис. 2.7 наведено архітектуру нейронної мережі, що складається з нейронів, які називають лінійними асоціаторами. Вхідний вектор у лінійному асоціаторі позначено як $X = \{x_i\}$, $i = \overline{1, n}$, який вибирається з простору R_n відповідно до деякого розподілу $\rho(X)$.

Вихідний вектор Y отримують з вхідного X за формулою

$$Y = WX, \tag{2.10}$$

де $W = ||w_{ij}||$ — матриця ваг n^*m; $W = (W_1, W_2, \ldots, W_m)$; W_j — стовпці матриці W; $W_j = (W_{1j}, W_{2j}, \ldots, W_{nj})^T$ — вектор ваг.

Позначимо через $Y = \{y_j\}$ бажаний вихід. Основна ідея лінійної асоціативної нейронної мережі полягає в тому, що мережа має навчатися на парах вхід—вихід:

$$(X_1, Y_1), (X_2, Y_2), \ldots, (X_L, Y_L).$$

Коли на вхід нейронної мережі подається вхідний сигнал X_k, то бажаний вихід Y' повинен дорівнювати Y_k. Якщо на вхід мережі подано вектор $Y_k + \varepsilon$ (де ε — досить мале), то вихід має дорівнювати $Y_k + \varepsilon$ (тобто на виході повинні отримати вектор, близький до Y_k).

Закон навчання Хебба записують як [2.12]:

$$w_{ij}^{\text{new}} = w_{ij}^{\text{old}} + y_{kj} x_{ki}, \tag{2.11}$$

де x_{ki} — i-та компонента вектора Y_k; y_{kj} — j-та компонента вектора Y_k.

У векторному вигляді вираз (2.11) перетворюється до виду

$$W^{\text{new}} = W^{\text{old}} + X_k Y_k^T. \tag{2.12}$$

Щоб реалізувати цей закон, у процесі навчання запроваджуються відповідні компоненти $Y_k = [y_{kj}]$, які показано стрілками на рис. 2.7.

Передбачається, що перед початком навчання всі $w_{ij} = 0$, тоді в результаті показу навчальної вибірки $(X_1, Y_1), \ldots, (X_L, Y_L)$ кінцевий стан матриці W задається так:

$$W = Y_1 X_1^T + Y_2 X_2^T + \ldots + Y_L X_L^T. \tag{2.13}$$

Рівняння (2.13) називають формулою суми зовнішнього множення для W. Ця назва походить з того факту, що — це зовнішнє множення. Переформулювання закону Хебба у вигляді суми зовнішнього множення дає змогу провести додаткові дослідження можливостей цього закону забезпечити асоціації пар векторів (X_k, Y_k).

Перший висновок полягає в тому, що якщо вектори $\{X_1, X_2, \ldots, X_L\}$ ортогональні і мають одиничну довжину, тобто ортонормовані, то маємо:

$$Y_k = WX_k. \tag{2.14}$$

Інакше кажучи, лінійна асоціативна нейронна мережа здійснюватиме бажане перетворення вхід—вихід.

Це наслідок властивості ортонормованості:

$$X_i^T X_j = \delta_{ij} = \begin{cases} 0, i \neq j; \\ 1, i = j. \end{cases} \tag{2.15}$$

Отже, маємо:

$$WX_k = \sum_{i=1}^{L} Y_i X_i^T X_k = Y_k \tag{2.16}$$

Однак проблема полягає в тому, що умова ортонормованості дуже жорстка. Далі ми обмежені вимогою, що $||x_j|| = 1$. Було б набагато доцільніше зняти це обмеження. Цього можна досягти, але не в лінійній асоціативній нейронній мережі. Тут, якщо вектори X_k не є ортонормованими, то з'являється помилка відтворення Y_k на виході:

$$WX_k = \sum_{i=1}^{L} Y_i X_i^T X_k = Y_k + \sum_{i \neq k}^{L} Y_i X_i^T X_k = Y_k + \eta. \tag{2.17}$$

Бажано досягти того, щоб η було мінімальним. Для того щоб забезпечити η_{min} або η = 0, потрібно перейти до нелінійної асоціативної мережі з нелінійними елементами.

Змагальне навчання

Змагальне навчання застосовують у завданнях самонавчання за відсутності класифікації вчителя.

Закони навчання, що належать до категорії змагальних, мають таку властивість: між окремими або всіма обробними елементами нейронної мережі виникає процес змагання. Елементи, які стають переможцями, здобувають право змінювати свої ваги. Натомість нейрони, які програють, або не змінюють своїх ваг, або змінюють їх за іншим правилом.

Змагальне навчання також відоме як *навчання Кохонена*. Його принцип суттєво відрізняється від навчання за Хеббом або алгоритму зворотного поширення помилки (*Back Propagation*), оскільки базується на принципі *самоорганізації*, а не на контрольованому навчанні з учителем.

Історія розвитку. Змагальне навчання має тривалу і складну історію. Наприкінці 1960-х — початку 1970-х рр. американський учений-когнітивіст і математик Стефан Гроссберг (Stephen Grossberg) запропонував низку змагальних схем навчання нейронної мережі. С. Гроссберг — професор когнітивних і нейронних систем Університету Бостона, а також професор математики, психології та біомедичної інженерії. Він є одним із засновників таких напрямів досліджень у сфері штучного інтелекту, як обчислювальна нейробіологія, когнітивна наука та штучні нейронні мережі. Зокрема, вчений розробив *теорію адаптивного резонансу нейронної архітектури* [2.1].

Ще одним дослідником, котрий зробив вагомий внесок у розвиток змагального навчання, був ван дер Мальсбург (van der Malsburg). Його закон навчання ґрунтувався на ідеї, що сума ваг, пов'язаних із входами для різних обробних нейронів одного вхідного елемента, має залишатися сталою під час навчання. Це означає, що якщо одна або кілька ваг збільшуються, інші мають пропорційно зменшуватися.

Розвиток ідеї Кохонена. На основі аналізу робіт Гроссберга, ван дер Мальсбурга та інших дослідників Теуво Калеві Кохонен, видатний фінський вчений у галузі штучних нейронних мереж і машинного навчання, дійшов висновку, що головною метою змагального навчання є створення набору векторів, які формують множину рівноймовірних представників із фіксованої функції щільності розподілу $P(X)$ вхідних векторів.

Попри те, що закони цього типу навчання були незалежно розроблені багатьма дослідниками, саме Кохонен першим звернув увагу на питання рівноймовірності. Його ідеї та праця здобули широке визнання завдяки пу-

блікації книги «Самоорганізація та асоціативна пам'ять» [2.15], а принцип змагального навчання став відомим як *закон навчання Кохонена*.

Закон навчання Кохонена

На рис. 2.8 наведено базову структуру шару Кохонена. Шар складається з N обробних елементів, кожен з яких отримує n вхідних сигналів $x_1, x_2, \ldots, x_n$ з нижчого шару, який є прямим передавачем сигналів. Входу x_i та зв'язку (i, j) припишемо вагу w_{ij}, $i = \overline{1, n}$; $j = \overline{1, N}$.

Кожний обробний елемент шару Кохонена підраховує свою вхідну інтенсивність I_j відповідно до формули [2.6, 2.15]:

$$I_j = D(W_j, X), \tag{2.18}$$

де $W_j = (W_{1j}, W_{2j}, \ldots, W_{nj})^T$ та $X = (x_1, x_2, \ldots, x_n)$; $D(W_j, X)$ — деяка міра (метрика) відстані між W_j і X.

Виділимо два унайзагальніші види функції $D(W_j, X)$:

1) евклідова відстань $d(W, X) = ||W - X||$;

2) сферична дугова відстань $\Delta(W, X) = 1 - W^T X = 1 - \cos\theta$, де $W_j = (W_{1j}, W_{2j}, \ldots, W_{nj})^T$ і $X = (x_1, x_2, \ldots, x_n)$; $W^T X$ — скалярний добуток, причому передбачається, що $||W|| = ||X|| = 1$; $\cos\theta$ — кут між векторами X та W.

Навчальні дані для шару Кохонена, ймовірно, складаються з послідовності вхідних векторів $\{X\}$, які випадково вилучаються з фіксованою щільністю розподілу ймовірностей $P(X)$. Щойно черговий з векторів X вводиться в мережу, обробні елементи Кохонена починають змагатися між собою, щоб знайти переможця, для якого досягається $\min_j d(W_j, X)$. Тоді для нейрона, що переміг, j^*, вихід встановлюється $z_j = 1$, а для всіх інших $z_j = 0$, $j \neq j^*$. У цей момент відбувається зміна ваг згідно із законом навчання Кохонена:

$$W_j^{\text{new}} = W_j^{\text{old}} + \alpha(X - W_j^{\text{old}})z_j, \tag{2.19}$$

де $0 < \alpha < 1$. Зазначений закон (2.19) можна переписати у вигляді

$$W_j^{\text{new}} = \begin{cases} (1 - \alpha_t)W_j^{\text{old}} + \alpha_t X_t \text{ для переможця } j = j^*; \\ W_j^{\text{old}} \ \forall_j \neq j^*, \end{cases} \tag{2.20}$$

де t — номер ітерації.

Очевидно, що за таким законом навчання ваговий вектор W_j рухається до вхідного вектора X. На початку навчання $\alpha_0 = 1$, а потім у процесі навчання α зменшується до $\alpha_{\min} = 0{,}1$. Зазвичай приймають $\alpha_t = e^{-\alpha_0 t}$.

Наведений алгоритм реалізує принцип «Переможець забирає все» (*Winner Takes All, WTA*) [2.14].

Слід зазначити подібність навчання за Кохоненом і статистичного процесу знаходження k-середніх. k-Середні для фіксованого набору векторів

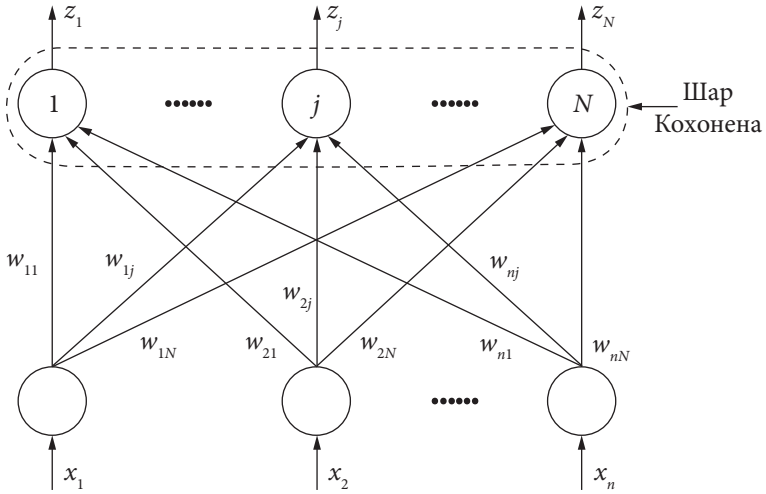


Рис. 2.8. Базова структура шару Кохонена

$\{X_1, X_2, \ldots, X_L\}$, вибраних випадково з деякої генеральної сукупності з фіксованою щільністю розподілу ймовірностей $P(x)$, складають множину k векторів $\{W_1, W_2, \ldots, W_k\}$, таких, що мінімізується функціонал

$$\min_{\{W_i\}} \sum_{i=1}^{L} D^2(X_i, W(X_i)), \tag{2.21}$$

де $W(X_i)$ — вектор W, найближчий до X_i.

Наостанок слід підкреслити, що навчальний закон Кохонена в загальному випадку не генерує множину рівноймовірних вагових векторів, тобто множину таких векторів, що X, обраний випадково, відповідно до щільності розподілу ймовірностей $P(x)$, матиме однакову ймовірність виявитися найближчим до кожного з вагових векторів W_j.

Модифіковані алгоритми змагального навчання

Як уже зазначалося, ми прагнемо домогтися того, щоб отримати вектори W_j, які самі були б приблизно рівноймовірними у сенсі найближчого сусідства відносно векторів X, які вибирають з R^n з деякою щільністю розподілу ймовірностей $P(x)$. Інакше кажучи, для довільного вектора X, вибраного з R^n з імовірністю $P(X)$, бажано, щоб імовірність того, що X виявиться найближчим до W_i, має приблизно дорівнювати $1/N$ для всіх $i = \overline{1, N}$.

Розроблено кілька підходів для вирішення проблем, що виникають під час реалізації базового закону навчання Кохонена [2.2, 2.15].

1. Перший підхід називається *Radial Sprouting* (радіальні щупальці). Він є найкращим для евклідової та аналогічних їй метрик (відстаней). Всі ва-

гові вектори $W_j = [w_{ij}]$ спочатку покладають $w_{ij}(0) = 0$. Всі вхідні вектори X спочатку множаться на деякий малий додатний скаляр β_k. Процес починається з β_1, близького до 0. Це забезпечує близькість вхідних векторів X до векторів W_j. У міру розвитку процесу β_k повільно збільшується, поки не досягне значення $\beta\infty = 1$. У процесі змагального навчання вагові вектори «виштовхуються» зі своїх початкових значень і рухаються за вхідними векторами. Ця схема працює досить добре, але зазвичай кілька вагових векторів відставатимуть від процесу змагального навчання і в результаті виявляться не задіяними, що уповільнює процес навчання.

2. Інший підхід, *Add noise* (додавання шуму), полягає в тому, щоб додати рівномірно розподілений шум ψ до векторів X, що полегшує ефект досягнення $\rho(X) > 0$ у всьому просторі. Рівень шуму спочатку вибирають досить великим, щоб вектори шуму були набагато більшими, ніж вектори даних X. Однак у міру розвитку процесу навчання рівень шуму $\psi(k)$ поступово знижується. Цей підхід працює правильно, але виявляється ще повільнішим ніж підхід *Radial Sprouting*. Тому підходи *Radial Sprouting* і *Add noise,* хоча й виконують презентацію складних законів з малою ймовірністю розподілу в певних зонах, але не вирішують головної проблеми змагального навчання — рівноймовірного позиціонування вхідних векторів щодо векторів W_j.

Загалом базовий закон навчання Кохонена призведе до надлишку в разі розміщення векторів W_j в ділянках з високою щільністю розподілу ймовірностей (РВ) $\rho(X)$ і до нестачі векторів W_j у зонах, де щільність $\rho(X)$ мала.

Головним недоліком розглянутих вище алгоритмів змагального навчання є те, що в них постійно перемагають одні й ті самі нейрони, тоді як інші завжди програють і не розвиваються, отже, основна мета змагального навчання не досягається. Щоб уникнути цього недоліку, *Duane Desieno* запропонував ввести «свідомість» (або пам'ять) у кожний нейронний елемент k, щоб вести моніторинг (контроль) за історією успішних результатів (перемог) кожного нейроелемента. Якщо обробний елемент Кохонена виграє змагання істотно частіше ніж $1/N$ разів (часу), то його «свідомість» виключає цей елемент зі змагання на деякий час, тим самим даючи можливість іншим елементам частіше вигравати і з перенасиченої зони переміщуватися в сусідні ненасичені ділянки. Такий підхід часто працює дуже добре і може згенеруватии хороший набір рівноймовірних вагових векторів.

Основна ідея механізму «свідомості» — відстеження частки часу f_j, протягом якого обробний елемент j виграє змагання. Цю величину можна обчислити локально кожним обробним елементом за формулою

$$f_j(t + 1) = f_j(t) + \beta(z_j - f_j(t)), \tag{2.22}$$

де константа β вибирається малим додатним числом (типове значення $\beta = 10^{-4} = 0{,}0001$). Відразу після закінчення змагання та визначення поточ-

ного значення z_j (0 або 1) обчислюється частка f_j, а затим визначається поточне значення «зміщення» b_j за формулою

$$b_j = \gamma(\frac{1}{N} - f_j), \tag{2.23}$$

де γ — додатна константа (порядку $\gamma \approx 10$).

Далі здійснюється корекція ваг. Однак, на відміну від звичайної ситуації, в якій ваги налаштовуються тільки для одного обробного елемента-переможця з $z_j = 1$, тут відбувається окреме змагання для знаходження обробного елемента, який має найменше значення величини:

$$D_{\text{new}}(W_j, X) = D(W_j, X) - b_j. \tag{2.24}$$

Елемент, що виграв, потім коригує свої ваги відповідно до звичайного закону навчання Кохонена.

Роль члена зміщення b_j полягає в такому. Для нейронів j, які часто перемагають, величина $f_j > 1/N$ і $b_j < 0$, тому для них величина $D(W_j, X) - b_j$ зростає порівняно з $D(W_j, X)$ і вони блокуються, тоді як для елементів, що виграють рідко, $b_j > 0$ і $D(W_j, X) - b_j$ зменшується, що підвищує їхні шанси перемогти.

Такий алгоритм реалізує механізм «свідомості» в роботі нейронної мережі із самоорганізацією і тому називається алгоритмом з пам'яттю (свідомістю) *CWTA* (*Conscience Winner Takes All*) [2.6, 2.15].

Розвиток алгоритму Кохонена

У 1982 р. Т. Кохонен запропонував ввести в базове правило змагального навчання інформацію про розміщення нейронів у вихідному шарі [2.6, 2.15]. Для цього нейрони вихідного шару впорядковуються, утворюючи одновимірну або двовимірну ґратку. Розміщення нейронів у таких ґратках маркується векторним індексом $i = (i_1, i_2)$. Таке впорядкування природно вводить відстань між нейронами $|i - j|$.

Модифіковане правило змагального навчання Кохонена враховує відстань нейронів від нейрона-переможця згідно з виразом [2.6, 2.8, 2.15]:

$$W_j(t+1) = W_j(t) + \alpha_t (X_{t+1} - W_j(t))\Lambda_t(d(i, j^*)), \tag{2.25}$$

де Λ — функція сусідства $\Lambda_t(d(i, j^*))$, що дорівнює 1 для нейрона-переможця з індексом j^* і поступово спадає в міру збільшення відстані d, наприклад, за законом

$$\Lambda_t(d) = e^{-\frac{d^2}{R_t^2}}. \tag{2.26}$$

Як темп навчання α_t, так і радіус взаємодії R_t поступово зменшуються в процесі навчання, так що на кінцевій стадії навчання ми повертаємось до базового закону адаптації ваги лише нейронів-переможців:

$$W_j(t) = \alpha_0 e^{-kt}, \; R_t = R_0 e^{-\beta t}.$$

Як бачимо, у цьому алгоритмі реалізується принцип «Переможець забирає максимум (доходу)», в зарубіжній літературі він має назву *Winner Takes MOST, WTM* [2.6, 2.8].

На відміну від газоподібної динаміки в разі індивідуального підстроювання нейронів навчання за Кохоненом нагадує натягування еластичної сітки прототипів на масив даних з навчальної вибірки. У процесі навчання еластичність сітки поступово зменшується. В результаті ми не тільки отримуємо квантування входів, а й упорядковуємо вхідну інформацію у вигляді одно- або двовимірної топографічної карти Кохонена. На цій сітці кожний багатовимірний вектор має свою координату, причому чим ближчі координати двох векторів на карті, тим ближчі вони і у вихідному просторі.

Алгоритм нейронного газу

Прискорення збіжності модифікованого алгоритму Кохонена WTM та кращої самоорганізації мережі можна досягти застосуванням методу, запропонованого М. Мартінесом, С. Берковичем та К. Шультеном [2.8] і названого авторами «алгоритмом нейронного газу», внаслідок подібності його динаміки руху молекул газу. У цьому алгоритмі на кожній ітерації всі нейрони сортуються залежно від їх відстані до вхідного вектора. Після сортування нейрони розміщуються у послідовності, що відповідає збільшенню їх віддаленості:

$$d_0 < d_1 < d_2 < ... < d_{n-1}, \tag{2.27}$$

де $d_i = ||x - W_{m(i)}||$ позначає віддаленість від вектора X i-го нейрона, який займає в результаті сортування m-ту позицію в послідовності, що починається з нейрона-переможця, якому ставиться у відповідність віддаленість d_0.

Значення функції сусідства для i-го нейрона $\Lambda(i, x)$ визначається за формулою

$$\Lambda(i, x) = \exp\left\{-\frac{m(i)}{\sigma(t)}\right\}, \tag{2.28}$$

де $m(i)$ визначає черговість, отриману в результаті сортування ($m(i) = 0, 1, 2, \ldots, n-1$ і $\sigma(t)$ — параметр, аналогічний рівню сусідства R в алгоритмі Кохонена WTM, що зменшується з часом t).

За $\sigma(t) = 0$ адаптації зазнає лише нейрон-переможець і алгоритм перетворюється на звичайний (базовий) алгоритм Кохонена, а за $\sigma(t) \neq 0$ адаптації підлягають ваги багатьох нейронів-сусідів, причому рівень зміни ваг залежить від величини $\Lambda(i, x)$.

Для досягнення задовільного результату самоорганізації процес навчання має починатися з досить великого значення σ, але з часом його величина зменшується до нуля. Функція $\sigma(t)$ може бути лінійною або показниковою.

У роботі [2.8] було запропоновано змінювати значення $\sigma(t)$ відповідно до виразу

$$\sigma(t) = \sigma_{max}\left(\frac{\sigma_{min}}{\sigma_{max}}\right)^{\frac{t}{T_{max}}}, \quad (2.29)$$

де $\sigma(t)$ — значення σ на ітерації t; σ_{min} і σ_{max} — прийняті мінімальне та максимальне значення. Величина T_{max} визначає задану максимальну кількість ітерацій.

Коефіцієнт навчання і-го нейрона $\alpha_i(t)$ може також змінюватися як лінійно, так і показниково, причому його показникова залежність визначається виразом:

$$\alpha_i(t) = \alpha_i(0)\left(\frac{\alpha_{min}}{\alpha_i(0)}\right)^{\frac{t}{T_{max}}}, \quad (2.30)$$

в якому $\alpha_i(0)$ — початкове значення α, а α_{min} — апріорно задане мінімальне значення, що відповідає $t = T_{max}$.

На практиці найкращі результати самоорганізації досягаються в разі лінійної зміни α_t. Для скорочення обсягу обчислень у реалізації алгоритму нейронного газу можна використовувати спрощення, яке полягає в тому, що адаптації ваги відбуваються тільки у перших k нейронів-сусідів у впорядкованій послідовності $d_0 < d_1 < d_2 < \ldots < d_k$.

Алгоритм нейронного газу разом з алгоритмом з пам'яттю Кохонена (*CWTA*), що враховує частку перемог кожного нейрона (f_j), є одними з найбільш ефективних засобів самоорганізації нейронів у мережі Кохонена.

Порівняльний аналіз алгоритмів самоорганізації

Розглянуті вище алгоритми порівнювали під час вирішення завдань відновлення двовимірних навчальних даних складної структури, зображених на рис. 2.9 [2.8]. Для відновлення даних використовували дві множини нейронів, що містять 40 і 200 елементів, які за упорядкування позицій нейронів будуть відображати розподіл даних. Вони мають розміщуватися в ділянках максимальної концентрації даних.

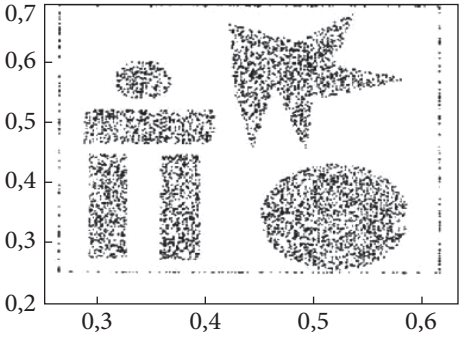


Рис. 2.9. Двовимірні навчальні дані складної структури

Результати самоорганізації 40 нейронів за використання трьох алгоритмів — алгоритму з пам'яттю (*CWTA*), нейронного газу та базового алгоритму Кохонена наведено на рис. 2.10. Для порівняння аналогічні відображення мережею Кохонена, що складається з 200 нейронів, наведено на рис. 2.11.

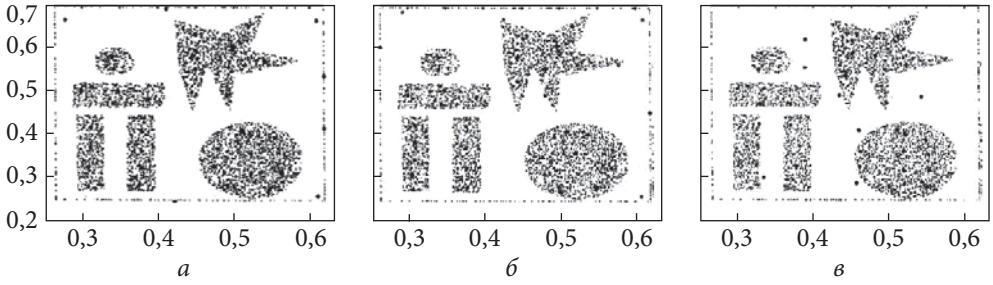


Рис. 2.10. Результати самоорганізації 40 нейронів за використання трьох алгоритмів: *а* — алгоритм з пам'яттю (*CWTA*); *б* — алгоритм нейронного газу; *в* — базовий алгоритм Кохонена

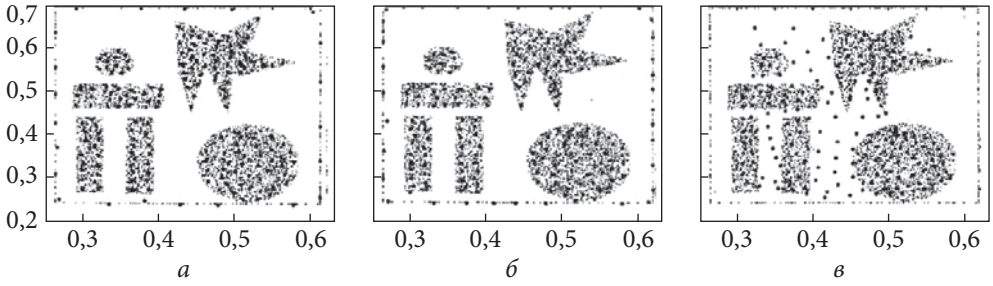


Рис. 2.11. Відображення мережею Кохонена, що складається з 200 нейронів за використання трьох алгоритмів: *а* — алгоритм з пам'яттю (*CWTA*); *б* — алгоритм нейронного газу; *в* — базовий алгоритм Кохонена

Як випливає з наведених результатів, незалежно від числа нейронів найкращі результати самоорганізації отримано з використанням алгоритмів з пам'яттю та нейронного газу.

Для кількісного порівняння результатів можна використати критерій «похибки квантування»:

$$E_q = \sum_{i=1}^{n} ||x_i - w_j^*||, \tag{2.31}$$

де w_j^* — вага нейрона-переможця за подання вектора x_i.

За 200 нейронів було отримано значення:

E_q = 0,007139 для *CWTA*; E_q = 0,007050 для алгоритму нейронного газу; E_q = 0,02539 для алгоритму Кохонена.

2.4. Застосування нейронних мереж Кохонена

Нейронні мережі із самоорганізацією застосовують у двох основних напрямах [2.6, 2.15]:

1) для автоматичної класифікації об'єктів;

2) для візуального відображення властивостей багатовимірного простору (зображення багатовимірних векторів ознак).

У першому випадку вирішується задача автоматичного розбиття множини об'єктів, які зображують багатовимірними векторами в деякому просторі ознак за подібністю-відмінністю значень ознак. Цю задачу іноді називають задачею кластерного аналізу, її буде докладно розглянуто у розділі 7.

У другому випадку вирішується завдання візуалізації властивостей багатовимірних векторів ознак на двовимірній площині. З цією метою використовують так звані самоорганізаційні карти ознак (*Self-organizing Feature Maps*, або *SOM*). Для цього в багатовимірному просторі натягують просторові ґратки, у вузлах яких розміщені обробні нейрони (шар нейронів Кохонена). Далі визначаються найближчі точки до кожного з цих нейронів. Вони визначають басейн притягання нейрона. Обчислюється середнє значення за кожною окремою ознакою нейронів з басейну притягання $X_{i\,\text{ср}}$.

Далі вузли ґратки відображаються на площині у вигляді квадратів або шестикутників; відповідне значення ознаки для кожного нейрона забарвлюють у певний колір: від синього (мінімальне значення ознаки) до червоного (максимальне значення). В результаті отримуємо карту ознак з самоорганізацією, аналогічну географічній карті. Такі карти будують за всіма ознаками і отримують своєрідний атлас.

Самоорганізаційна карта ознак (СОК) є методом проєктування N-розмірного вхідного простору в дискретний вихідний простір, який виробляє ефективну компресію вхідного простору в множину кодованих (вагових) векторів. Вихідний простір зазвичай являє собою двовимірну ґратку. СОК використовує ґратку для апроксимації функції щільності ймовірності вхідного простору, при цьому зберігаючи його структуру, тобто якщо два вектори близькі один до одного у вхідному просторі, то вони мають бути близькими і на карті. У ході процесу самоорганізації СОК здійснює ефективну кластеризацію вхідних векторів, зберігаючи структуру вихідного простору.

Стохастичний алгоритм навчання. Навчання СОК ґрунтується на стратегії змагального навчання. Розглянемо N-вимірні вхідні вектори X_p, де індекс p означає один навчальний патерн. Першим кроком процесу навчання є визначення структури карти, зазвичай двовимірної ґратки. Карта, як правило, квадратна, але може бути прямокутною. Число елементів (нейронів вихідного шару) на карті менше, ніж кількість навчальних патернів (зразків). В ідеалі число нейронів має дорівнювати числу незалежних навчальних патернів. Кожен нейрон на карті пов'язаний з N-вимірним ваговим вектором, який утворює центр одного кластера. Великі кластерні угруповання утворюються в результаті групування разом «подібних» сусідніх нейронів.

Структуру СОК наведено на рис. 2.12.

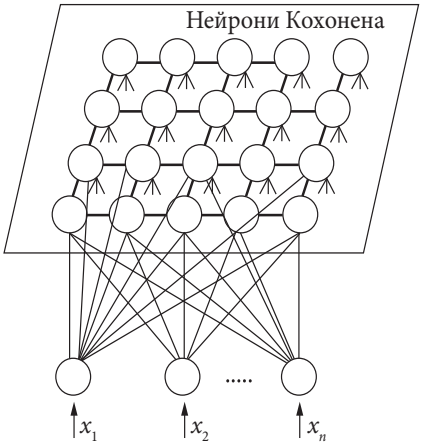


Рис. 2.12. Структура самоорганізаційної карти ознак

Ініціалізацію вагових векторів можна виконувати різними способами.

1. Кожній вазі $w_{kj} = \{w_{kj_1}, w_{kj_2}, \ldots, w_{kj_N}\}$, де k — число рядків, а j — число стовпців, приписують випадкові значення. Початкові значення обмежені діапазоном відповідного вхідного параметра (змінної). Хоча випадкову ініціалізацію вагових векторів просто реалізувати, такий спосіб ініціалізації вводить у СОК значну варіацію компонент, що збільшує час навчання.

2. Векторам ваг приписують випадково обрані вхідні патерни, тобто $w_{kj} = X_p$, де p — індекс патерну.

Такий підхід може призвести до передчасної збіжності доти, доки ваги не будуть збурені малими випадковими значеннями.

3. Знайти основні компоненти векторів вхідного простору та ініціалізувати вагові вектори цими значеннями.

4. Інша технологія ініціалізації ваг полягає у визначенні гіперкуба досить великого розміру, що покриває всі навчальні патерни. Алгоритм починається зі знаходження чотирьох екстремальних точок за допомогою визначення чотирьох екстремальних навчальних патернів. Спочатку знаходять два патерни з найбільшою відстанню між ними в евклідовій метриці. Третій патерн розміщується у найвіддаленішій точці від цих двох патернів, а четвертий патерн — з найбільшою відстанню від цих трьох патернів за Евклідом. Ці чотири патерни утворюють кути ґратки на карті (СОК: (1, 1); (1, J); (K, 1); (K, J). Вагові вектори інших нейронів визначають методом інтерполяції чотирьох обраних патернів так, як наведено нижче.

Ваги граничних нейронів ініціалізуються за формулами [2.10, 2.15]:

$$
\begin{aligned}
w_{1j} &= \frac{w_{1J} - w_{11}}{J-1}(j-1) + w_{11};\\
w_{Kj} &= \frac{w_{KJ} - w_{K1}}{J-1}(j-1) + w_{K1};\\
w_{k1} &= \frac{w_{K1} - w_{11}}{K-1}(k-1) + w_{11};\\
w_{kJ} &= \frac{w_{KJ} - w_{1J1}}{K-1}(k-1) + w_{1J}
\end{aligned}
\qquad (2.32)
$$

для всіх $j = 2, 3, \ldots, J-1$ та $k = 2, 3, \ldots, K-1$.

Інші вагові вектори ініціалізуються як

$$w_{kj} = \frac{w_{kJ} - w_{k1}}{J - 1}(j - 1) + w_{k1}. \tag{2.33}$$

Стандартний алгоритм навчання для СОК є стохастичним, в якому вагові вектори адаптуються після показу кожного патерну мережі. Для кожного нейрона пов'язаний з ним кодовий вектор (вага) адаптується як

$$w_{kj}(t + 1) = w_{kj}(t) + h_{mn,\,kj}(t)[X_p - w_{kj}(t)], \tag{2.34}$$

де *m, n* — індекс рядка та стовпця нейрона-переможця.

Нейрон-переможець визначається, як завжди, обчисленням евклідової відстані від кожного вагового вектора до вхідного вектора та вибором нейрона, найближчого до вхідного вектора, тобто:

$$||w_{mn} - X_p||^2 = \min_{(k,\,j)}\{||w_{kj} - X_p||^2\}. \tag{2.35}$$

Функція $h_{mn,\,kj}(t)$ у рівнянні (2.34) сприймається як функція сусідства. Отже, тільки ті нейрони, які перебувають в околиці (по сусідству) нейрона-переможця *mn*, змінюють свої ваги. Для забезпечення збіжності необхідно, щоб $h_{mn,\,kj}(t) \to 0$ при $t \to \infty$.

Функція сусідства зазвичай є функцією відстані між координатами нейронів, які представлені на карті, тобто:

$$h_{mn,\,kj}(t) = h(||c_{mn} - c_{kj}||^2, t), \tag{2.36}$$

де $c_{mn}, c_{kj} \in R^2$, причому зі збільшенням відстані $||c_{mn} - c_{ki}||^2$, $h_{mn,\,kj}(t) \to 0$. Сусідство можна визначити за квадратом або шестикутником. Однак найчастіше використовують гладке ядро Гаусса згідно з виразом

$$h_{mn,\,kj}(t) = \alpha(t)\exp\left(\frac{-||c_{mn} - c_{kj}||^2}{2\sigma^2(t)}\right). \tag{2.37}$$

де $\alpha(t)$ — швидкість навчання; $\sigma^2(t)$ — ширина ядра. Обидві функції є монотонно спадними зі збільшенням часу *t*.

Побудова СОК у пакетному режимі. Стохастичний алгоритм навчання СОК є надто повільним через необхідність адаптації ваг усіх нейронів після кожного показу патерну. Було розроблено версію алгоритму навчання СОК у пакетному режимі. Перший пакетний алгоритм навчання СОК, розроблений Кохоненом, складається з таких кроків:

1. Ініціалізувати вагові вектори через призначення їм перших *KJ* навчальних патернів, де *KJ* — загальна кількість патернів на карті.

Поки не виконається умова зупинки do,

для кожного нейрона kj do

скласти список всіх патернів X_p, що виявляються найближчими до вагового вектора певного нейрона;

end.

2. Для кожного вагового вектора w_{kj} обчислити нове значення вагового вектора як середнє відповідного списку патернів.

Було розроблено також прискорену версію пакетного алгоритму навчання СОК. Одна з проєктних проблем, що виникають під час побудови СОК, — визначення розмірів карти. Занадто багато нейронів може зумовити перенасичення, коли кожний навчальний патерн буде приписаний різним нейронам. Або альтернативно, кінцева СОК може успішно сформувати хороші кластери з подібних патернів, але багато нейронів виявляться з нульовою або близькою до нульової частотою використання, де частота нейрона означає чисельність патернів, для яких нейрон став переможцем. Надто мала кількість таких нейронів, з іншого боку, призводить до появи кластерів з великою внутрішньокластерною дисперсією.

Прискорений пакетний алгоритм побудови СОК:

1. Ініціалізувати вагові вектори w_{kj}, використовуючи будь-який спосіб ініціалізації.

Поки не виконається умова зупинки do,

для кожного нейрона kj do

обчислити середнє всіх патернів, для яких певний нейрон був переможцем;

позначити середнє $\overline{w_{kj}}$;

end.

2. Адаптувати значення ваг для кожного нейрона, використовуючи вираз:

$$w_{kj} = \frac{\sum_n \sum_m N_{nm} h_{nm,\,kj} \overline{w_{nm}}}{\sum_n \sum_m N_{nm} h_{nm,\,kj}}, \tag{2.38}$$

де індекси *m, n* підсумовуються за всіма номерами рядків та стовпців; N_{nm} —число патернів, для яких нейрон *nm* виявився переможцем; $h_{mn,\,kj}$ — це функція сусідства, яка вказує, чи потрапляє нейрон *nm* в ділянку сусідства *kj* і якою мірою;

end.

Метод знаходження близької до оптимальної структури СОК полягає в тому, щоб почати з малої структури та збільшувати розміри карти, коли потрібно збільшити число нейронів. Зауважимо, що процес розвитку карти відбувається одночасно з процесом навчання. Розглянемо один із алгоритмів розвитку структури СОК для прямокутної карти.

Алгоритм розвитку структури СОК [2.16]:

Ініціалізувати вектори ваг для невеликої за розмірами СОК,

доки не виконається умова зупинки, do;

поки не спрацює умова зростання карти, do,

навчати СОК для показів t патернів, використовуючи будь-який метод

навчання СОК;

end.

Якщо виконується умова зростання карти, то:

1) знайти нейрон *kj* з найбільшою похибкою квантування (внутрішньокластерною дисперсією);

2) знайти найдальшого безпосереднього сусіда *mn* за рядками карти;

3) знайти найвіддаленіший нейрон за стовпчиками карти;

4) вставити стовпець між нейронами *kj* і *rs* та рядок між нейронами *kj* і *mn* (цей крок зберігає прямокутну структуру карти);

5) для кожного нейрона (*a*, *b*) в новому стовпці ініціалізувати відповідний вектор w_{ab}, використовуючи

$$w_{ab} = \gamma(w_{a,\,b-1} + w_{a,\,b+1}), \tag{2.39}$$

і для кожного нейрона у новому рядку обчислити

$$w_{ab} = \gamma(w_{a-1,\,b} + w_{a+1,\,b}), \tag{2.40}$$

де $\gamma \in (0, 1)$; *end;*

6) налаштовувати ваги остаточної структури СОК, використовуючи додаткові ітерації навчання, доки не буде досягнуто подібності.

Збільшення розмірів карти необхідно зупинити, коли буде задоволено один із таких критеріїв:

- досягнуто максимального розміру карти;
- найбільша похибка квантування для нейрона стане меншою від певного заданого користувачем порога ε;
- похибка квантування карти досягла заданого значення.

Деякі аспекти цього алгоритму потребують пояснення. Є константи ε, γ і максимальний розмір СОК, а також різні умови зупинки. Для параметра γ добрим вибором є $\gamma = 0{,}5$. Ідея кроку інтерполяції полягає у призначенні вагового вектора новому нейрону для того, щоб він захопив патерни від нейрона *kj* з найбільшою похибкою квантування, щоб зменшити помилку цього нейрона. Значення $\gamma < 0{,}5$ розмістить нейрон (*a*, *b*) ближче до *kj*, що, можливо, призведе до того, що більше патернів буде захоплено ним у нейрона *kj*, значення $\gamma > 0{,}5$ зумовить зворотний ефект.

Поріг похибки квантування ε важливий для того, щоб забезпечити достатній розмір карти СОК. Мале значення ε призводить до надто великого розміру СОК, тоді як завелике значення ε може спричинити збільшення часу навчання, щоб досягти достатньо великого розміру структури.

Верхню межу розмірів карти легко визначити, вона дорівнює просто числу навчальних патернів p_T. Однак це, звісно, небажано. Максимальний розмір карти можна вибрати як βP_T, де $\beta \in (0, 1)$. Оптимальне значення β залежить від розв'язуваної задачі, і потрібно вжити заходів, щоб

забезпечити не дуже мале значення β, якщо не передбачено збільшення розмірів СОК.

Процес навчання СОК надто повільний унаслідок великої кількості ітерацій коригування ваг. Для зменшення обчислювальної складності та прискорення збіжності навчання запропоновано кілька механізмів. Одним з них є пакетний режим конструювання СОК. До двох інших механізмів входить налаштування функцій сусідства та швидкості навчання.

Якщо використовують гауссову функцію сусідства, всі нейрони потраплять у ділянку сусідства нейрона-переможця, але різною мірою. Тому, вводячи певний поріг θ, можна обмежити кількість нейронів, які потраплятимуть у цю ділянку, і цим скоротити обчислювальні витрати на корекцію їхніх ваг. Крім того, ширину функції сусідства можна динамічно змінювати впродовж процесу навчання. Наприклад, можна вибрати цю функцію як

$$\sigma(t) = \sigma(0)e^{-\frac{t}{\tau_1}}, \tag{2.41}$$

де $\tau_1 > 0$ — деяка константа; σ(0) — початкова досить велика варіація.

Аналогічно можна використовувати швидкість навчання α(*t*), що зменшується зі збільшенням часу, за формулою

$$\alpha(t) = \alpha(0)e^{-\frac{t}{\tau_1}}, \tag{2.42}$$

де $\tau_2 > 0$ — деяка константа; α(0) — початкова велика швидкість навчання.

Кластеризація та візуалізація. Застосування СОК. Дія процесу навчання СОК полягає у кластеризації (групуванні) подібних патернів, зберігаючи при цьому топологію вхідного простору. Після навчання виходить множина навчених ваг без явних меж між кластерами. Для знаходження цих меж потрібен додатковий крок.

Один із способів визначити й візуалізувати межі між кластерами — обчислити уніфіковану матрицю відстаней (U-матрицю [2.1, 2.2]), яка передбачає геометричну апроксимацію розподілу вагових векторів на карті. U-Матриця для кожного нейрона відображує відстані до вагових векторів сусідніх нейронів. Великі значення матриці відстаней U вказують на розташування меж між кластерами.

Як приклад розглянемо завдання кластеризації ірисів. Його постановку та опис наведено у розд. 4. Наприклад, на рис. 2.13, *а* зображено U-матрицю для завдання кластеризації ірисів з використанням масштабування карти СОК у відтінках сірого. Межі між кластерами позначено темнішим кольором.

Для цього завдання візуалізовано кластери на повній карті (рис. 2.13, *б*). Межі поміж них зазвичай можна знайти одним з методів кластеризації, наприклад (див. розд. 7). Кластеризація методом Джоелома Х. Уорда (Joe H. Ward Jr.) використовує підхід «знизу вгору», в якому кожний нейрон спочат-

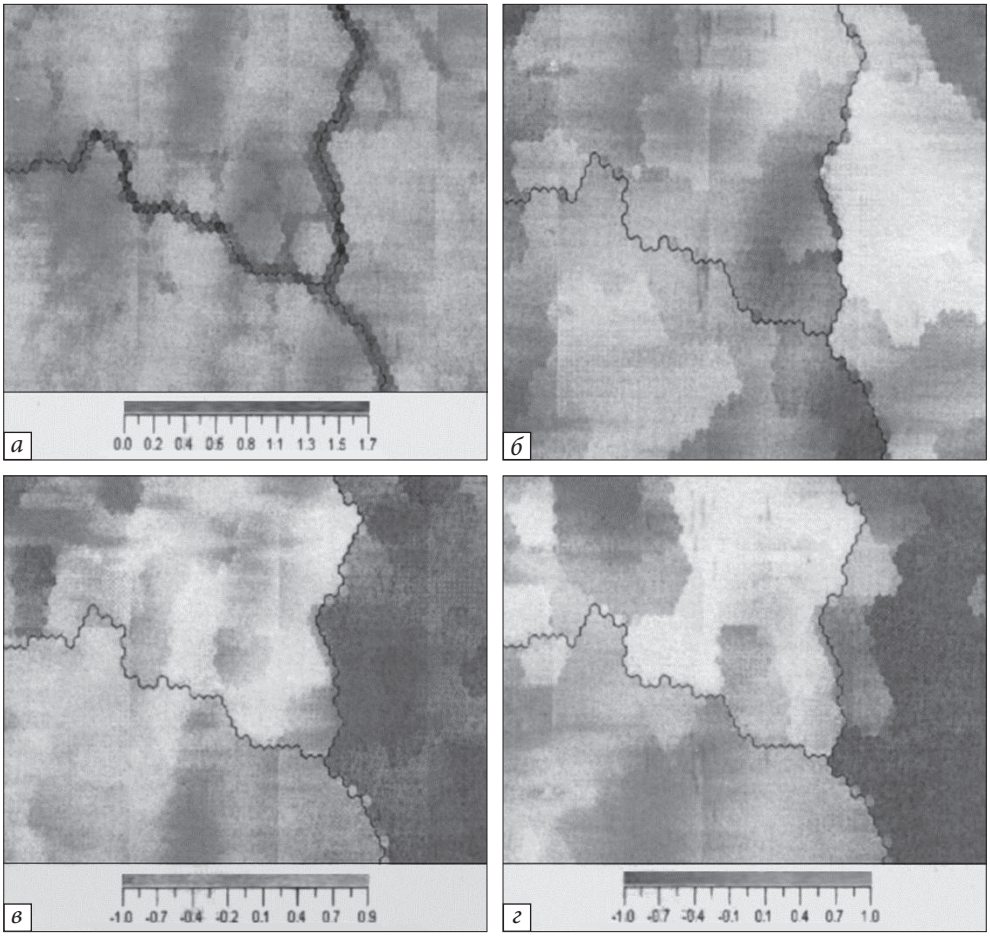


Рис. 2.13. U-Матриця для завдання кластеризації ірисів з використанням масштабування карти СОК у відтінках сірого

ку утворює свій власний кластер. На наступних ітераціях два найближчих кластери зливаються в один доти, доки не буде сконструйовано оптимальне або певне число кластерів. Остаточним результатом кластеризації є набір кластерів з мінімальною внутрішньокластерною дисперсією і значною міжкластерною дисперсією.

Для визначення кластерів, які необхідно об'єднати, використовують метрику (відстаней) за Уордом (див. розд. 7). Метрику відстані визначають так:

$$d_{rs} = \frac{n_r n_s}{n_r + n_s} ||w_r - w_s||^2, \qquad (2.43)$$

де s та r — індекси кластерів; n_r і n_s — число патернів у відповідних кластерах; w_s і w_r — вектори центрів ваг цих кластерів.

Два кластери — *s* і *r* об'єднуються, якщо їхня метрика d_{rs} є найменшою. Для нового сформованого кластера *q* його вагу визначають так:

$$w_q = \frac{1}{n_r + n_s}(n_r w_r + n_s w_s); \tag{2.44}$$

$$n_q = n_r + n_s. \tag{2.45}$$

Зауважимо, що за збереження топологічної структури два кластери можуть бути об'єднані лише в тому разі, якщо вони є суміжними.

Основна перевага СОК полягає у легкій візуалізації та інтерпретації кластерів, сформованих на карті. Крім візуалізації повної карти, наведеної на рис. 2.13, б, можуть бути візуалізовані окремі компоненти векторів, тобто для кожного вхідного параметра-ознаки можна побудувати окрему карту для візуалізації (відображення) розподілу значень цієї ознаки в просторі, використовуючи відповідну гаму кольорів. Таку карту і площини ознак можна використати для дослідження та аналізу даних. Наприклад, зазначений регіон на візуалізованій карті можна спроєктувати на площину ознак, щоб знайти розподіл значень відповідних вхідних параметрів (ознак) для цього регіону. На рис. 2.13, *в* та 2.13, *г* наведено СОК за третьою й четвертою ознаками для завдання кластеризації ірисів.

Навчену СОК можна також використати як класифікатор [2.6, 2.16], оскільки інформація щодо кластерів недоступна протягом процесу навчання. Кластери, сформовані на карті (СОК), потрібно досліджувати вручну і присвоїти їм мітки класів. Далі вводять вектор вхідних даних на карту і визначають «нейрона-переможця». Відповідну мітку кластера, куди потрапляє введений вхідний вектор, використовують як ім'я (номер) класу.

У разі використання в режимі відновлення неповної інформації СОК можуть бути використані для інтерполяції пропущених значень у патерні. Під час подання на вхід такого патерну визначається нейрон-переможець, ігноруючи входи з пропущеними значеннями. Далі визначається пропущене значення або через заміну пропущених значень відповідними значеннями ознак нейрона-переможця, або інтерполяцією значень сусідніх нейронів.

Описані топографічні карти дають наочне уявлення про структуру даних у багатовимірному вхідному просторі, геометрію якого ми можемо уявити інакше. Візуалізація багатовимірної інформації є головним застосуванням карт Кохонена.

Зауважимо, що, згідно із загальним життєвим принципом «безкоштовних обідів немає», топографічні карти зберігають відношення близькості лише локально, отже, близькі на карті зони близькі й у вихідному просторі, але не навпаки (рис. 2.14). Загалом немає відображення, що знижує розмірність і зберігає у своєму представленні близькості глобально.

Порушення неперервності відображення ілюструє рис. 2.14: близькі точки на площині відображаються на протилежних кінцях карти.

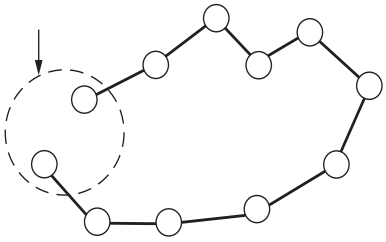


Рис. 2.14. Приклад порушення неперервності відображення за використання СОК

Ефективним інструментом візуалізації є кольорове кодування топографічних карт, подібно до методів, які використовують у географічних картах. Кожна ознака генерує своє забарвлення карти відповідно до середнього значення цієї ознаки для даних, що потрапили в певні кластери [2.16].

Об'єднавши карти всіх цікавих ознак, можна створити топографічний атлас, який дає інтегральне уявлення про структуру багатовимірних даних. Самонавчальні мережі Кохонена широко застосовують для попереднього оброблення даних у завданнях розпізнавання образів у просторах великої розмірності. Для забезпечення ефективності цього підходу спершу потрібно стиснути вхідну інформацію за допомогою:

- зменшення розмірності, виділивши найсуттєвіші ознаки;
- квантування даних.

Самоорганізаційні карти Кохонена (СОК) використовують для вирішення широкого кола практичних завдань, у тому числі аналізу зображень, розпізнавання мови, аналізу музичних образів, оброблення сигналів, робототехніки, телекомунікацій, проєктування електронних схем, виявлення прихованих знань та аналізу часових рядів.

Висновки

Розділ 2 присвячений рекурентним нейронним мережам (РНМ) та нейронним мережам із самоорганізацією Кохонена.

Рекурентні нейронні мережі. Розглянуто мережі Хопфілда, введено поняття атрактора та його басейну притягання. Докладно описано алгоритм асинхронної корекції роботи РНМ Хопфілда і визначено умови вибору атракторів. Наведено приклад застосування мережі Хопфілда для розпізнавання літер за різних рівнів шуму. Визначено властивості та обмеження мереж Хопфілда, показано їхню роль як наочної моделі асоціативної пам'яті, яка, однак, не набула широкого практичного використання через істотні обмеження.

Альтернативою РНМ Хопфілда є *мережі Хеммінга*, які позбавлені основних недоліків РНМ Хопфілда. Описано структуру та алгоритм навчання мереж Хеммінга. Проведено порівняльний аналіз обох типів мереж у завданні класифікації літер латинського алфавіту за умов зростання рівня шуму, що демонструє переваги мереж Хеммінга.

Самоорганізаційні мережі Кохонена. Розглянуто завдання змагального навчання та їх особливості. Сформульовано головну мету змагального навчання за Кохоненом, наведено архітектуру мережі Кохонена і базовий алгоритм змагального навчання *WTA (Winner Takes All).* Проаналізовано властивості і недоліки цього алгоритму, зокрема його неспроможність повністю досягати цілей змагального навчання. Для подолання цих недоліків запропоновано модифікацію — алгоритм *CWTA (Conscience Winner Takes All* з пам'яттю).

Також розглянуто модифікований алгоритм самоорганізації Кохонена з функцією сусідства *WTM (Winner Takes Most)* та його властивості. Детально описано алгоритм нейронного газу, який є вдосконаленням WTM.

Практичні застосування. Мережі самоорганізації застосовують для автоматичної класифікації об'єктів за ознаками та візуалізації багатовимірних об'єктів. Розглянуто різні алгоритми побудови СОК та проаналізовано їхні властивості. Наведено приклади використання цих алгоритмів у задачах відновлення двовимірного розподілу у вибірці даних.

СПИСОК ЛІТЕРАТУРИ

2.1. Engelbrecht A. Computational Intelligence: An Introduction. Wiley, 2007. 628 p.

2.2. Zgurovsky M. Z., Zaychenko Yu. P. The Fundamentals of Computational Intelligence: System Approach. Springer, 2016. 375 p.

2.3. Russell S., Norvig P. Artificial Intelligence: A Modern Approach. 4th edition. Pearson, 2021. 1152 p.

2.4. Haykin S. Neural Networks: A Comprehensive Foundation. Prentice Hall, 1998. 842 p.

2.5. Bishop C.M. Pattern Recognition and Machine Learning. Springer, 2006. 738 p.

2.6. Schalkoff R.J. Intelligent Systems: Principles, Paradigms, and Pragmatics. Jones & Bartlett, 2011. 608 p.

2.7. Luger Y.F. Artificial Intelligence: Structures and Strategies for Complex Problem Solving. Pearson, 2005. 896 p.

2.8. Osowski S. Neural Networks for Information Processing. Springer, 1994. 300 p.

2.9. Rutkowska D., Pilinski M., Rutkowski L. Neural Networks, Genetic Algorithms, and Fuzzy Systems. Springer, 1997. 368 p.

2.10. Haykin S. Neural Networks and Learning Machines. 3rd edition. Pearson, 2008. 936 p.

2.11. Bezdek J.C. On the relationship between neural networks, pattern recognition and intelligence. *Int. J. Approximate Reasoning.* 1992. Vol. 6. P. 85—107.

2.12. Hebb D.O. The organization of Behavior: A Neuropsychological Theory. New York: John Wiley and Sons, 1949.

2.13. Hopfield J.J. Neurons, dynamics and computation. *Phys. Today*. 1994. No. 47. P. 40—46.

2.14. Kohonen T.K. An introduction to neural computing. *Neural Networks*. 1988. No. 1. P. 3—16.

2.15. Kohonen T.K. Self-organization and Associative memory. New York: Springer—Verlag, 1988. 312 p.

2.16. Kohonen T.K. Self-organized formation of topologically correct feature maps. *Biol. Cybernet.* 1982. No. 43. P. 59—69.

2.17. Hamming R.W. Coding and Information Theory. Prentice-Hall, 1986. 259 p.

РОЗДІЛ 3

СИСТЕМИ НЕЧІТКОГО ЛОГІЧНОГО ВИСНОВКУ, НЕЧІТКІ НЕЙРОННІ МЕРЕЖІ ТА ЇХ ЗАСТОСУВАННЯ В ЗАДАЧАХ ПРОГНОЗУВАННЯ

3.1. Алгоритми нечіткого логічного висновку

3.2. Методи приведення до чіткості

3.3. Теореми про універсальну апроксимацію

3.4. Нечіткі нейронні мережі з висновком Мамдані і Цукамото

3.5. Градієнтний алгоритм навчання ННК Мамдані і Цукамото

3.6. Нечітка нейронна мережа *ANFIS*

3.7. Нечіткі нейронні мережі *TSK* та Ванга — Менделя

3.8. Каскадна нечітка (неофаззі) нейронна мережа

3.9. Експериментальні дослідження каскадної нечіткої (неофаззі) мережі

Висновки

Список літератури

Останніми роками значна увага дослідників у галузі обчислювального інтелекту зосереджена на проблемах прийняття рішень (ПР) за умов невизначеності, неповноти вихідних даних і якісних критеріїв. Новий напрям у теорії прийняття складних рішень, який активно розвивається, — це прийняття рішень в умовах невизначеності. Перспективним інструментом для вирішення таких завдань став апарат нечітких множин і відносин, запропонований Лотфі Заде.

Лотфі Заде — видатний американський математик і логік, автор терміна «нечітка логіка», засновник теорії нечітких множин, лінгвістичних змінних та їх використання в нечіткій логіці. Він також розробив новий напрям у галузі штучного інтелекту — *Soft Computing* (м'які обчислення), заснований на нечіткій математиці та нейронних мережах. Його підходи широко застосовують у сфері обчислювального інтелекту. Запровадження ученим поняття лінгвістичних змінних, що описуються за допомогою нечітких множин, сприяло створенню нового класу систем — системи нечіткого логічного висновку (НЛВ), що дають можливість формалізувати експертні знання з нечіткістю. Використання таких систем, а також нечітких нейронних мереж (ННМ), побудованих на їх основі, значно розширило можливості вирішення завдань прийняття рішень в умовах невизначеності, неповноти та якісної інформації, зокрема в прогнозуванні, класифікації, кластерному аналізі та розпізнаванні образів.

3.1. Алгоритми нечіткого логічного висновку

Механізм нечітких висновків, який широко використовується в експертних і керівних системах, ґрунтується на базі знань, сформованій фахівцями предметної галузі у вигляді сукупності нечітких предикатних правил такого виду [3.1, 3.2, 3.5]:

$$П_1: \text{якщо } x \in A_1, \text{ то } y \in B_1;$$
$$П_2: \text{якщо } x \in A_2, \text{ то } y \in B_2;$$
$$\dots$$
$$П_n: \text{якщо } x \in A_n, \text{ то } y \in B_n,$$

де x, $x \in X$ — вхідна змінна (ім'я для відомих значень даних); y, $y \in Y$ — змінна виводу (ім'я для значення даних, яке буде обчислено); A_i і B_i — функції належності, задані відповідно на множинах X і Y.

Наведемо детальніше пояснення. Знання експерта $A \to B$ відображає нечітке причинне відношення передумов і висновків, тому його можна назвати нечітким відношенням і позначити через R:

$$R : A \to B,$$

де «→» називають нечіткою імплікацією.

Відношення R можна розглядати як нечітку підмножину прямого добутку $X \times Y$ повної множини передумов X і висновків Y. Отже, процес отримання (нечіткого) результату виведення B' з використанням даного спостереження A' і знання $A \to B$ можна подати у вигляді композиційного правила — нечіткий «*modus ponens*»:

$$B' = A' \bullet R = A' \bullet (A \to B),$$

де «•» — операція згортки.

Як операцію композиції, так і операцію імплікації в алгебрі нечітких множин можна реалізовувати по-різному (при цьому буде відрізнятися і отримуваний результат), але в будь-якому разі загальний логічний висновок здійснюється за такі чотири кроки [3.1, 3.9]:

1. **Введення нечіткості** (фазифікація — *fuzzification*). Функції належності, визначені на вхідних змінних, застосовуються до їх фактичних значень для визначення ступеня істинності кожної передумови (антецеденту) кожного правила.

2. **Логічний висновок** виконується в два кроки. На першому кроці знаходиться ступінь істинності всіх передумов правил α. Для цього використовується операція перетину функцій належності передумов правил у формі операцій мінімума або множення.

На другому кроці обчислене значення істинності для передумов кожного правила α застосовується до висновків (консеквенту) кожного правила. Це призводить до однієї нечіткої підмножини, яка буде призначена кожною змінною виводу для кожного правила. Як правило, для логічного висновку зазвичай використовуються тільки операції «*min*» (МІНІМУМ) або «*prod*» (МНОЖЕННЯ). У логічному висновку МІНІМУМУ функція належності висновку «відсікається» по висоті, відповідній обчисленій мірі істинності передумов правила α (нечітка логіка «І»). У логічному висновку МНОЖЕННЯ функція належності висновку масштабується за допомогою обчисленого ступеня істинності передумов правила.

3. **Композиція.** Всі нечіткі підмножини, призначені до кожної змінної висновку (у всіх правилах), об'єднуються разом, щоб сформувати одну за-

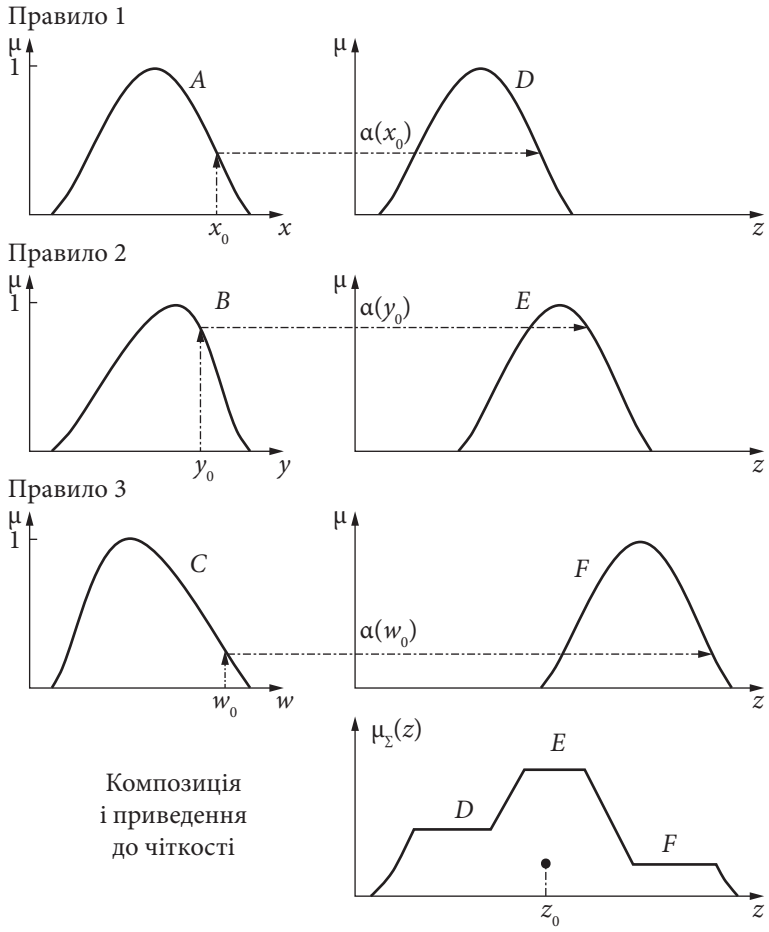


Рис. 3.1. Процедура отримання логічного висновку

гальну нечітку підмножину для всіх правил висновку. За такого об'єднання зазвичай використовуються операції «*max*» (МАКСИМУМ) або «*sum*» (СУМА). За композиції МАКСИМУМ комбінований висновок загальної нечіткої підмножини конструюється як поточковий максимум за всіма нечіткими підмножинами (нечітка логіка «АБО»). За композиції СУМИ комбінований висновок загальної нечіткої підмножини формується як поточкова сума за всіма нечіткими підмножинами висновків правил.

4. **Приведення до чіткості** (дефазифікація — *defuzzification*) використовують, якщо потрібно перетворити нечіткий набір висновків на чітке число. Деякі з методів приведення до чіткості розглянуто далі.

Приклад 3.1. Нехай деяка система описується такими нечіткими правилами:

$П_1$: якщо $x \in A$, то $z \in D$;

$П_2$: якщо $y \in B$, то $z \in E$;

$П_3$: якщо $w \in C$, то $z \in F$,

де x, y і w — імена вхідних лінгвістичних змінних; z — ім'я змінної виводу; A, B, C, D, E, F — задані функції належності.

Процедуру отримання логічного висновку показано на рис. 3.1. Передбачається, що задані конкретні (чіткі) значення вхідних змінних — x_0, y_0, w_0.

На першому кроці виходячи з даних значень і функцій належності A, B, C знаходять ступінь істинності $\alpha(x_0)$, $\alpha(y_0)$ і $\alpha(z_0)$ для передумови кожного з трьох наведених правил. На другому кроці відбувається «відсікання» функцій належності висновків правил (D, E, F) на рівнях $\alpha(x_0)$, $\alpha(y_0)$ і $\alpha(z_0)$. На третьому кроці розглядають функції належності, відсічені на попередньому кроці, і об'єднують їх з використанням операції «*max*», у результаті чого виходить комбінована нечітка підмножина, що описується функцією належності $\mu_\Sigma(z)$, і відповідне логічному висновку для вихідної змінної z. Нарешті, на четвертому етапі за потреби знаходять чітке значення вихідної змінної, наприклад, із застосуванням центроїдного методу, за яким чітке значення вихідної змінної визначається як центр ваг для кривої $\mu_\Sigma(z_0)$.

Приклад 3.2. Нехай задано нечітку систему логічного висновку, яка описується двома нечіткими правилами висновку, такого виду:

z = два нечіткі правила виду:

$П_1$: якщо $x \in A_1$ та $y \in B_1$, то $z \in C_1$;

$П_2$: якщо $x \in A_2$ та $y \in B_2$, то $z \in C_2$;

де x і y — імена вхідних змінних; z — ім'я змінної виводу; A_1, B_1, C_1, A_2, B_2, C_2 — деякі задані функції належності лінгвістичних змінних. При цьому чітке значення z_0 необхідно визначити на основі наведеної інформації та чітких значень x_0 і y_0.

Розглянемо основні алгоритми нечіткого висновку для цього прикладу.

Алгоритм нечіткого висновку Мамдані

Розгдлянемо алгоритм *нечіткого висновку Мамдані* (Ебрахім Мамдані— англійський вчений у галузі математики та інформатики, розробник одного з перших алгоритмів нечіткого логічного висновку та його практичного застосування в системах управління — відомого контролера Мамдані. Професор математики Імперського коледжу Лондона). Він відповідає розглянутому прикладу, зображеному на рис. 3.2, і математично може бути описаний так [3.5, 3.9].

1. Введення нечіткості. Знаходять ступені істинності для передумов кожного правила: $A_1(x_0)$, $A_2(x_0)$, $B_1(y_0)$, $B_2(y_0)$.

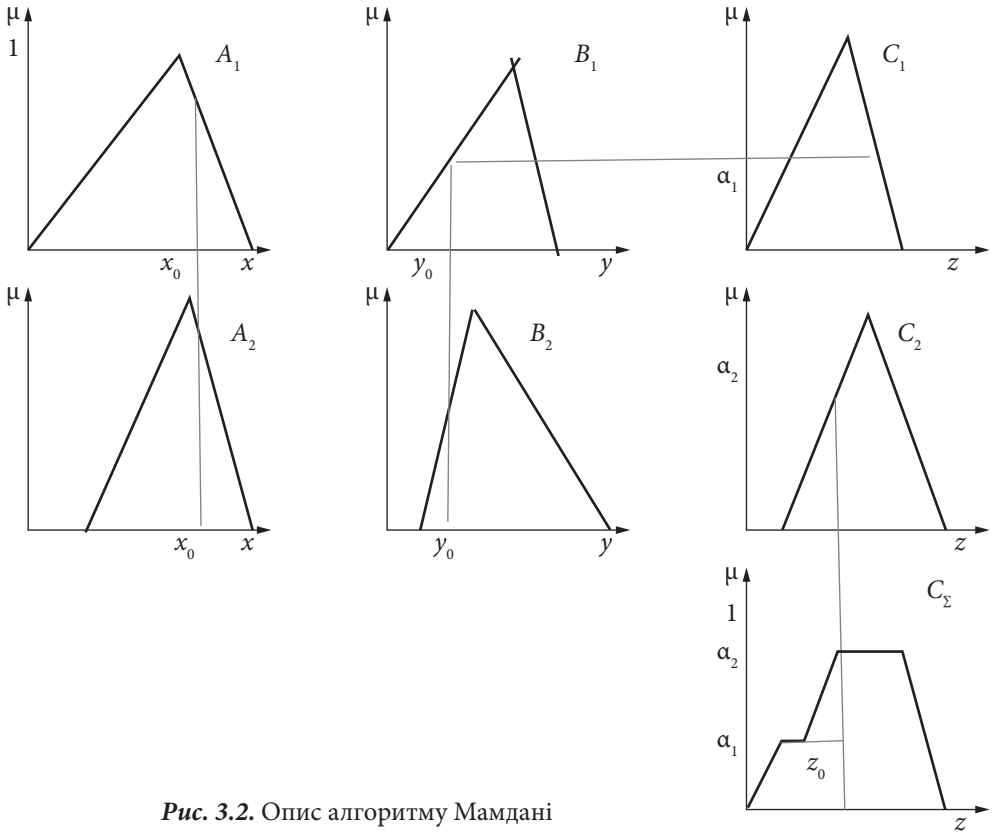


Рис. 3.2. Опис алгоритму Мамдані

2. Логічний висновок. Знаходять рівні «відсікання» для передумов кожного з правил (з використанням операції МІНІМУМ):

$$\alpha_1 = A_1(x_0) \wedge B_1(y_0);$$

$$\alpha_2 = A_2(x_0) \wedge B_2(y_0),$$

де через «^» позначено операцію логічного мінімуму («*min*»). Потім знаходять «відсічені» функції належності висновків правил:

$$C'_1 = (\alpha_1 \wedge C_1(z));$$

$$C'_2 = (\alpha_2 \wedge C_2(z)).$$

3. Композиція. Виконують об'єднання знайдених відсічених функцій за допомогою операції МАКСИМУМ («*max*», позначена далі як «∨»), що дає змогу отримати підсумкову нечітку підмножину для змінної виходу з функцією належності:

$$\mu_\Sigma(z) = C_\Sigma(z) = C_1'(z) \vee C_2'(z) = (\alpha_1 \wedge C_1(z)) \vee (\alpha_2 \wedge C_2(z)). \qquad (3.1)$$

4. Приведення до чіткості. Виконується для знаходження, наприклад, центроїдним методом: $z_0 = \frac{\int z\mu_\Sigma(z)dz}{\int \mu_\Sigma(z)dz}$.

Роботу алгоритму Мамдані наведено на рис. 3.2.

Алгоритм нечіткого висновку Цукамото

Вихідні дані такі самі, як у попередньому алгоритмі, але тут передбачається, що функції $C_1(z)$, $C_2(z)$ є монотонними [3.5, 3.9, 3.18]. Опис алгоритму наведено на рис. 3.3.

1. Введення нечіткості (так само, як в алгоритмі Мамдані).

2. Нечіткий висновок. Спочатку знаходять рівні «відсікання» α_1 та α_2 (як в алгоритмі Мамдані), а потім розв'язують рівняння

$$\alpha_1 = C_1(z_1) \text{ та } \alpha_2 = C_2(z_2).$$

При цьому визначають чіткі значення (z_1 і z_2) для кожного вихідного правила.

3. Композиція висновків правил центроїдним методом. Визначають чітке значення змінної виводу (як зважене середнє z_1 і z_2):

$$z_0 = \frac{\alpha_1 z_1 + \alpha_2 z_2}{\alpha_1 + \alpha_2}. \tag{3.2}$$

У загальному випадку (дискретний варіант центроїдного методу) маємо такий вираз:

$$z_0 = \frac{\sum_{i=1}^{n} \alpha_i z_i}{\sum_{i=1}^{n} \alpha_i}. \tag{3.3}$$

4. Дефазифікацію не виконують, оскільки вихідне значення є чітким.

Алгоритм нечіткого висновку Сугено

Сугено і Такагі використовували набір правил у такій формі (як і раніше, наведемо приклад для двох правил):

Π_1: якщо $x \in A_1$ та $y \in B_1$, то $z_1 = a_1 x + b_1 y + c_1$;

Π_2: якщо $x \in A_2$ та $y \in B_2$, то $z_2 = a_2 x + b_2 y + c_2$.

Опис алгоритму наведено на рис. 3.4.

1. Введення нечіткості (як в алгоритмі Мамдані).

2. Нечіткий висновок.

Спочатку знаходимо ступінь виконання передумов кожного правила:

$$\alpha_1 = A_1(x_0) \wedge B_1(y_0) = A_1(x_0) * B_1(y_0);$$

$$\alpha_2 = A_1(x_0) \wedge B_1(y_0) = A_1(x_0) * B_1(y_0).$$

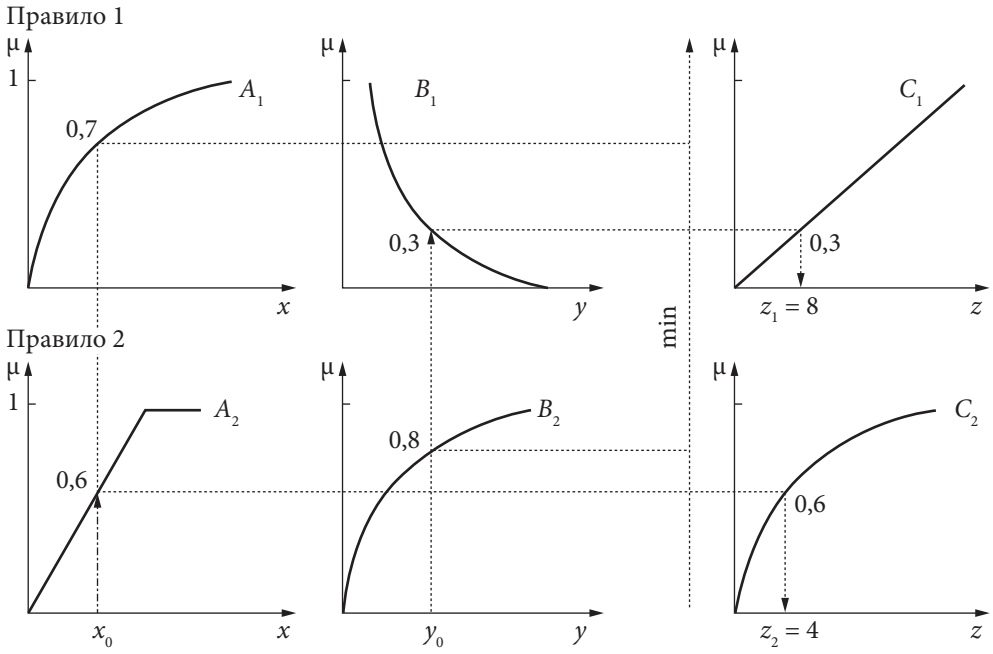


Рис. 3.3. Опис алгоритму Цукамото

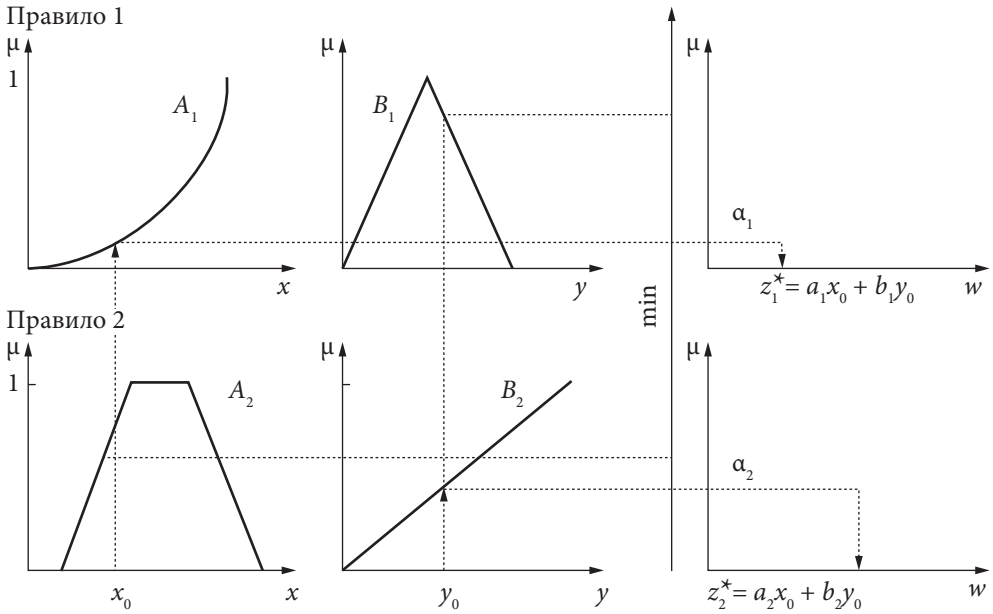


Рис. 3.4. Опис алгоритму Сугено

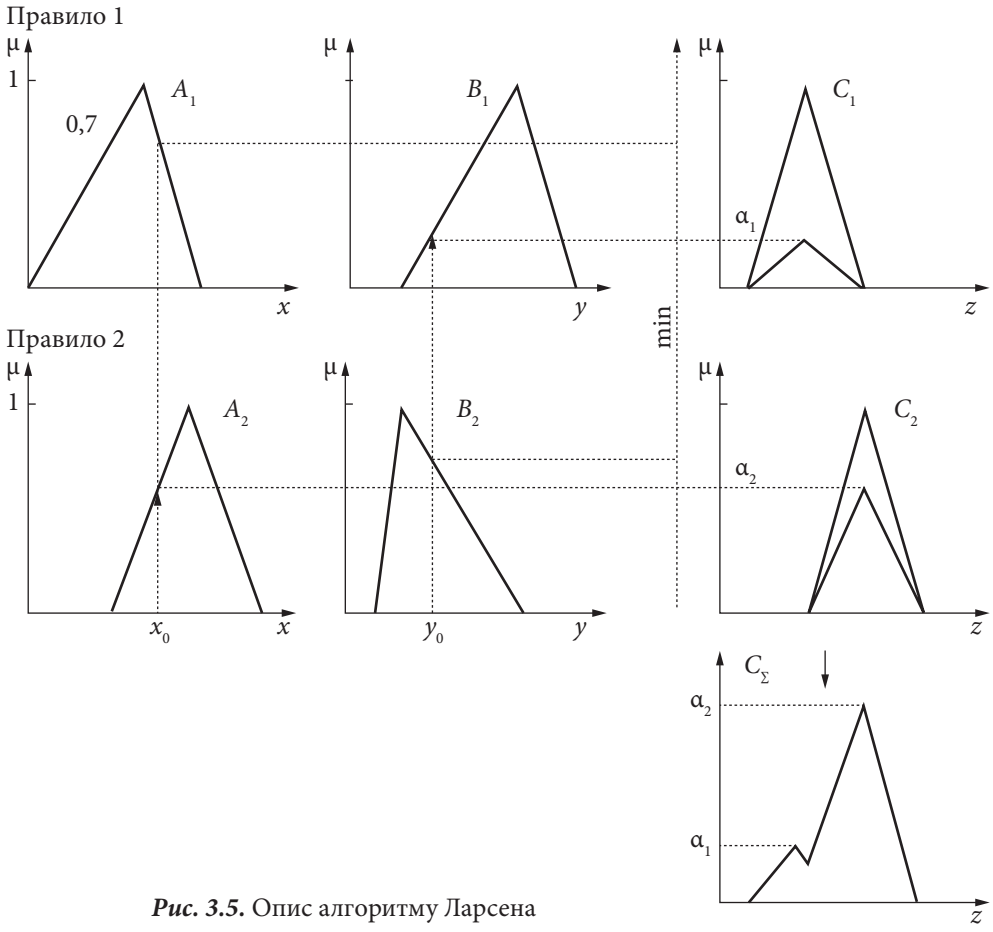


Рис. 3.5. Опис алгоритму Ларсена

Далі знаходимо індивідуальні висновки правил:

$$\dot{z}_1 = a_1 x_0 + b_1 y_0 + c_1;$$

$$\dot{z}_2 = a_2 x_0 + b_2 y_0 + c_2.$$

3. Виконуємо композицію висновків правил центроїдним методом:

$$z_0 = \frac{\alpha_1 \dot{z}_1 + \alpha_2 \dot{z}_2}{\alpha_1 + \alpha_2}.$$

У загальному випадку знаходимо (дискретний варіант центроїдного методу):

$$z_0 = \frac{\sum_{i=1}^{n} \alpha_i z_i}{\sum_{i=1}^{n} \alpha_i}.$$

4. Дефазифікацію не виконують, як і в методі Цукамото, оскільки висновки правил чіткі.

Алгоритм нечіткого висновку Ларсена

В алгоритмі Ларсена нечітка імплікація моделюється з використанням оператора множення. Опис алгоритму зображено на рис. 3.5.

1. Введення нечіткості (як в алгоритмі Мамдані).

2. Нечіткий висновок. Спочатку, як у алгоритмі Сугено, знаходять значення

$$\alpha_1 = A_1(x_0) \wedge B_1(y_0) = A_1(x_0) * B_1(y_0);$$
$$\alpha_2 = A_2(x_0) \wedge B_2(y_0) = A_2(x_0) * B_2(y_0),$$

а потім визначають вихідні нечіткі підмножини для кожного правила:

$$\alpha_1 C_1(z),\ \alpha_2 C_2(z).$$

3. Знаходять загальну вихідну нечітку підмножину:

$$\mu_\Sigma(z) = C(z) = (\alpha_1 C_1(z)) \vee (\alpha_2 C_2(z)).$$

У загальному випадку для n правил $\mu_\Sigma(z) = C(z) = \vee_{i=1}^{n}(\alpha_i C_i(z))$.

4. За потреби виконують приведення до чіткості (як у розглянутих алгоритмах).

3.2. Методи приведення до чіткості

Наведемо основні методи дефазифікації.

1. Вище вже було розглянуто один із цих методів — центроїдний. Наведемо відповідні формули ще раз. У загальному випадку маємо:

а) для неперервного варіанта

$$z_0 = \frac{\int_\Omega zC(z)dz}{\int_\Omega C(z)dz}, \tag{3.4}$$

б) для дискретного варіанта

$$z_0 = \frac{\sum_{i=1}^{n}\alpha_i z_i}{\sum_{i=1}^{n}\alpha_i}.$$

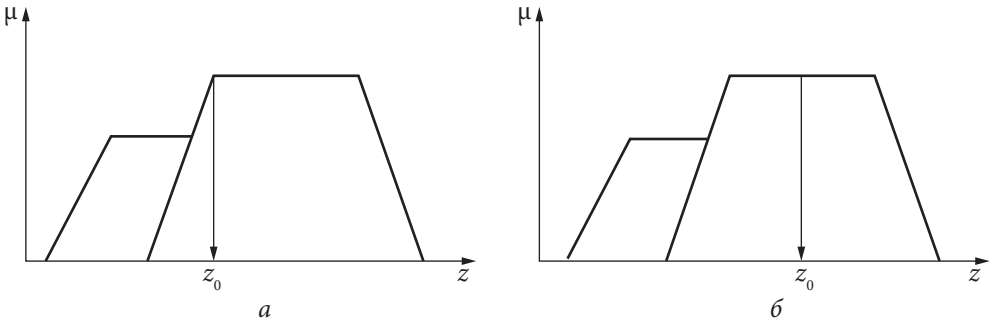
2. Перший максимум. Чітку величину висновків знаходять як найменше значення, за якого досягається максимум підсумкової функції належності нечіткої множини (рис. 3.6, *а*):

$$z_0 = \min\{z \mid C(z) = \max_U C(U)\}.$$

3. Середній максимум (рис. 3.6, *б*). Чітке значення знаходять за формулою

$$z_0 = \frac{\int_G zdz}{\int_G dz}, \tag{3.5}$$

де G — підмножина елементів, які максимізують C.



а *б*

Рис. 3.6. Середній максимум

Для дискретного варіанта (*G* — дискретне) маємо:

$$z_0 = \frac{1}{n} \sum_{i=1}^{n} z_i. \tag{3.6}$$

4. Критерій максимуму. Чітке значення вибирається довільно серед множини елементів, для яких *C* досягає максимуму:

$$z_0 \in \{z : C(z) = \max_u C(\mathrm{U})\}. \tag{3.7}$$

5. Висотна дефазифікація. Елементи зони визначення Ω, для яких значення функції належності менше, ніж деякий рівень α, у розрахунок не беруть, і чітке значення розраховують відповідно до виразу:

$$z_0 = \frac{\int_{C\alpha} z \cdot C(z)dz}{\int_{C\alpha} C(z)dz}, \tag{3.8}$$

де C_α — нечітка множина α рівня (див. вище).

3.3. Теореми про універсальну апроксимацію

Можливість використовувати апарат нечіткої логіки ґрунтується на таких результатах [3.26]:

1. У 1992 р. Ванг показав, що нечітка система є універсальним апроксиматором, тобто може апроксимувати будь-яку неперервну функцію на компакті *U* з довільною точністю, якщо використовує набір *n* ($n \to \infty$) правил:

$П_i$: якщо $x \in A_i$ і $y \in B_i$, то $z \in C_i$, $i = 1, \ldots, n$ за таких умов:

• гауссові функції належності:

$$A_i(x) = \exp\left[-\frac{1}{2}\left(\frac{x - \alpha_{i1}}{\beta_{i1}}\right)^2\right],$$

$$B_i(y) = \exp\left[-\frac{1}{2}\left(\frac{y - \alpha_{i2}}{\beta_{i2}}\right)^2\right], \tag{3.9}$$

$$C_i(z) = \exp\left[-\frac{1}{2}\left(\frac{z - \alpha_{i3}}{\beta_{i3}}\right)^2\right];$$

• перетин у вигляді добутку: $[A_i(x) \text{ і } B_i(y)] = A_i(x)B_i(y)$;
• імплікація у формі Ларсена: $[A_i(x) \text{ і } B_i(y)] \to C_i(z) = A_i(x)B_i(y) \cdot C_i(z)$;
• центроїдний метод приведення до чіткості:

$$z_0 = \frac{\sum_{i=1}^{n} \alpha_{i3} A_i B_i}{\sum_{i=1}^{n} A_i B_i}, \tag{3.10}$$

де α_{i3} — центри C_i.

Інакше кажучи, Ванг довів теорему: для кожної дійсної неперервної функції g, заданої на компакті U, і для довільного $\varepsilon > 0$ існує нечітка система, яка генерує вихідну функцію $f(x)$ таку, що

$$\sup_{x \in U} ||g(x) - f(x)|| \le \varepsilon,$$

де $||\bullet||$ — символ прийнятої відстані між функціями.

2. У 1995 р. Коско показав, що логічний контролер Мамдані також є універсальним апроксиматором за таких умов [3.20, 3.21]:

• симетричні трикутні функції належності:

$$A_i(x) = \begin{cases} \dfrac{1 - |a_i - x|}{\alpha_i}, \text{ якщо } |a_i - x| \le \alpha_i; \\ 0, \text{ якщо } |a_i - x| > \alpha_i; \end{cases}$$

$$B_i(y) = \begin{cases} \dfrac{1 - |b_i - y|}{\beta_i}, \text{ якщо } |b_i - y| \le \beta_i; \\ 0, \text{ якщо } |b_i - y| > \beta_i; \end{cases}$$

• перетин з використанням операції «*min*»:

$$[A_i(x) \cap B_i(y)] = \min\{A_i(x), B_i(y)\};$$

• імплікація у формі Мамдані і центроїдний метод приведення до чіткості:

$$z_0 = \frac{\sum_{i=1}^{n} \alpha_{i3} A_i B_i}{\sum_{i=1}^{n} A_i B_i}, \tag{3.11}$$

де α_{i3} — центри C_i.

Загалом системи з нечіткою логікою доцільно застосовувати у таких випадках [3.5, 3.9]:

• для складних процесів, коли немає простої математичної моделі;
• якщо експертні знання про об'єкт чи процес можна сформулювати лише у лінгвістичній формі.

Системи, що ґрунтуються на нечіткій логіці, застосовувати недоцільно у таких випадках:

• якщо необхідний результат можна отримати іншим (стандартним) шляхом;

• якщо для об'єкта чи процесу вже знайдено адекватну і легко досліджувану математичну модель.

Основні переваги систем з нечіткою логікою:

• дають можливість працювати з якісною інформацією завдяки використанню лінгвістичних змінних, а також з нечітко заданою або неповною інформацією;

• дають можливість використовувати знання експертів у вигляді нечітких правил виведення.

Основні недоліки таких систем:

• вихідний набір нечітких правил формулюється експертом-людиною і може виявитися неповним або суперечливим;

• вид і параметри функцій належності, які описують вхідні та вихідні змінні системи, вибираються суб'єктивно і можуть бути такими, що не повністю відображають реальність.

Для усунення зазначених недоліків необхідно використовувати навчання систем нечіткої логіки, тобто зробити їх адаптивними.

3.4. Нечіткі нейронні мережі з висновком Мамдані і Цукамото

Нечіткий контролер на основі нейронних мереж. Одним із перших практичних застосувань систем з нечіткою логікою була сфера управління, для чого широко використовувалися так звані нечіткі контролери (НК) [3.9].

Для проєктування НК мають бути задані лінгвістичні правила та функції належності для зображення лінгвістичних величин. Специфікація задовільних лінгвістичних правил залежить від знання експертом системи управління. Однак трансформація цих знань у нечіткі множини є неформалізованим завданням, і потрібно зробити її вибір на підставі, наприклад, форми функції належності (ФН). Якість нечіткого контролера досягається зміною форми ФН.

Штучні нейронні мережі (ШНМ) є високопаралельними архітектурами і складаються з аналогічних елементів, що взаємодіють через зв'язки, які задаються вагами. Використовуючи НК, можливо не лише апроксимувати функції, а й вивчати (досліджувати) об'єкти управління, застосовуючи навчання та самонавчання. Проблема полягає в тому, що на навчання витрачається досить багато часу і при цьому не завжди гарантується результат. Однак можна запровадити раніше набуті знання у вигляді правил вже навченої НМ для спрощення процедури навчання.

Інтеграція НК та НМ дає змогу об'єднати всі їхні переваги і уникнути недоліків. З метою інтеграції двох технологій — систем нечіткої логіки (СНЛ) та НМ структуру СНЛ зображують у вигляді нейронної мережі. Цей підхід використовує НМ для оптимізації кінцевих параметрів звичайного НК або для отримання правил даних. Вибір нечіткої множини, що позначає лінгвістичний терм, більш-менш довільний. Наприклад, розглянемо лінгвістичний терм «приблизно нуль». Очевидно, що відповідна нечіткій множині ФН має бути унімодальною і досягати свого максимуму в значенні «нуль». Правильний вибір функції належності є одним з основних і найважливіших завдань НК.

НМ пропонує можливість вирішення цієї проблеми. Метод прямого поширення (сигналів) у НМ допускає вибір форми ФН, яка залежить від декількох параметрів і може бути скоригована в процесі навчання. Як ФН можна взяти симетричну трикутну форму, що залежить від двох параметрів, один з яких визначається значенням, за якого ФН досягає максимального значення, а другий — довжиною інтервалу.

Дані навчання мають бути поділені на r кластерів: R_1, …, R_r. Кожний кластер R_i відповідає вирішальному правилу R_i. Елементи кластера зображують у вигляді значень у формі (X, Y), де $X = [x_1, \ldots, x_n]$ — вектор вхідних змінних, Y — вихідна змінна.

Розглянемо динамічну систему S, в якій управління здійснюється за допомогою змінної C, і її стан може бути описано n змінними: $x_1, \ldots, x_n$.

Лінгвістичні значення змінних моделюються функціями належності, а керівна дія, що приводить систему до бажаного стану, описується нечітким правилом «ЯКЩО, ТО». Щоб отримати шукане значення (тобто управління), потрібно вирішити проблему дефазифікації, для чого використовуємо монотонну функцію належності Цукамото, де дефазифікація зводиться до застосування оберненої функції.

Структуру нечіткого нейронного контролера наведено на рис. 3.7. Модулі x_1 і x_2 тут є вхідними змінними, вони посилають свої значення у μ-модулі, які містять відповідні ФН. μ-Модулі пов'язані з R-модулями, які є нечіткими правилами «якщо, то». Кожен μ-модуль передає всім пов'язаним з ним R-модулям значення ФН $\mu_{ij}(x_i)$ її вхідної величини x_i. R-Модуль використовує операцію перетину, знаходить і передає це значення далі, в ν-модуль, який містить ФН, що описує вихідні значення.

ν-Модуль, використовуючи монотонні функції належності, обчислює r_i і $\nu^{-1}(r_i)$ та передає їх в C-модуль, який обчислює підсумкову вихідну змінну — керівний вплив C згідно з формулою (3.14), тобто використовує алгоритм центру мас (COA):

$$C = \frac{\sum_{i=1}^{n} r_i \nu^{-1}(r_i)}{\sum_{i=1}^{n} r_i}, \qquad (3.14)$$

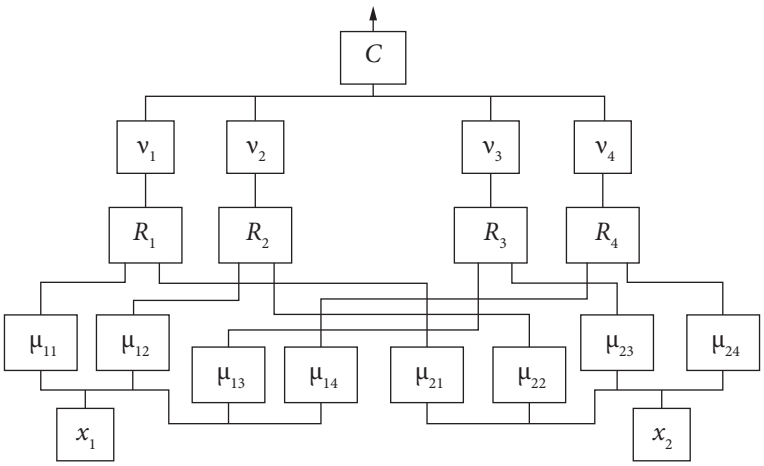


Рис. 3.7. Структура нечіткого нейронного контролера

де n — кількість правил виведення; r_i — ступінь, в якому правило R_i виконується.

Як неважко помітити, система, зображена на рис. 3.7, нагадує послідовну багатошарову НМ, де x-, R- і C-модулі виконують функції нейронів, а μ- і ν-модулі відіграють роль ваг зв'язків мережі, що адаптуються.

Процес навчання визначається нечіткою помилкою і працює паралельно для кожного нечіткого правила. Після того, як керівний сигнал C вироблений контролером і новий стан об'єкта стає відомим, обчислюється помилка E, яка поширюється у зворотному напрямку. Кожне правило аналізує свій внесок у вихід управління і оцінює свій висновок (тобто воно привело до збільшення або зменшення помилки). При цьому, якщо правило діяло в правильному напрямку, воно має стати більш чутливим і виробити такий висновок, який збільшує сигнал управління. І навпаки, якщо правило діяло в неправильному напрямку, правило має стати менш чутливим.

3.5. Градієнтний алгоритм навчання ННК Мамдані і Цукамото

Розглянемо алгоритм навчання градієнтного спуску для ННК (наведеного на рис. 3.7), збіжність якого доведено. З цією метою доцільно перейти до гауссових ФН для умов та правил [3.5, 3.9].

Отже, нехай ФН i-го μ-модуля, пов'язаного з правилом R_k, описується виразом

$$\mu_{ik}(x_i) = \exp\left\{-\frac{1}{2}\frac{(x_i - \alpha_{ik})^2}{\sigma_{ik}^2}\right\}, \qquad (3.15)$$

де a_{ik}, σ_{ik} — параметри, що зазнаватимуть налаштування в процесі навчання, а ФН ν_k-модуля має вигляд, аналогічний (3.15):

$$\mu_k(z_k) = \exp\left\{-\frac{1}{2}\frac{(z_k - \alpha_k)^2}{\sigma_k^2}\right\}. \tag{3.16}$$

При цьому перетин умов правил задається у вигляді добутку:

$$\alpha_k = \prod_{i=1}^{n}\mu_{ik}(x_i) = \exp\left\{-\sum_{i=1}^{n}-\frac{1}{2}\frac{(x_i - \alpha_{ik})^2}{\sigma_{ik}^2}\right\}. \tag{3.17}$$

Припустимо, що дефазифікація відбувається за центроїдним методом, тоді загальний вихід дорівнює

$$Z_0 = \frac{\sum_k z_k \alpha_k}{\sum_k \alpha_k}. \tag{3.18}$$

Нехай для визначення наслідків правила використовуються монотонні ФН і z_k визначаються розв'язуванням такого рівняння (контролер Цукамото):

$$C_k(z_k) = \alpha_k, \tag{3.19}$$

де

$$C_k(z_k) = \exp\left\{-\frac{1}{2}\frac{(z_k - \alpha_k)^2}{\sigma_k^2}\right\}. \tag{3.20}$$

Тоді, розв'язуючи рівняння (3.19) $\exp\left\{-\frac{1}{2}\frac{(z_k - \alpha_k)^2}{\sigma_k^2}\right\} = \alpha_k$, знаходимо два корені:

$$Z_k = \alpha_k \pm \sqrt{2\ln\frac{1}{\alpha_k}}\cdot\sigma_k.$$

Перший корінь $Z_{lk} = \alpha_k - \sqrt{2\ln\frac{1}{\alpha_k}}\cdot\sigma_k$ знаходять на монотонно зростаючій ділянці кривої $C_k(z_k)$, а другий z_{2k} — на монотонно спадній ділянці цієї кривої.

Нехай критерій $E(z) = \frac{1}{2}(z_0 - z^*)^2 \to \min$, де z^* — фактичний вихід, а z_0 — вихід НК. Тоді знаходимо похідні [3.2, 3.9]:

$$\frac{\partial E}{\partial a_k} = \frac{\partial E}{\partial z_0}\frac{\partial z_0}{\partial z_k}\frac{\partial z_k}{\partial a_k} = +(z_0 - z^*)\frac{\alpha_k}{\sum_{k=1}^{K}\alpha_k}; \tag{3.21}$$

$$\frac{\partial E}{\partial \sigma_k} = \frac{\partial E}{\partial z_0}\frac{\partial z_0}{\partial z_k}\frac{\partial z_k}{\partial \sigma_k} = -(z_0 - z^*)\frac{\alpha_k}{\sum_{k=1}^{K}\alpha_k}\sqrt{2\ln\frac{1}{\alpha_k}} \tag{3.22}$$

на монотонно зростаючій ділянці кривої ФП μ_k і

$$\frac{\partial E}{\partial \sigma_k} = (z_0 - z^*)\frac{\alpha_k}{\sum_{k=1}^{K}\alpha_k}\sqrt{2\ln\frac{1}{\alpha_k}} \tag{3.23}$$

на монотонно спадній ділянці кривої.

Для вхідних μ-модулів знаходимо похідні:

$$\frac{\partial E}{\partial \alpha_{ik}} = \frac{\partial E_0}{\partial z_0}\frac{\partial z_0}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial \alpha_{ik}} = +(z_0 - z^\star)\frac{z_k\sum_k^K \alpha_k - \sum_k^K z_k\alpha_k}{(\sum_k \alpha_k)^2}\prod_{k=1}^K \exp\left\{-\frac{(x_i - \alpha_{ik})^2}{2\sigma_{ik}^2}\right\} \times$$

$$\times \frac{(x_i - \alpha_{ik})}{\sigma_{ik}^2} = (z_0 - z^\star)\alpha_k \frac{z_k\sum_k^K \alpha_k - \sum_k z_k\alpha_k}{(\sum_k \alpha_k)^2}\frac{(x_i - \alpha_{ik})^2}{\sigma_{ik}^2};$$

$$\frac{\partial E}{\partial \sigma_{ik}} = \frac{\partial E}{\partial z_0}\frac{\partial z_0}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial \sigma_{ik}} = \left((z_0 - z^\star)\alpha_k \frac{z_k\sum_k^K \alpha_k - \sum_k z_k\alpha_k}{(\sum_k \alpha_k)^2}\frac{(x_i - \alpha_{ik})^2}{\sigma_{ik}^3}\right). \qquad (3.24)$$

Тоді градієнтний алгоритм навчання ННК Мамдані має такий вигляд [3.2, 3.9]:

а) для вихідних модулів

$$a_k(n+1) = a_k(n) - \gamma_n\frac{\partial E}{\partial \alpha_k} = a_k(n) - \gamma_n(z_0 - z^\star)\frac{\alpha_k}{\sum_k \alpha_k}; \qquad (3.25)$$

$$\sigma_i(n+1) = \sigma_k(n) - \gamma'_n\frac{\partial E}{\partial \sigma_k} = \sigma_k(n) \pm \gamma_{n_1}(z_0 - z^\star)\frac{\alpha_k}{\sum_k \alpha_k}\sqrt{2\ln\frac{1}{\alpha_k}}; \qquad (3.26)$$

б) для вхідних μ-модулів

$$a_{ik}(n+1) = a_{ik}(n) - \gamma_{n_2}\frac{\partial E}{\partial \alpha_{ik}} =$$

$$= a_{ik}(n) - \gamma_{n_2}(z_0 - z^\star)\frac{\alpha_k(z_k\sum_k^K \alpha_k - \sum_k^K z_k\alpha_k)}{(\sum_k \alpha_k)^2}\frac{(x_i - \alpha_{ik})}{\sigma_{ik}^2}; \qquad (3.27)$$

$$\sigma_{ik}(n+1) = \sigma_{ik}(n) - \gamma_{n_3}(z_0 - z^\star)\frac{\alpha_k(z_k\sum_k^K \alpha_k - \sum_k^K z_k\alpha_k)}{(\sum_k^K \alpha_k)^2}\frac{(x_i - \alpha_{ik})^2}{\sigma_{ik}^3}, \qquad (3.28)$$

де γ_n, γ_{n_1}, γ_{n_2}, γ_{n_3} — розмір кроку.

Для збіжності методу необхідно, щоб виконувались такі умови [3.2, 3.6]:

а) $\gamma_n \to 0$, $n \to \infty$;

б) $\sum_{n=0}^{\infty}\gamma_n = \infty$, $\sum_{n=0}^{\infty}\gamma_n^2 < \infty$.

3.6. Нечітка нейронна мережа *ANFIS*

Розглянемо адаптивну нечітку систему з механізмом логічного висновку, запропонованого Сугено на базі правил «ЯКЩО, ТО» [3.2, 3.9], яка дістала назву «мережа *ANFIS*» (*Adaptive Network Based Fuzzy Inference System*). Цю НМ можна успішно використовувати для налаштування функції належності та налаштування бази правил у нечіткій експертній системі. Далі

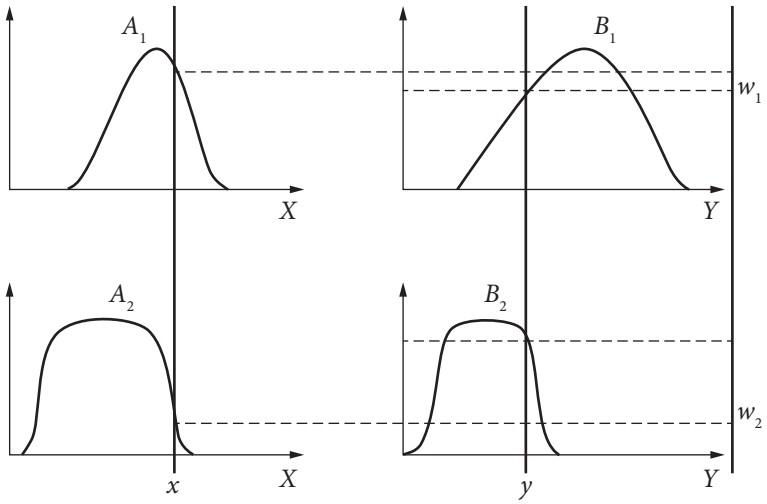


Рис. 3.8. Модель нечіткого логічного висновку Сугено

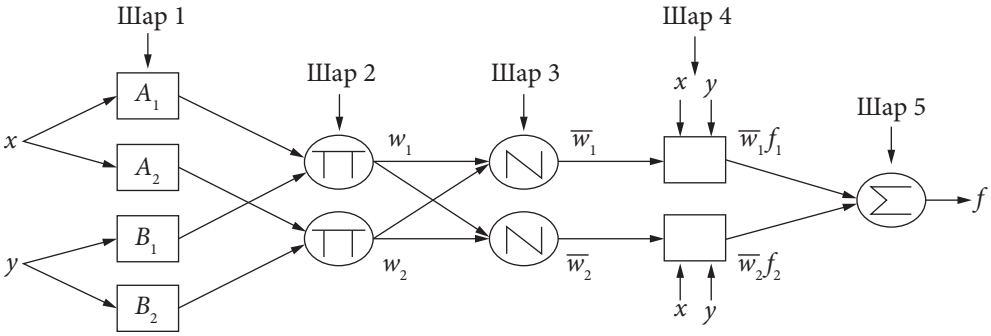


Рис. 3.9. Структурна схема нейронної мережі *ANFIS*

наведено модель нечіткого логічного висновку Сугено (рис. 3.8) та структурну схему нейронної мережі *ANFIS* (рис. 3.9), де

$$f_1 = a_1 x + b_1 y + r_1;$$

$$f_2 = a_2 x + b_2 y + r_2;$$

$$f = \frac{w_1 f_1 + w_2 f_2}{w_1 + w_2} = \overline{w}_1 f_1 + \overline{w}_2 f_2.$$

Нейронна мережа *ANFIS* використовує таку базу правил:

якщо $x = A_1$ та $y = B_1$, то $f_1 = a_1 x + b_1 y + r_1$;

якщо $x = A_2$ та $y = B_2$, то $f_2 = a_2 x + b_2 y + r_2$,

де A_i і B_i — лінгвістичні змінні; a_i, b_i, r_i — деякі константи.

Шари цієї нечіткої нейронної мережі виконують такі функції:

Шар 1. Кожний нейрон цього шару перетворює вхідний сигнал x або y за допомогою функції належності (фазифікатор). Найчастіше використовують дзвоноподібну функцію вигляду

$$\mu_{Ai}(x) = \frac{1}{1 + \left[\left(\frac{x - c_i}{a_i}\right)^2\right]} \tag{3.29}$$

або функцію Гаусса:

$$\mu_{Ai}(x) = \exp\left[-\left(\frac{x - c_i}{a_i}\right)^2\right]. \tag{3.30}$$

Шар 2. Кожний нейрон у цьому шарі, позначений як П, здійснює перетин множини вхідних сигналів, моделюючи логічну операцію AND, і посилає на вихід сигнал w_i:

$$w_i = \mu_{Ai}(x) \times \mu_{Bi}(y), \quad i = 1, 2. \tag{3.31}$$

Шар 3. Кожний нейрон у цьому шарі обчислює нормовану силу правила:

$$\overline{w}_i = \frac{w_i}{w_1 + w_2}, \quad i = 1, 2. \tag{3.32}$$

Шар 4. На цьому шарі в нейронах формуються значення вихідних змінних:

$$O_i^4 = \overline{w}_i f_i = \overline{w}_i (a_i x + b_i y + r_i). \tag{3.33}$$

Шар 5. В останньому шарі отримуємо вихідний сигнал нейронної мережі та виконуємо дефазифікацію результатів:

$$O^5 = \text{overall output} = \sum_i \overline{w}_i f_i = \frac{\sum_i w_i f_i}{\sum_i w_i}. \tag{3.34}$$

Нейронна мережа архітектури *ANFIS* навчається за допомогою методу градієнтного спуску.

Відтворення бази правил і налаштування параметрів функцій належності. Одним із ключових завдань у системах з нечіткими нейронними мережами є розроблення оптимального методу налаштування бази нечітких правил на основі навчальної вибірки. Це дає можливість створювати конструктивні й оптимальні моделі нечітких систем, які можуть ефективно використовуватися в практичних задачах.

Зазвичай нечіткі правила формулюють експерти на основі їхніх знань і досвіду щодо відповідних процесів. Проте в деяких випадках створення нечітких систем ускладнюється, і отримання точних правил чи функцій належності (*membership functions*) стає досить непростим або майже неможливим через невизначеність, неповноту даних або складність самої системи. У таких ситуаціях доцільно використовувати спеціальні навчальні алгорит-

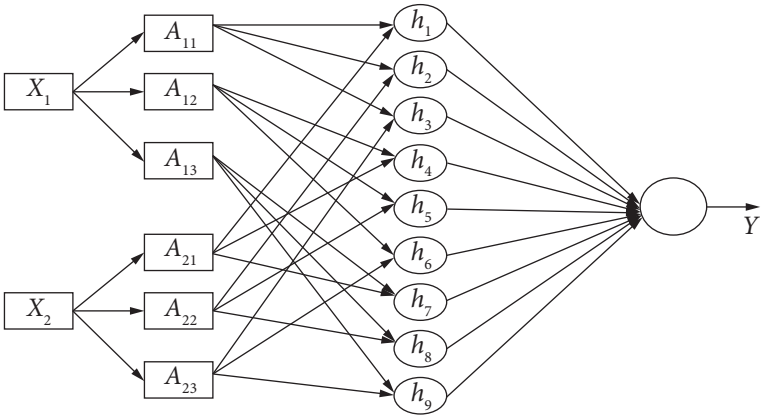


Рис. 3.10. Схема нейронної нечіткої мережі *ANFIS*

ми для генерування й уточнення нечітких правил. Серед найпопулярніших методів є алгоритм зворотного поширення помилки для нечітких мереж, описаний у п. 3.4. Цей алгоритм дає змогу створювати оптимальні моделі нечітких систем і бази правил. Він був незалежно запропонований Ічіаші (Ichihashi), Номурою (Nomura), а також Вангом і Менделем (Wang and Mendel) [3.5, 3.9]. Паралельно Ші і Міцумото запропонували інший підхід, який адаптує цей алгоритм для практичного використання [3.2].

Характерною особливістю такого підходу є налаштування параметрів нечітких правил без необхідності модифікації самої таблиці правил. Для ілюстрації розглянемо цей алгоритм на моделі, яка містить дві вхідні лінгвістичні змінні (x_1, x_2) та одну вихідну змінну (y). Схему нейронної нечіткої мережі *ANFIS* наведено на рис. 3.10.

Нехай є база правил, яка містить усі можливі комбінації A_{1i} і A_{2j} ($i = 1, \ldots, r$; $j = 1, \ldots, k$).

Правило 1: $A_{11}, A_{21} \Rightarrow y_{11}$;
2: $A_{11}, A_{22} \Rightarrow y_{12}$;
…
k: $A_{11}, A_{2k} \Rightarrow y_{1k}$;
$k + 1$: $A_{12}, A_{21} \Rightarrow y_{21}$;
…
$2k$: $A_{12}, A_{2k} \Rightarrow y_{2k}$;
Правило ij: $A_{1i}, A_{2j} \Rightarrow y_{ij}$;
…
Правило $r \times k$: $A_{1r}, A_{2k} \Rightarrow y_{r \times k}$,

де A_{1i} та A_{2j} — нечіткі множини для змінних відповідно X_1 і X_2; y_{ij} — дійсне число з Y.

Очевидно, що цей набір правил можна подати у вигляді таблиці (табл. 3.1).

З огляду на вищезазначене, якщо відомий набір величин, то, відповідно до нечіткої бази правил, вихід може бути отриманий на основі методів нечіткої логіки.

Насамперед позначимо ступінь виконання умов як

$$h_{ij} = A_{1i}(x_1)A_{2j}(x_2). \tag{3.35}$$

Відповідно до центроїдного методу запишемо загальний вихід y:

$$y = \frac{\sum_{i=1}^{r}\sum_{j=1}^{k} h_{ij}y_{ij}}{\sum_{i=1}^{r}\sum_{j=1}^{k} h_{ij}} = \frac{\sum_{i=1}^{r}\sum_{j=1}^{k} A_{1i}(x_1)\,A_{2j}(x_2)y_{ij}}{\sum_{i=1}^{r}\sum_{j=1}^{k} A_{1i}(x_1)\,A_{2j}(x_2)}. \tag{3.36}$$

У разі навчання системи за допомогою навчальної вибірки $(x_1, x_2; y^*)$ похибка системи може бути описана як $E = \dfrac{(y^* - y)^2}{2}$.

Виходячи з опису нечітких величин, для A_{1i} маємо a_{1i} — центр функції належності, b_{1i} — ширину для даної функції, аналогічно для A_{2j} отримуємо a_{2j} і b_{2j}.

Відповідно до методу градієнтного спуску для мінімізації похибки виходу E можна записати формули для розрахунку коефіцієнтів a_{1i}, a_{2j}, b_{2j} та y_{ij} ($i = 1, 2, \ldots, r$; $j = 1, 2, \ldots, k$), у такому вигляді:

$$a_{1i}(t+1) = a_{1i}(t) - \frac{\alpha\partial E}{\partial a_{1i}(t)} = a_{1i}(t) - \alpha\left(\frac{\partial E}{\partial y}\right)\left(\frac{\partial y}{\partial h_{ij}}\right)\left(\frac{\partial h_{ij}}{\partial A_{1i}}\right)\left(\frac{\partial A_{1i}}{\partial a_{1i}(t)}\right) =$$

$$= a_{1i}(t) + \frac{\alpha(y^* - y)[\sum_{j=1}^{k}(y_{ij} - y)A_{2j}]\left(\dfrac{\partial A_{1i}}{\partial a_{1i}(t)}\right)}{\sum_{i=1}^{r}\sum_{j=1}^{k} h_{ij}}; \tag{3.37}$$

Таблиця 3.1

Набір правил

x_1/x_2	A_{21}	A_{22}	…	A_{2j}	…	A_{2k}
A_{11}	y_{11}	y_{12}	…	y_{1j}	…	y_{1k}
A_{12}	y_{21}	y_{22}	…	y_{2j}		y_{2k}
…	…	…	…	…	…	…
A_{1i}	…	…	…	y_{ij}	…	…
…	…	…	…	…	…	…
A_{1r}	y_{r1}	…	…	…	…	y_{rk}

$$b_{1i}(t+1) = b_{1i}(t) - \frac{\beta\partial E}{\partial b_{1i}(t)} = b_{1i}(t) - \beta\left(\frac{\partial E}{\partial y}\right)\left(\frac{\partial y}{\partial h_{ij}}\right)\left(\frac{\partial h_{ij}}{\partial A_{1i}}\right)\left(\frac{\partial A_{1i}}{\partial b_{1i}(t)}\right) =$$

$$= b_{1i}(t) + \frac{\beta(y^{\star} - y)[\sum_{j=1}^{k}(y_{ij} - y)A_{2j}]\left(\frac{\partial A_{1i}}{\partial b_{1i}(t)}\right)}{\sum_{i=1}^{r}\sum_{j=1}^{k} h_{ij}}; \quad (3.38)$$

$$a_{2i}(t+1) = a_{2i}(t) - \frac{\alpha\partial E}{\partial a_{2i}(t)} = a_{2i}(t) - \alpha\left(\frac{\partial E}{\partial y}\right)\left(\frac{\partial y}{\partial h_{ij}}\right)\left(\frac{\partial h_{ij}}{\partial A_{2i}}\right)\left(\frac{\partial A_{2i}}{\partial a_{2i}(t)}\right) =$$

$$= a_{2i}(t) + \frac{\alpha(y^{\star} - y)[\sum_{j=1}^{k}(y_{ij} - y)A_{1j}]\left(\frac{\partial A_{2i}}{\partial a_{2i}(t)}\right)}{\sum_{i=1}^{r}\sum_{j=1}^{k} h_{ij}}; \quad (3.39)$$

$$b_{2i}(t+1) = b_{2i}(t) - \frac{\beta\partial E}{\partial b_{2i}(t)} = b_{2i}(t) - \beta\left(\frac{\partial E}{\partial y}\right)\left(\frac{\partial y}{\partial h_{ij}}\right)\left(\frac{\partial h_{ij}}{\partial A_{2i}}\right)\left(\frac{\partial A_{2i}}{\partial b_{2i}(t)}\right) =$$

$$= b_{2i}(t) + \frac{\beta(y^{\star} - y)[\sum_{j=1}^{k}(y_{ij} - y)A_{1j}]\left(\frac{\partial A_{2i}}{\partial b_{2i}(t)}\right)}{\sum_{i=1}^{r}\sum_{j=1}^{k} h_{ij}}; \quad (3.40)$$

$$y_{ij}(t+1) = y_{ij}(t) - \frac{\gamma\partial E}{\partial y_{ij}(t)} = y_{ij}(t) - \gamma\left(\frac{\partial E}{\partial y}\right)\left(\frac{\partial y}{\partial y_{ij}(t)}\right) = y_{ij}(t) + \frac{\gamma(y_{ij} - y)\, h_{ij}}{\sum_{i=1}^{r}\sum_{j=1}^{k} h_{ij}}, \quad (3.41)$$

де α, β, γ — швидкість навчання, а t — означає ітерацію в процесі навчання.

3.7. Нечіткі нейронні мережі *TSK* та Ванга — Менделя

ННМ *TSK*

Узагальненням нейронної мережі *ANFIS* є нечітка нейронна мережа *TSK* (Takagi — Sugeno — Kang). Узагальнену схему виведення моделі *TSK* за використання M правил і N змінних x_j можна подати у такому вигляді:

R_1: якщо $x_1 \in A_1^{(1)}; x_2 \in A_2^{(1)}, ..., x_n \in A_n^{(1)}$, то $y_1 = p_{10} + \sum_{j=1}^{N} p_{1j}\, x_j$;

R_M: якщо $x_1 \in A_1^{(M)}; x_2 \in A_2^{(M)}, ..., x_n \in A_n^{(M)}$, то $y_1 = p_{M0} + \sum_{j=1}^{N} p_{M_j}\, x_j$;

де $A_i^{(k)}$ — значення лінгвістичної змінної x_i для правила R_k з функцією належності виду

$$\mu_A^{(k)}(x_i) = \frac{1}{1 + \left(\frac{x_i - c_i^{(k)}}{\sigma_i^{(k)}}\right)^{2b_i^{(k)}}}, \quad i = \overline{1, N}; k = \overline{1, M}. \quad (3.42)$$

У нечіткий мережі *TSK* перетин умов правила R_k визначається ФП у вигляді добутку:

$$\mu_A^{(k)}(x) = \prod_{j=1}^{k}\left[\frac{1}{1+\left(\frac{x_j - c_j^{(k)}}{\sigma_j^{(k)}}\right)^{2b_j^{(k)}}}\right]. \tag{3.43}$$

За M правил виведення композиція вихідних результатів мережі визначається за формулою (3.44) (аналогічно висновку Сугено):

$$y(x) = \frac{\sum_{k=1}^{M} w_k y_k(x)}{\sum_{k=1}^{M} w_k}, \tag{3.44}$$

де $y_k(x) = p_{k0} + \sum_{j=1}^{N} p_{kj} x_j$.

Ваги w_k у виразі (3.44) інтерпретують як ступінь виконання умов правила $w_k = \mu_A^{(k)}(x)$, які задаються формулами (3.43).

Нечітка нейронна мережа *TSK*, яка реалізує виведення згідно з (3.44), задається багатошаровою структурною мережею, наведеною на рис. 3.11. У такій мережі виділяють п'ять шарів.

1. Перший шар виконує окрему фазифікацію кожної змінної x_i, $i = 1, 2, \ldots, N$, визначаючи для кожного k-го правила висновку значення ФП $\mu_A^{(k)}(x_i)$ відповідно до функції фазифікації, яка застосовується, наприклад (3.42). Це параметричний шар з параметрами $c_j^{(k)}$, $\sigma_j^{(k)}$, $b_j^{(k)}$, які зазнають адаптації в процесі навчання.

2. Другий шар виконує агрегування окремих змінних x_i, визначаючи підсумковий ступінь належності $w_k = \mu_A^{(k)}(x)$ для вектора x з умовою k-го правила. Це непараметричний шар.

3. Третій шар становить собою генератор функцій *TSK*, в якому розраховуються значення $y_k(x) = p_{k0} + \sum_{j=1}^{N} p_{kj} x_j$. У цьому шарі також відбувається перемноження функцій $y_k(x)$ на w_k, сформованих на попередньому шарі. Це параметричний шар, в якому адаптації зазнають лінійні параметри (ваги) p_{k0}, p_{kj} для $j = \overline{1, N}$; $k = \overline{1, M}$, що визначають функції наслідків правил.

4. Четвертий шар складають два нейрони-суматори, один з яких розраховує зважену суму сигналів $y_k(x)$, а другий визначає суму ваг $\sum_{k=1}^{M} w_k$.

5. П'ятий шар складається з єдиного вихідного нейрона. У ньому ваги зазнають нормалізації і обчислюється вихідний сигнал $y(x)$ відповідно до виразу

$$y(x) = \frac{f_1}{f_2} = \frac{\sum_{k=1}^{M} w_k y_k(x)}{\sum_{k=1}^{M} w_k}. \tag{3.45}$$

Це також непараметричний шар.

З наведеного опису можна зазначити, що нечітка мережа *TSK* містить тільки два параметричні шари (перший і третій), параметри яких уточню-

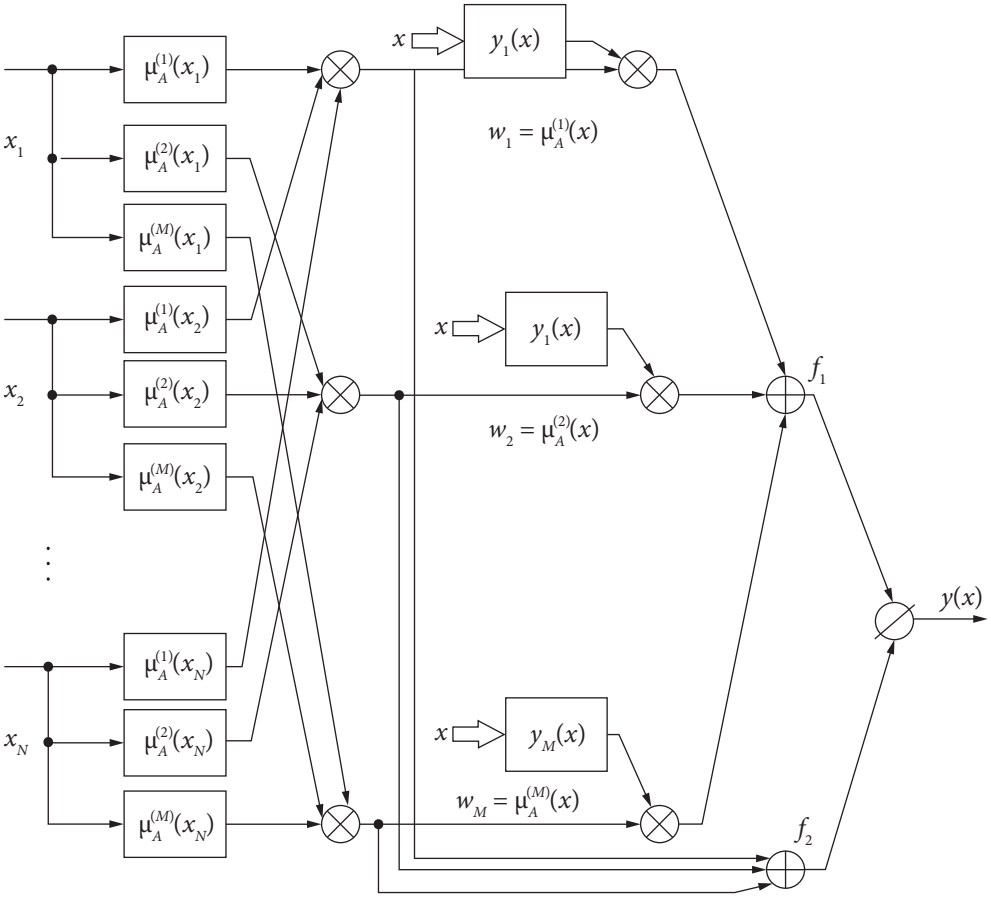


Рис. 3.11. Нечітка нейронна мережа *TSK*

ються в процесі навчання. Параметри першого шару ($c_j^{(k)}$, $\sigma_j^{(k)}$, $b_j^{(k)}$) називатимемо нелінійними, а третього шару $\{p_{kj}\}$ — лінійними. Загальний вираз для функціональної залежності (3.45) для мережі *TSK* задається як

$$y(x) = \frac{1}{\sum_{k=1}^{M}\prod_{j=1}^{M}\mu_A^{(k)}(x_j)}\sum_{k=1}^{M}(p_{k0} + \sum_{j=1}^{N} p_{kj}\, x_j)\prod_{j=1}^{N}\mu_A^{(k)}(x_j). \qquad (3.46)$$

Якщо прийняти, що в конкретний момент часу параметри умов фіксовані, то функція $y(x)$ є лінійною відносно змінних x_j.

За наявності N вхідних змінних кожне правило R_k формує $(N + 1)$ змінну $p_j^{(k)}$ лінійної залежності $y_k(x)$. За M правил виведення маємо $M(N + 1)$ лінійних параметрів мережі. Зі свого боку кожна ФП використовує три параметри, які зазнають адаптації. За M правил виведення отримуємо $M(N + 1)$ лінійних параметрів. Аналогічно кожна ФН використовує три параметри (c, σ, b), які зазнають адаптації. За M правил виведення отри-

муємо $3MN$ нелінійних параметрів. У сумі це дає $M(4N + 1)$ лінійних та нелінійних параметрів, значення яких мають визначатися у процесі навчання. Як правило, це дуже велике число. З метою скорочення кількості параметрів, що мають зазнати адаптації, оперують із меншим числом ФП. Зокрема, можна прийняти, що частина параметрів ФН однієї змінної $\mu_A^{(k)}(x_j)$ фіксується, наприклад $\sigma_j^{(k)}$ і $b_j^{(k)}$.

Структура мережі Ванга — Менделя

Якщо як вихідні функції $y_k(x)$ у правилах виведення вибрати ν_k, то отримаємо структуру ННС, що називається *нечіткою нейронною мережею Ванга — Менделя* (рис. 3.12). Це чотиришарова структура, в якій перший шар виконує фазифікацію вхідних змінних, другий — агрегування (перетин) умов правил, третій (лінійний) — композицію M правил виведення (перший нейрон) і генерацію, що нормалізує сигнал (другий нейрон), тоді як один нейрон останнього шару формує вихідний сигнал $y(x)$, який обчислюють за формулою

$$y(x) = \frac{\sum_{k=1}^{M} \nu_k \left[\prod_{j=1}^{N} \mu_A^{(k)}(x_j)\right]}{\sum_{k=1}^{M} \nu_k}. \qquad (3.47)$$

Відмітимо значну подібність структур обох мереж. Частини, що визначають умови правил, — перший і другий шари — у них ідентичні (тобто відповідають компонентам правил «ЯКЩО …»), розбіжності проявляються в поданні наслідків правил («ТО …»). У мережі *TSK* вихідна функція є поліномом першого порядку, а мережі Ванга — Менделя — константою $\nu_k = c_k$, де величину c_k можна інтерпретувати як центр ФН наслідків. Отже, ННМ Ванга — Менделя є окремим випадком ННМ *TSK*.

Завдання обох мереж полягає у знаходженні такого відображення пар даних (x, d), за якого очікуване значення, що відповідає вхідному вектору x, формувалося вихідною функцією мережі $y(x)$.

Навчання нечітких нейронних мереж, так само, як і чітких, можна здійснювати як відповідно до алгоритму з учителем, за якого використовується цільова функція $E = \frac{1}{2}\sum_{l=1}^{p\Sigma}(y(x)^{(l)} - d^{(l)})^2 \to \min$, так і з алгоритмом самоорганізації без учителя.

Гібридний алгоритм навчання нечітких нейронних мереж

Розглянемо гібридний алгоритм навчання ННМ, який застосовують як для мереж *TSK*, так і для мереж Ванга — Менделя (у яких усі $p_{kj} = 0$ та $p_{k0} = \nu_k$) [3.1, 3.9].

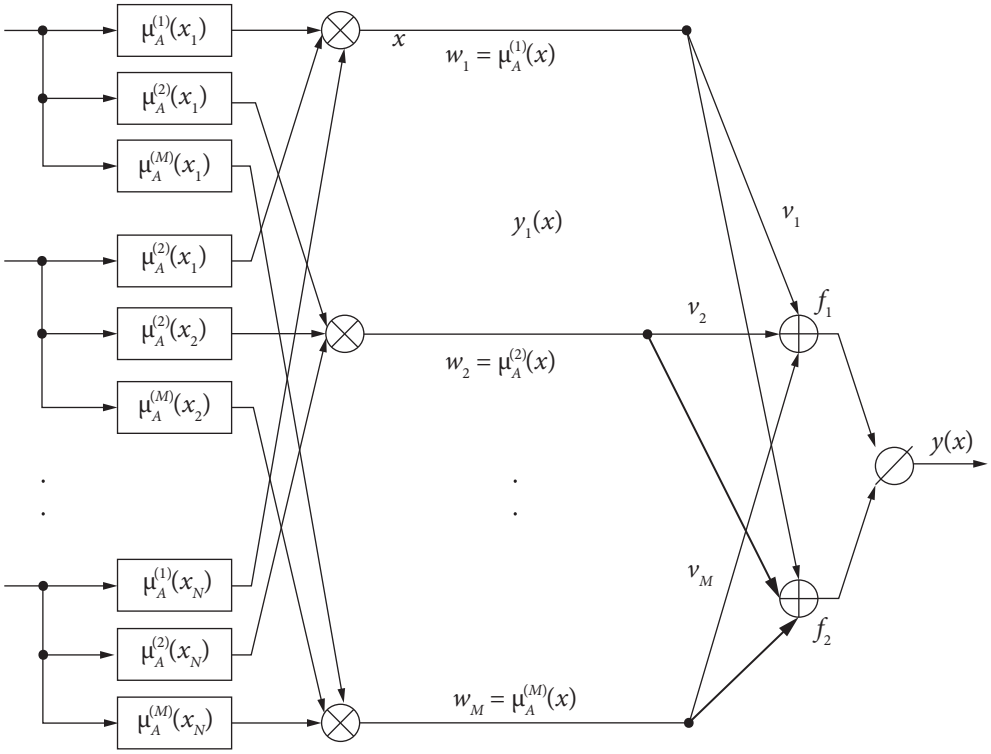


Рис. 3.12. Нечітка нейронна мережа Ванга — Менделя

У гібридному алгоритмі параметри, що зазнаватимуть адаптації, поділяють на дві групи. Перша група складається з лінійних параметрів p_{kj} третього шару, а друга — з параметрів нелінійної ФП першого шару. Уточнення параметрів відбувається у два етапи.

На першому етапі за фіксації окремих значень параметрів функції належності (у першому циклі — це значення, отримані ініціалізацією), розв'язуючи систему лінійних рівнянь, розраховують лінійні параметри p_{kj} полінома *TSK*. За відомих значень ФП залежність для виходу можна подати у вигляді лінійної форми щодо параметрів p_{kj}:

$$y_k(x) = \Sigma_{k=1}^{M} w'_k (p_{k0} + \Sigma_{j=1}^{N} p_{kj} x_j), \tag{3.48}$$

де

$$w'_k = \frac{\Pi_{j=1}^{N} \mu_A^{(k)}(x_j)}{\Sigma_{r=1}^{M} \Pi_{j=1}^{N} \mu_A^{(r)}(x_j)}, \; k = \overline{1, M}. \tag{3.49}$$

За розміру навчальної вибірки L $(x^{(l)}, d^{(l)})$, $(l = 1, 2, \ldots, L)$ і заміни вихідного сигналу мережі очікуваним значенням $d^{(l)}$ отримуємо систему з L

лінійних рівнянь виду

$$\begin{bmatrix} w'_{11} & w'_{11}x_1^{(1)} & \dots & w'_{11}x_N^{(1)} & \dots & w'_{1M} & w'_{1M}x_1^{(1)} & \dots & w'_{1M}x_N^{(1)} \\ w'_{21} & w'_{21}x_1^{(2)} & \dots & w'_{21}x_N^{(2)} & \dots & w'_{2M} & w'_{2M}x_1^{(2)} & \dots & w'_{2M}x_N^{(2)} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ w'_{L1} & w'_{L1}x_1^{(L)} & \dots & w'_{L1}x_N^{(L)} & \dots & w'_{LM} & w'_{LM}x_1^{(L)} & \dots & w'_{LM}x_N^{(L)} \end{bmatrix} \times \begin{bmatrix} p_{10} \\ p_{11} \\ \dots \\ p_{1N} \\ \dots \\ p_{M0} \\ p_{M1} \\ \dots \\ p_{MN} \end{bmatrix} = \begin{bmatrix} d^{(1)} \\ d^{(2)} \\ \dots \\ d^{(L)} \end{bmatrix}, \quad (3.50)$$

де w'_{li} означає рівень активації (вагів) умов i-го правила при введенні l-го вхідного вектора x^l.

Цей вираз можна записати в матрицевому вигляді:

$$Ap = d.$$

Розмірність матриці A дорівнює $L(N + 1)M$. При цьому кількість рядків L зазвичай буває значно більшою кількістю стовпців $(N + 1)M$. Розв'язання цієї системи рівнянь можна отримати як звичайними методами, так і за один крок, використовуючи псевдоінверсію матриці A [3.2, 3.5]:

$$p = A^+ d,$$

де A^+ — псевдоінверсна матриця.

На другому етапі після фіксації значень лінійних параметрів p_{kj} розраховують фактичні вихідні сигнали $y^{(l)}$, $l = 1, 2, \dots, L$, для чого використовують лінійну залежність

$$y(L) = Ap. \quad (3.51)$$

Після цього розраховують вектор помилки $\varepsilon = (y - d)$ та критерій

$$E = \frac{1}{2} \sum_{l=1}^{L} (y(x^{(l)}) - d^{(l)})^2.$$

Сигнали помилок направляють через мережу в зворотному напрямку за методом *Back Propagation* до першого шару, де можуть бути розраховані компоненти вектора градієнта цільової функції відносно параметрів $c_j^{(k)}$, $\sigma_j^{(k)}$, $b_j^{(k)}$. Після знаходження вектора градієнта прораховують крок спуску градієнтним методом. Відповідні формули навчання (для найпростішого методу градієнтного спуску) набувають вигляду

$$c_j^{(k)}(n + 1) = c_j^{(k)}(n) - \eta_c \frac{\partial E(n)}{\partial c_j^{(k)}}, \quad (3.52)$$

$$\sigma_j^{(k)}(n + 1) = \sigma_j^{(k)}(n) - \eta_\sigma \frac{\partial E(n)}{\partial \sigma_j^{(k)}}, \quad (3.53)$$

$$b_j^{(k)}(n+1) = b_j^{(k)}(n) - \eta_b \frac{\partial E(n)}{\partial b_j^{(k)}}, \tag{3.54}$$

де n — номер ітерації.

Після уточнення нелінійних параметрів знову запускається процес адаптації лінійних параметрів функції *TSK* (перший крок) та нелінійних параметрів (другий крок). Цей цикл триває доти, доки не стабілізуються всі параметри процесу.

Формули (3.52)—(3.54) потребують розрахунку градієнта цільової функції щодо параметрів ФП. Остаточний вигляд цих формул залежить від виду ФП. Наприклад, у разі використання узагальненої дзвіноподібної функції

$$\mu_A(x) = \frac{1}{1 + \left(\frac{x - c}{\sigma}\right)^{2b}} \tag{3.55}$$

відповідні формули градієнта цільової функції для однієї пари даних (x, d) набувають вигляду:

$$\frac{\partial E}{\partial c_j^{(k)}} = (y(x) - d) \sum_{r=1}^{M} (p_{r0} + \sum_{j=1}^{N} p_{rj} x_j) \frac{\partial w'_r}{\partial c_j^{(k)}}, \tag{3.56}$$

$$\frac{\partial E}{\partial \sigma_j^{(k)}} = (y(x) - d) \sum_{r=1}^{M} (p_{r0} + \sum_{j=1}^{N} p_{rj} x_j) \frac{\partial w'_r}{\partial \sigma_j^{(k)}},$$

$$\frac{\partial E}{\partial b_j^{(k)}} = (y(x) - d) \sum_{r=1}^{M} (p_{r0} + \sum_{j=1}^{N} p_{rj} x_j) \frac{\partial w'_r}{\partial b_j^{(k)}}.$$

Похідні $\frac{\partial w'_r}{\partial c_j^{(k)}}$, $\frac{\partial w'_r}{\partial \sigma_j^{(k)}}$, $\frac{\partial w'_r}{\partial b_j^{(k)}}$, отримані на основі залежностей (3.49), набувають вигляду

$$\frac{\partial w'_r}{\partial c_j^{(k)}} = \frac{\delta_{rk} m(x_j) - l(x_j)}{[m(x_j)]^2} \prod_{i=1,\, i \neq j}^{M} \frac{\left[\frac{2b_j^{(k)}}{\sigma_j^{(k)}} \left(\frac{x_j - c_j^{(k)}}{\sigma_j^{(k)}}\right)^{2b_j^{(k)}-1}\right]}{\left[1 + \left(\frac{x_j - c_j^{(k)}}{\sigma_j^{(k)}}\right)^{2b_j^{(k)}}\right]^2} \mu_A^{(k)}(x_i); \tag{3.57}$$

$$\frac{\partial w'_r}{\partial \sigma_j^{(k)}} = \frac{\delta_{rk} m(x_j) - l(x_j)}{[m(x_j)]^2} \prod_{i=1,\, i \neq j}^{M} \frac{\left[\frac{2b_j^{(k)}}{\sigma_j^{(k)}} \left(\frac{x_j - c_j^{(k)}}{\sigma_j^{(k)}}\right)^{2b_j^{(k)}}\right]}{\left[1 + \left(\frac{x_j - c_j^{(k)}}{\sigma_j^{(k)}}\right)^{2b_j^{(k)}}\right]^2} \mu_A^{(k)}(x_i); \tag{3.58}$$

$$\frac{\partial w'_r}{\partial b_j^{(k)}} = \frac{\delta_{rk} m(x_j) - l(x_j)}{[m(x_j)]^2} \times$$

$$\times \prod_{i=1,\, i\neq j}^{M} \frac{\left[-2\left(\frac{x_j - c_j^{(k)}}{\sigma_j^{(k)}}\right)^{2b_j^{(k)}} \ln\left(\frac{x_j - c_j^{(k)}}{\sigma_j^{(k)}}\right)\right]}{\left[1 + \left(\frac{x_j - c_j^{(k)}}{\sigma_j^{(k)}}\right)^{2b_j^{(k)}}\right]^2} \mu_A^{(k)}(x_i), \qquad (3.59)$$

де $r = 1, 2, \ldots, M$, δ_{rk} — дельта Кронекера, $l(x_j) = \prod_{j=1}^{N} \mu_A^{(k)}(x_j)$; $m(x_j) = \prod_{j=1}^{N} \mu_A^{(k)}(x_j)$.

Під час практичної реалізації гібридного методу навчання нечітких мереж домінантним фактором їх адаптації вважається перший крок, на якому ваги p_{kj} знаходять з використанням псевдоінверсії за один крок. Для врівноваження його впливу другий етап повторюється багато разів у кожному циклі.

Наведений гібридний алгоритм — один з найефективніших способів навчання нечітких нейронних мереж. Його характерною ознакою є поділ процесу на два відокремлені в часі етапи. Якщо враховувати, що обчислювальна складність кожного алгоритму оптимізації нелінійно залежить від кількості параметрів, що підлягають оптимізації, то зменшення розмірності задач оптимізації значно скорочує обсяг розрахункових операцій і підвищує швидкість збіжності алгоритму. Завдяки цьому гібридний алгоритм більш ефективний порівняно зі звичайним градієнтним методом.

Порівняльний аналіз нечітких нейромереж *ANFIS* та *TSK* у завданнях прогнозування у фінансовій сфері

Опис експериментів. Навчальна вибірка складалася з 50 значень котирувань індексу РТС. Вхід мережі формувався в такий спосіб (на прикладі мережі з двома входами x_1 та x_2 з прогнозованим значенням x_3) [3.1, 3.2]:

x_p	x_{p+1}	x_{p+2}
⋮	⋮	⋮
x_1	x_2	x_3

Розглядалися мережі *ANFIS* і *TSK* з 2—4 входами, сформованими за цією схемою та пронормованими від 0 до 1. Якість оцінювали передусім за критеріями *MSE* та *MAPE*.

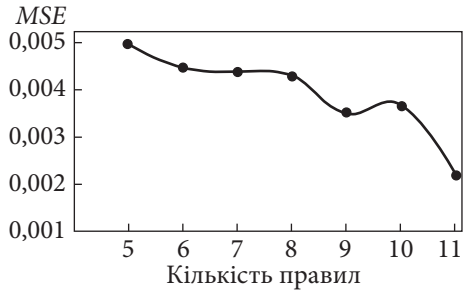


Рис. 3.13. Графік залежності середньоквадратичної похибки від кількості правил для мережі *TSK*

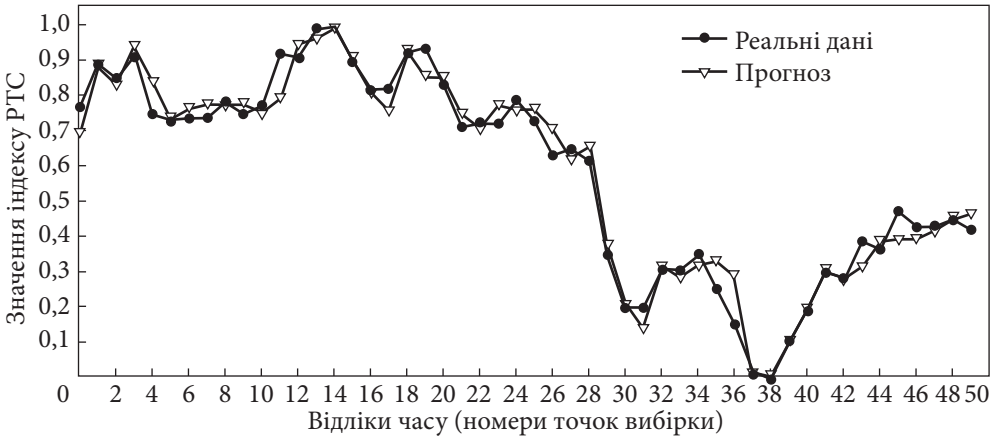


Рис. 3.14. Результати симуляції та прогнозування для мережі *TSK*

Мережі з двома вхідними змінними. Графік залежності середньоквадратичної похибки від кількості правил для мережі *TSK* наведено на рис. 3.13. Як видно, *MSE* зменшується зі збільшенням кількості правил. Проте мережі з більш ніж 11 правилами стають нестабільними у процесі навчання градієнтним методом. За такого навчання найкращий результат серед мереж *TSK* з двома вхідними змінними показала мережа з 11 правилами. Результати симуляції та прогнозування для мережі *TSK* наведено на рис. 3.14.

Значення критеріїв: *MSE* = 0,002201205; *MAE* = 0,0337978497; *MAPE* = = 0,0794041670.

Графік залежності середньоквадратичної похибки від кількості правил для мережі *ANFIS* наведено на рис. 3.15. Як видно з графіків, найкращий результат (мінімальна похибка) було отримано за семи правил, далі процес став розбіжним. Результати прогнозування для найкращої мережі *ANFIS* з сімома правилами показано на рис. 3.16.

Значення критеріїв: *MSE* = 0,025036; *MAE* = 0,134271; *MAPE* = 0,561735.

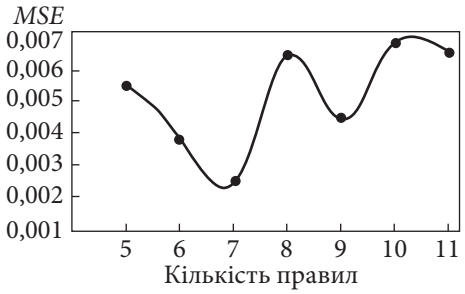


Рис. 3.15. Графік залежності середньоквадратичної похибки для мережі *ANFIS* від кількості правил

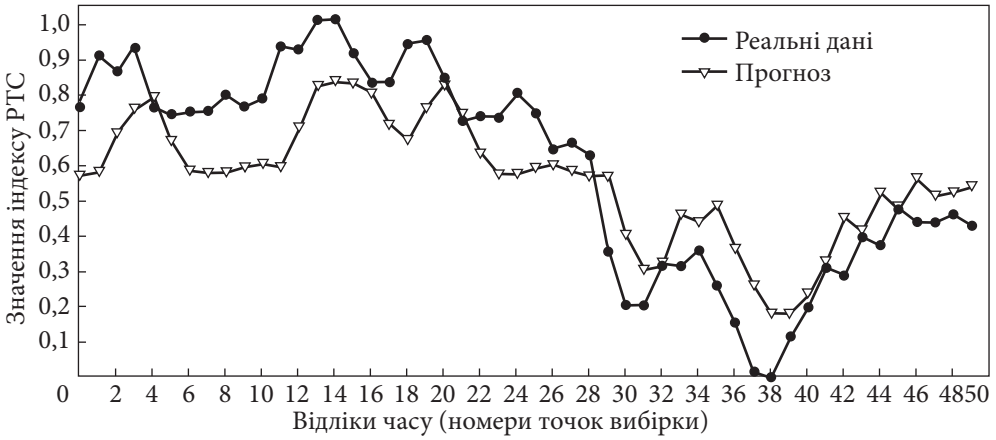


Рис. 3.16. Результати прогнозування для найкращої мережі *ANFIS* з 7 правилами

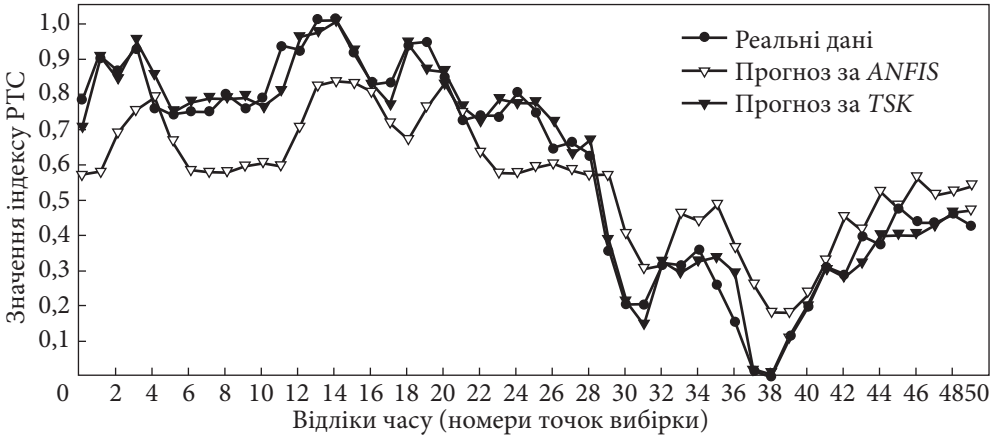


Рис. 3.17. Графіки прогнозування для мереж *TSK* і *ANFIS*

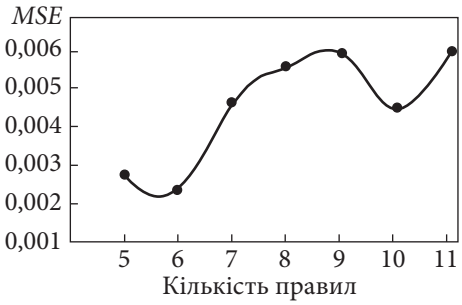


Рис. 3.18. Графік залежності середньоквадратичної похибки від кількості правил для мережі *TSK* з трьома входами

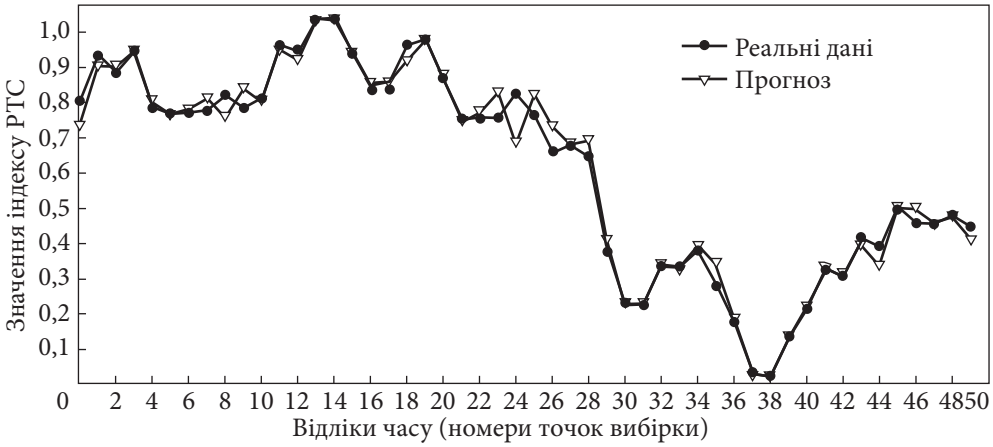


Рис. 3.19. Результати прогнозування мережі *TSK*

Для порівняння отриманих результатів два графіки прогнозування для мереж *TSK* і *ANFIS* наведено разом на рис. 3.17. Найменшу похибку прогнозування на один крок уперед серед мереж з двома входами показала мережа *TSK* з 11 правилами.

Мережі з трьома вхідними змінними. Графік залежності середньоквадратичної похибки від кількості правил для мережі *TSK* з трьома входами наведено на рис. 3.18. Найкращий результат було отримано при 6 правилах. Відповідний результат прогнозування мережі *TSK* наведено на рис. 3.19.

Значення критеріїв: *MSE* = 0,001241; *MAE* = 0,021809; *MAPE* = 0,051053.

Графік залежності середньоквадратичної похибки для мережі *ANFIS* від кількості правил показано на рис. 3.20. Найкращий результат (мінімальну похибку) отримано за 6 правил. Результати прогнозування для мережі *ANFIS* при шести правилах наведено на рис. 3.21.

Значення критеріїв: *MSE* = 0,023477; *MAE* = 0,1282; *MAPE* = 0,495862.

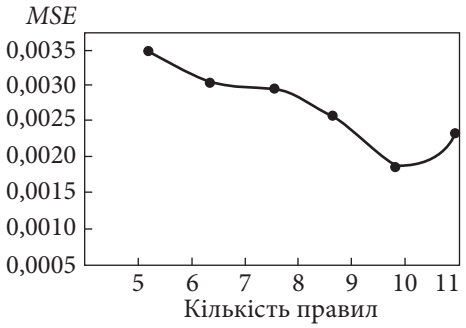


Рис. 3.20. Графік залежності середньоквадратичної похибки від кількості правил для мережі *ANFIS*

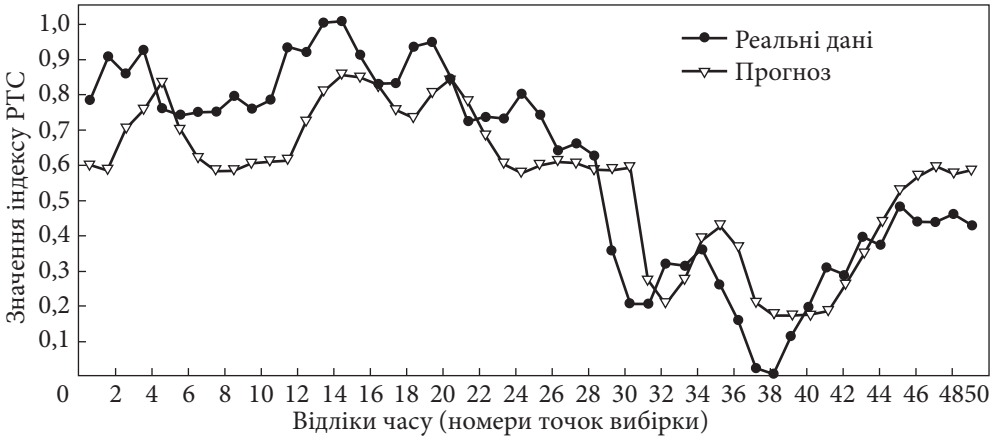


Рис. 3.21. Результати прогнозування для мережі *ANFIS* при 6 правилах

Найменшу похибку прогнозування на один крок уперед серед мереж з трьома входами знову показала мережа *TSK* з 10 правилами (*MSE* = 0,0012).

Мережі з чотирма вхідними змінними. Графік залежності середньоквадратичної похибки від кількості правил для мережі *TSK* з чотирма входами наведено на рис. 3.22.

Значення критеріїв: *MSE* = 0,00192; *MAE* = 0,03232; *MAPE* = 0,075162.

Результати прогнозування для кращої мережі *TSK* з чотирма входами показано на рис. 3.23.

Графік залежності середньоквадратичної похибки від кількості правил для мережі *ANFIS* із чотирма входами наведено на рис. 3.24, а результати прогнозування мереж з чотирма входами — на рис. 3.25.

Отже, найменшу похибку прогнозування на один крок серед мереж із чотирма входами продемонструвала мережа *TSK* з 7 правилами (*MSE* =

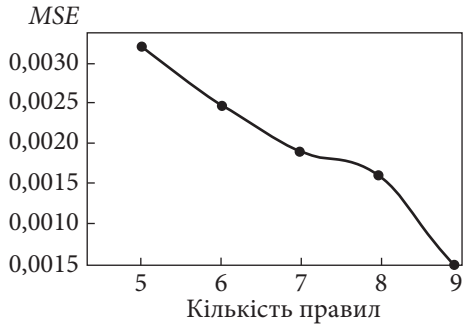


Рис. 3.22. Графік залежності середньоквадратичної похибки від кількості правил для мережі *TSK* з чотирма входами

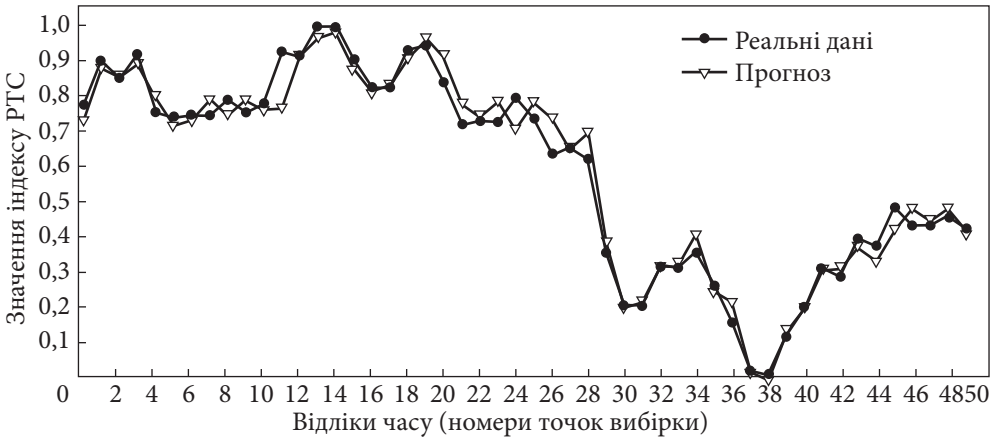


Рис. 3.23. Результати прогнозування для кращої мережі *TSK* з чотирма входами

Рис. 3.24. Графік залежності середньоквадратичної похибки від кількості правил для мережі *ANFIS* із чотирма входами

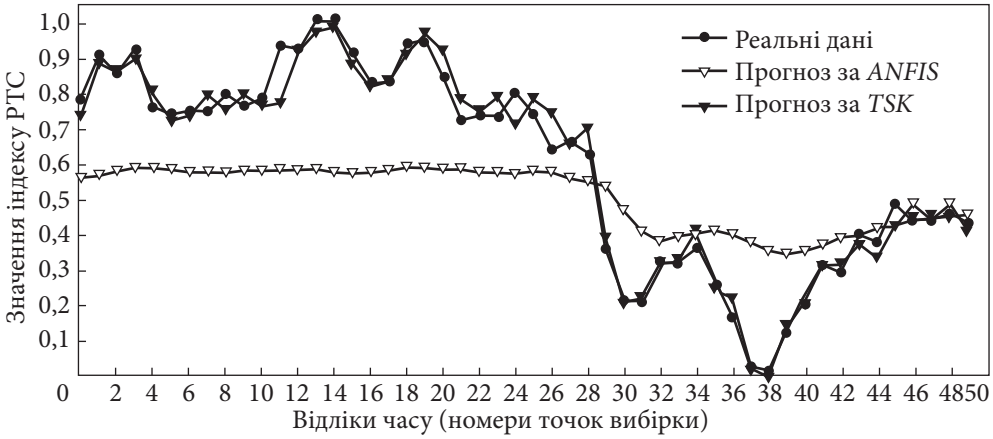


Рис. 3.25. Результати прогнозування мереж *ANFIS* та *TSK* з чотирма входами

= 0,0019). Загалом порівняльний аналіз результатів прогнозування ННМ *ANFIS* і *TSK* свідчить про значні переваги ННМ *TSK*. Її точність прогнозування показників у фінансовій сфері в кілька разів вища порівняно з мережею *ANFIS*.

3.8. Каскадна нечітка (неофаззі) нейронна мережа

Основним недоліком алгоритмів навчання ННМ *ANFIS* і *TSK* є необхідність адаптації як параметрів функцій належності, так і ваг правил. Це потребує значних обчислювальних витрат і часу на навчання. Для подолання цих проблем і зменшення кількості параметрів оптимізації було запропоновано інший клас нечітких нейромереж — каскадні нечіткі (неофаззі) нейронні мережі (*CNFNN*), які були вперше розроблені в роботах Ямакави та його співробітників [3.29, 3.30].

Структурним блоком таких мереж є неонейрон. Розглянемо неофаззі-нейрон з кількома входами та єдиним виходом, зображений на рис. 3.26 [3.29, 3.30]. Цей нейрон реалізує функцію відображення:

$$\hat{y} = \Sigma_{i=1}^{n} f_i(x_i), \tag{3.60}$$

де x_i — i-й вхід ($i = 1, 2, \ldots, n$), $\hat{y}$ — вихід системи.

Структурні блоки неофаззі-нейрона є нелінійним синапсом *NSi*, який переводить i-й вхідний сигнал у форму

$$f_i(x_i) = \Sigma_{j=1}^{h} w_{ji}\, \mu_{ji}(x_i) \tag{3.61}$$

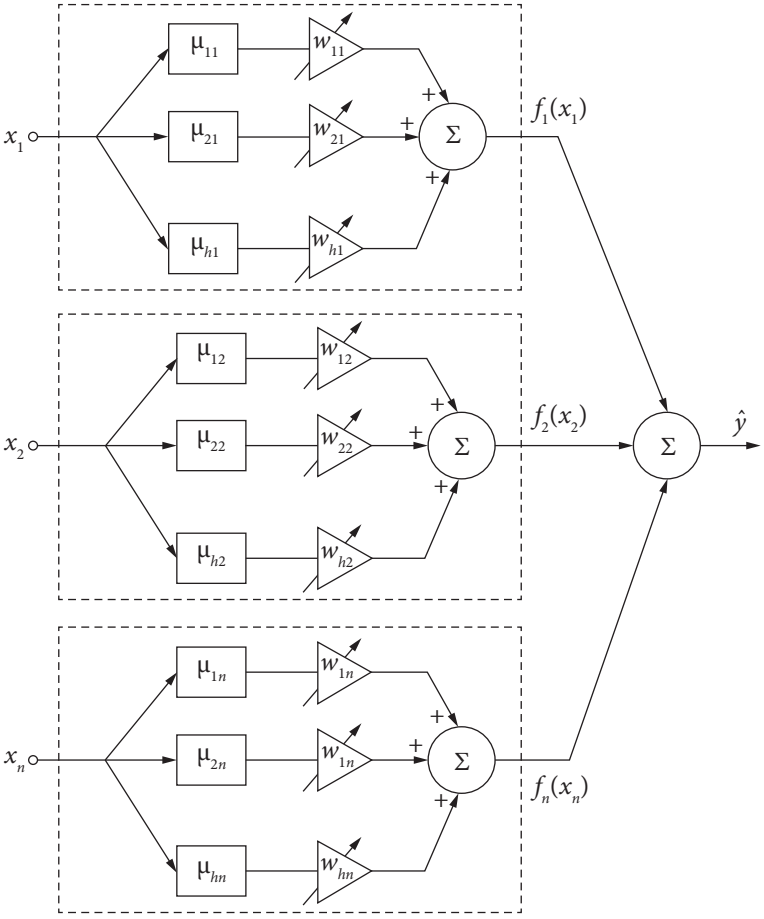


Рис. 3.26. Неофаззі-нейрон з кількома входами та єдиним виходом

і виконує нечіткий висновок:

якщо $x_i \in x_{ji}$, то вихід є w_{ji},

де x_{ji} — нечітке число, функція належності якого μ_{ji}; w_{ji} — синаптична вага. Очевидно, що нелінійний синапс фактично реалізує нечіткий висновок Такагі — Сугено нульового порядку.

Коли векторний сигнал $x(k) = (x_1(k), x_2(k), \ldots, x_n(k))^T$ ($k = 1...$), дискретний час) подається на вхід неофаззі-нейрона, вихід цього нейрона визначається усіма функціями належності $\mu_{ji}(x_i(k))$ і налаштованими синаптичними вагами $w_{ji}(k-1)$, які було отримано в попередній ітерації навчання:

$$\hat{y}(k) = \sum_{i=1}^{n} f_i(x_i(k)) = \sum_{i=1}^{n} \sum_{j=1}^{h} w_{ji}(k-1)\, \mu_{ji}(x_i(k)), \qquad (3.62)$$

і так само неофаззі-нейрон містить hn синаптичні ваги, які потрібно визначити.

Архітектура каскадної нечіткої нейронної мережі

Архітектуру каскадної неофаззі-нейронної мережі *CNFNN* зображено на рис. 3.27, а її модель має такий вигляд [3.29]:

• неофаззі-нейрон першого каскаду

$$\hat{y}^{[1]} = \sum_{i=1}^{n} \sum_{j=1}^{h} w_{ji}^{[1]} \mu_{ji}(x_i); \tag{3.63}$$

• неофаззі-нейрон другого каскаду

$$\hat{y}^{[2]} = \sum_{i=1}^{n} \sum_{j=1}^{h} w_{ji}^{[2]} \mu_{ji}(x_i) + \sum_{j=1}^{h} w_{j,n+1}^{[2]} \mu_{j,n+1}(\hat{y}^{[1]}); \tag{3.64}$$

• неофаззі-нейрон третього каскаду

$$\hat{y}^{[3]} = \sum_{i=1}^{n} \sum_{j=1}^{h} w_{ji}^{[3]} \mu_{ji}(x_i) + \sum_{j=1}^{h} w_{j,n+1}^{[3]} \mu_{j,n+1}(\hat{y}^{[1]}) + \sum_{j=1}^{h} w_{j,n+2}^{[3]} \mu_{j,n+2}(\hat{y}^{[2]});$$

• неофаззі-нейрон m-го каскаду

$$\hat{y}^{[m]} = \sum_{i=1}^{n} \sum_{j=1}^{h} w_{ji}^{[m]} \mu_{ji}(x_i) + \sum_{l=n+1}^{n+m-1} \sum_{j=1}^{h} w_{j,n+1}^{[m]} \mu_{j,l}(\hat{y}^{[l-n]}). \tag{3.65}$$

Відповідно, каскадна неофаззі-нейронна мережа містить $h(n + \sum_{l}^{m-1} l)$ налаштовуваних параметрів, і, що важливо, всі вони лінійно внесені в опис моделі.

Нехай $h(n + m - 1) \times 1$ — вектор функцій належності m-го неофаззі-нейрона має вигляд

$$\mu^{[m]} = (\mu_{11}(x_1), \ldots, \mu_{hl}(x_1), \mu_{12}(x_2), \ldots, \mu_{h2}(x_2), \ldots, \mu_{ji}(x_i), \ldots,$$

$$\mu_{hn}(x_n), \mu_{1,n+1}(\hat{y}^{[1]}), \ldots, \mu_{h,n+1}(\hat{y}^{[1]}), \ldots, \mu_{h,n+m-1}(\hat{y}^{m-1}))^T.$$

Відповідний вектор синаптичних ваг

$$w^{[m]} = (w_{11}^{[m]}, w_{21}^{[m]}, \ldots, w_{h1}^{[m]}, w_{12}^{[m]}, \ldots, w_{h2}^{[m]}, \ldots, w_{ji}^{[m]}, \ldots, w_{hn}^{[m]}, w_{1,n+1}^{[m]}, \ldots, w_{h,n+1}^{[m]}, \ldots, w_{h,m+n-1}^{[m]})^{\mathrm{T}}$$

має таку саму розмірність. Тоді вираз (3.64) можна подати у векторному вигляді:

$$\hat{y}^{[m]} = w^{[m]T} \mu^{[m]}.$$

Навчання каскадної неофаззі-нейронної мережі може бути виконано як у пакетному режимі, так і в режимі послідовного оброблення інформації (адаптивні налаштування ваг).

По-перше, нехай розглядається ситуація, коли навчальна вибірка визначена апріорно, тоді ми отримуємо вибірку значень $x(1), y(1); x(2), y(2); \ldots; x(k), y(k); \ldots; x(N), y(N)$. Для неофаззі-нейрона першого каскаду *NFN* вибірку значень функцій належності $\mu^{[1]}(1), \mu^{[1]}(2), \ldots, \mu^{[1]}(k), \ldots, \mu^{[1]}(N)$ (вектор

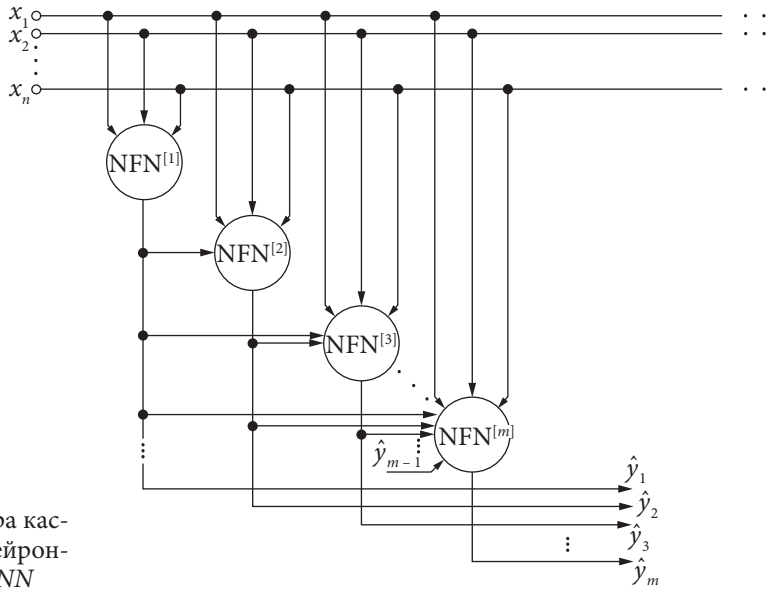


Рис. 3.27. Структура каскадної неофаззі-нейронної мережі *CNFNN*

розміру *hn*) визначають так:

$$\mu^{[1]}(k) = (\mu_{11}(x_1(k)), \ldots, \mu_{h1}(x_1(k)), \mu_{12}(x_2(k)), \ldots, \mu_{h2}(x_2(k)), \ldots, \mu_{ji}(x_i(k)), \ldots, \mu_{hn}(x_n(k)))^T.$$

Шукатимемо його мінімум з використанням критерію навчання *MSE*:

$$E_N^{[1]} = \frac{1}{2}\sum_{k=1}^{N} e^{[1]}(k))^2 = \frac{1}{2}\sum_{k=1}^{N}(y(k) - \hat{y}^{[1]}(k))^2.$$

Використовуючи МНК у режимі пакетної обробки даних, вектор синаптичних ваг можна знайти так:

$$w^{[1]}(N) = (\sum_{k=1}^{N}(\mu^{[1]}(k)\mu^{[1]T}(k))^{+}\sum_{k=1}^{N}(\mu^{[1]}(k)y(k) = P^{[1]}(N)\sum_{k=1}^{N}(\mu^{[1]}(k)y(k), \quad (3.66)$$

де $(\bullet)^{+}$ означає псевдоінверсію Мура — Пенроуза.

За послідовного оброблення даних використовують рекурентний метод найменших квадратів [3.30]:

$$w^{[1]}(k+1) = w^{[1]}(k) + \frac{P^{[1]}(k)(y(k+1) - w^{[1]T}(k)\mu^{[1]}(k+1))}{1 + \mu^{[1]T}(k+1)P^{[1]}(k)\mu^{[1]}(k+1)}\mu^{[1]}(k+1);$$

$$P^{[1]}(k+1) = P^{[1]}(k) + \frac{P^{[1]}(k)(\mu^{[1]}(k+1)\mu^{[1]T}(k+1)P^{[1]}(k))}{1 + \mu^{[1]T}(k+1)P^{[1]}(k)\mu^{[1]}(k+1)}, P^{[1]}(0) = \beta I, \quad (3.67)$$

де β — велике додатне число; I — одинична матриця відповідного розміру.

Можна використовувати алгоритми адаптації (3.67) або (3.66), що забезпечує спрощення обчислювальної складності процесу навчання. У будь-якому разі

використання процедур (3.67), (3.66) суттєво скорочують час навчання порівняно з градієнтним методом, що покладено в основу алгоритму *Back Propagation.*

Після навчання першого каскаду синаптичні ваги неофаззі-нейрона *NFN* [1] стають «замороженими», всі значення $\hat{y}^{[1]}(1)$, $\hat{y}^{[1]}(2)$, …, $\hat{y}^{[1]}(k)$, …, $\hat{y}^{[1]}(N)$ оцінені, і переходять до другого каскаду мережі, який складається з єдиного неофаззі-нейрона *NFN* [2]. Він має один додатковий вхід для сигналу виходу першого каскаду. Потім знову використовують процедуру (3.66) для налаштування вектора вагових коефіцієнтів $w^{[2]}$, розмірність якого $h(n + 1) \times 1$.

За онлайн-методу нейрони навчаються послідовно, виходячи з вхідних сигналів $x(k)$. Нехай вже оцінено синаптичні ваги $w^{[1]}(k)$ і отримано вектор виходів $\hat{y}^{[1]}(k)$, потім, використовуючи вектор входів другого каскаду ($x^T(k)$, $\hat{y}^{[1]}(k)$, обчислюються його ваги $w^{[2]}(k)$ і виходи $\hat{y}^{[1]}(k)$. Для цього можна використовувати алгоритми (3.66) і (3.67).

Процес зростання нейронної мережі (збільшення кількості каскадів) триває, доки не отримаємо необхідну точність рішення, а для налаштування вагових коефіцієнтів останнього m-го каскаду застосовують такі алгоритми [3.29]:

• у пакетному режимі

$$w^{[m]}(N) = (\Sigma_{k=1}^{N} (\mu^{[m]}(k)\, \mu^{[m]T}(k))^{+} \Sigma_{k=1}^{N} \mu^{[m]}(k)\, y(k) = P^{[m]}(N) \Sigma_{k=1}^{N} \mu^{[m]}(k)\, y(k);$$

• у послідовному режимі

$$w^{[m]}(k+1) = w^{[m]}(k) + \frac{P^{[m]}(k)(y(k+1) - w^{[m]T}(k)\mu^{[m]}(k+1))}{1 + \mu^{[1]T}(k+1)P^{[1]}(k)\mu^{[1]}(k+1)}\, \mu^{[m]}(k+1);$$

$$P^{[m]}(k+1) = P^{[m]}(k) + \frac{P^{[m]}(k)(\mu^{[m]}(k+1)\mu^{[m]T}(k+1)P^{[m]}(k))}{1 + \mu^{[m]T}(k+1)P^{[m]}(k)\mu^{[m]}(k+1)},\ P^{[m]}(0) = \beta I, \quad (3.68)$$

або

$$w^{[m]}(k+1) = w^{[m]}(k) + (r^{[m]-1}(y(k+1) - w^{[m]T}(k)\mu^{[m]}(k+1))\, \mu^{[m]}(k+1);$$

$$r^{[m]}(k+1) = ar^{[m]}(k) + ||\mu^{[m]}(k+1)||^2,\ 0 \le \alpha \le 1. \quad (3.69)$$

Отже, запропонована *CNFNN* значно перевершує каскадну архітектуру в швидкості навчання і може навчатися як у пакетному, так і в послідовному (адаптивному) режимах. Лінгвістична інтерпретація отриманих результатів значно розширює функціональні засоби каскадної неофаззі-нейронної мережі.

Результати моделювання. Дослідимо прогнозування числового ряду для оцінки роботи запропонованої архітектури. Застосуємо каскадну неофаззі-нейронну мережу *CNFNN* для прогнозування хаотичного процесу Макі — Гласса, вираженого рівнянням

$$y'(t) = \frac{0{,}2x(t-\tau)}{1 + y^{10}(t-\tau)} - 0{,}1y(t). \quad (3.70)$$

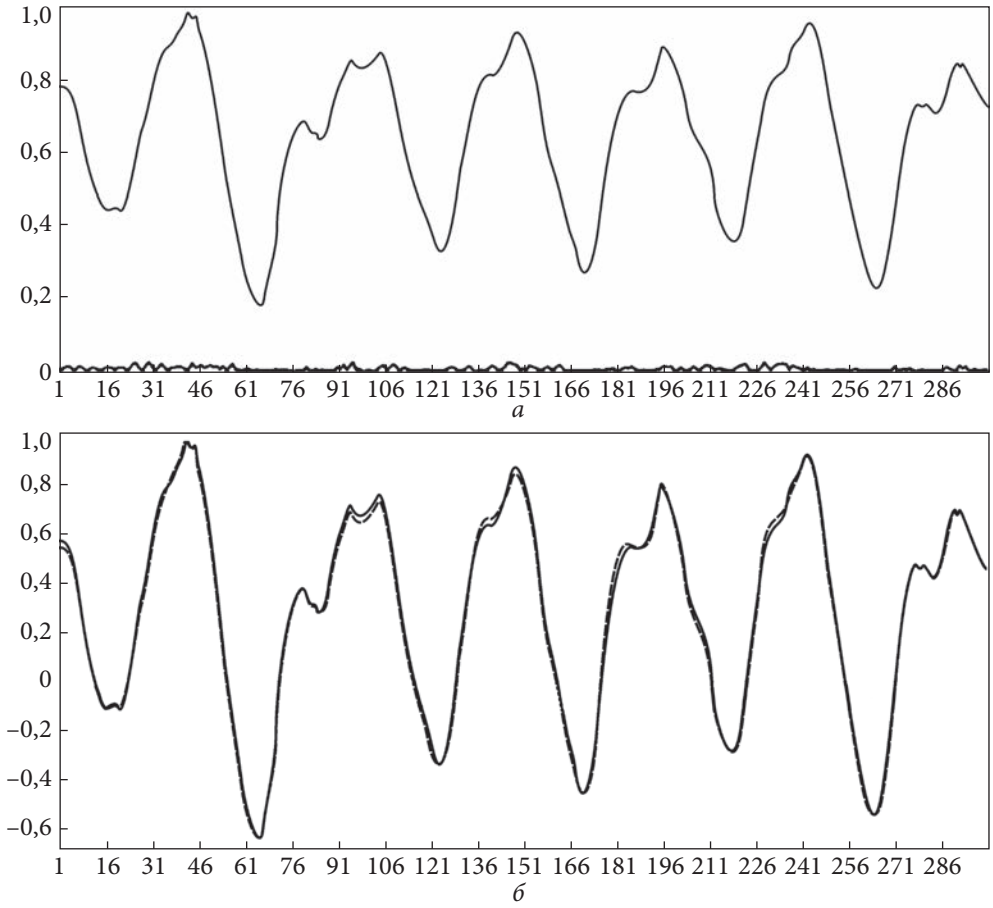


Рис. 3.28. Експериментальні результати прогнозування часового ряду та його похибки: *а* — виходи мережі; *б* — спрогнозований і реальний часові ряди

Сигнал $y(t)$, визначений за (3.70), був дискретизований з кроком 0,1. Візьмемо фрагмент, що містить 500 вимірів для навчальної вибірки. Необхідно передбачити значення часового ряду на шість кроків уперед, використовуючи його значення у моменти часу k, $(k – 6)$, $(k – 12)$, $(k – 18)$. Тестова вибірка містить 9500 вимірів, тобто значення часового ряду в часовому інтервалі від 501 до 1000. Для оцінки отриманих результатів було використано нормалізовану середньоквадратичну похибку:

$$\mathrm{NRMSE}(k; N) = \frac{\sum_{q=1}^{N} e^2(k+q)}{\mathrm{N}\sigma},$$

де σ — середньоквадратичне відхилення прогнозу на навчальній вибірці.

У процесі моделювання використовували мережу *CNFNN* із п'ятьма каскадами. Перший каскад містив чотири нелінійні синапси для кожного

вхідного значення часового ряду і кожний синапс утримував 10 функцій активації (тобто функцій належності). Вхідні сигнали були представлені в інтервалі [0, 1]. Отримані експериментальні результати прогнозування часового ряду та його похибки показано на рис. 3.28. Після завершення навчання мережі *CNFNN* похибка досліджуваної вибірки становила $4 \cdot 10^{-4}$ [3.28].

Застосування методу МГУА для синтезу оптимальної структури каскадної неофаззі-нейронної мережі. Важливим завданням, яке доводиться вирішувати у разі використання каскадних нейронних неофаззі-мереж, є вибір кількості каскадів і синтезу структури цієї мережі. Таке завдання доцільно вирішувати з використанням методу евристичної самоорганізації МГУА [3.5, 3.9]. Розглянемо цей метод і наведемо відповідні результати експериментальних досліджень.

Алгоритм МГУА для синтезу структури каскадної нечіткої нейронної мережі. Якщо використовувати неофаззі-нейрони з двома входами, то можна застосовувати метод МГУА для синтезу оптимальної структури неофаззі-нейронної мережі. Основна ідея алгоритму МГУА полягає в тому, що відбувається послідовне нарощування шарів нейронної мережі, доки зовнішній критерій не почне зростати.

Опис алгоритму синтезу [3.5, 3.30]

1. Формуємо пари із виходів нейронів поточного шару (на першій ітерації використовуємо множину вхідних сигналів). Кожна пара подається на входи неофаззі-нейрона.

2. Використовуючи навчальну вибірку, налаштовуємо вагові коефіцієнти зв'язків кожного неофаззі-нейрона.

3. З використанням перевірної вибірки обчислюємо зовнішній критерій регулярності кожного неофаззі-нейрона:

$$\varepsilon_p^{[s]} = \frac{1}{N_{\text{пр}}} \sum_{i=1}^{N_{\text{пр}}} (y(i) - \hat{y}_p^{[s]}(i))^2, \tag{3.70}$$

де $N_{\text{пр}}$ — розмір перевірної вибірки; s — номер шару; $\hat{y}_p^{[s]}(i)$ — вихідний сигнал нейрона ps-го шару на i-й вхідний сигнал.

4. Знаходимо мінімальне значення зовнішнього критерію для всіх нейронів поточного шару:

$$\varepsilon^{[s]} = \min_p \varepsilon_p^{[s]}.$$

Виконуємо перевірку умови зупинки:

$$\varepsilon^{[s]} > \varepsilon^{[s-1]}, \tag{3.71}$$

де $\varepsilon^{[s]}$, $\varepsilon^{[s-1]}$ — значення критерію (3.70) для кращих нейронів s-го та $(s-1)$-го шарів відповідно.

Якщо умова (3.71) виконується, то повертаємось на попередній шар, знаходимо найкращий нейрон із мінімальним значенням критерію та переходимо на крок 5. Інакше — вибрати F кращих нейронів відповідно до критерію на кроці 4 і перейти на крок 1 для конструювання наступного шару нейронів.

5. Визначаємо остаточну структуру мережі. Для цього рухаємося у зворотному напрямку від найкращого нейрона від його вхідних зв'язків і, послідовно проходячи всі шари нейронів, зберігаємо в кінцевій структурі мережі ті неофаззі-нейрони, які використовуються на наступному шарі.

6. Після завершення роботи алгоритму МГУА буде синтезовано оптимальну структуру мережі. Як неважко помітити, ми отримуємо не тільки оптимальну структуру мережі, а й навчену нейронну мережу, яка готова обробляти нові дані. Одна з важливих переваг використання МГУА для синтезу архітектури каскадної мережі — це її здатність використовувати дуже швидкі навчальні процедури для ваг неофаззі-нейронів, оскільки навчання відбувається за шарами (шар за шаром).

Результати експериментальних досліджень. Проведено експериментальні дослідження неофаззі-нейронної мережі у вирішенні проблеми прогнозування. Мета прогнозу полягала у прогнозуванні індексу РТС (Російська торговельна система) на основі поточних цін акцій провідних компаній [3.30]. Щоденні ціни акцій індексу РТС за період з 5 лютого по 5 травня 2009 р. Вихідна змінна — індекс РТС наступного дня. Розмір навчальної вибірки — 100 значень. Критерії прогнозування — СКВП прогнозу (*MSE*) та середня абсолютна відсоткова похибка прогнозу (*MAPE*).

Типи експериментів з неофаззі-нейронною мережею

1. Варіація співвідношення навчальна / перевірна вибірки в діапазоні: 25 : 75, 50 : 50, 75 : 25.

2. Варіація числа шарів: 1-3-5.

3. Варіація числа ітерацій: 1000, 10 000, 100 000.

4. Варіація числа прогнозованих точок: 1-3-5.

5. Зміна максимальної похибки — умова зупинки: 0,01 та 0,09.

Деякі з експериментальних результатів наведено далі [3.30].

Експеримент 1. Варіація співвідношення навчальної / перевірної вибірки.

Експеримент А. Співвідношення навчальної / перевірної вибірки 75 : 25. Графіки реальних та прогнозних значень наведено на рис. 3.29. *MSE* = 0,050158. На подальших рисунках позначення такі самі.

Експеримент В. Співвідношення навчальної / перевірної вибірки 50 : 50. *MSE* = 0,053562.

Експеримент С. Співвідношення навчальної / перевірної вибірки 25 : 75. Результати наведено на рис. 3.30. *MSE* = 0,068489.

Рис. 3.29. Графіки реальних та прогнозних значень індексу РТС

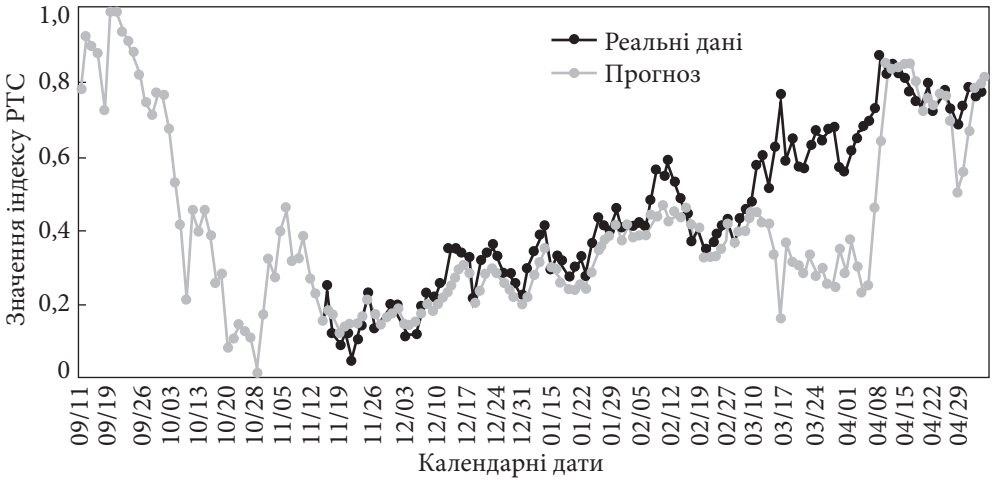


Рис. 3.30. Графіки реальних та прогнозних значень індексу РТС для співвідношення 25 : 75

Експеримент 2. Порівняння роботи алгоритму під час варіації числа шарів: 1-3-5-7. Прогноз на одну точку за співвідношення навчальна / перевірна вибірки 75:25.

Експеримент A. Число шарів 1, *MSE* = 0,04662. Результати прогнозування наведено на рис. 3.31.

Експеримент B. Число шарів 3, *MSE* = 0,255.

Експеримент C. Число шарів 5, *MSE* = 0,0446.

Експеримент D. Число шарів 7, *MSE* = 0,0544.

Рис. 3.31. Графіки реальних та прогнозних значень індексу РТС з одним шаром

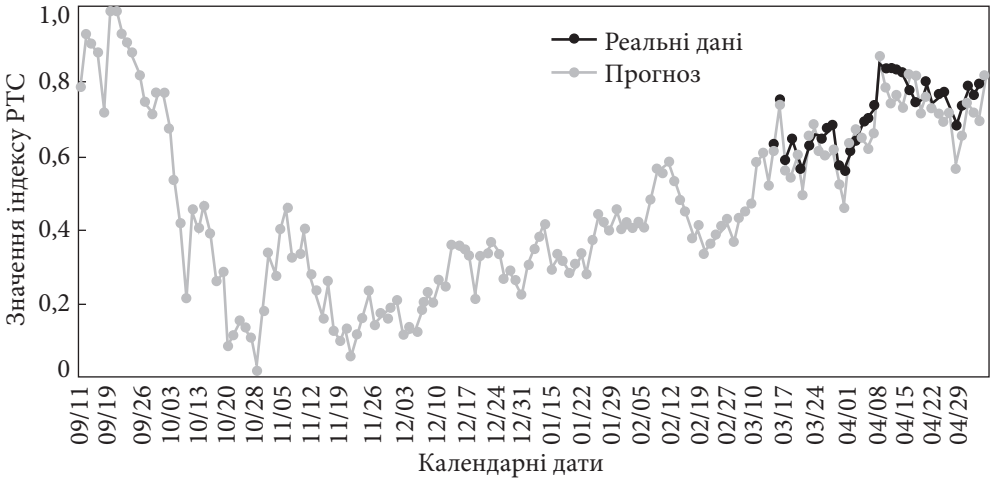


Рис. 3.32. Графіки реальних та прогнозних значень індексу РТС за прогнозування на одну точку

Експеримент 3. Варіація числа ітерацій: 1000, 10 000, 100 000.

Експеримент A. Число ітерацій 1000, *MSE* = 0,0588.

Експеримент B. Число ітерацій 10 000, *MSE* = 0,0575.

Експеримент C. Число ітерацій 100 000, *MSE* = 0,0525.

Експеримент 4. Порівняння точності алгоритму прогнозування у разі варіювання числа прогнозованих точок 1-3-5, за використання співвідношення навчальна / перевірна вибірки 75 : 25.

Експеримент A. Число прогнозованих точок 1. Результати наведено на рис. 3.32. *MSE* = 0,0495.

Експеримент B. Число прогнозованих точок 3, *MSE* = 0,4469.

Експеримент C. Число прогнозованих точок 5, *MSE* = 1,0418.

Повну структуру неофаззі-нейронної мережі наведено на рис. 3.33 (4 входи, 1 вихід (1 передбачувана точка), 1 прихований шар), а оптимальну структуру мережі, синтезованої за МГУА, — на рис. 3.34 (4 входи, 1 вихід (1 прогнозована точка), 1 прихований шар).

Висновки експерементальних досліджень. Після виконання серії експериментів з неофаззі-нейронною мережею з повною структурою та оптимальною структурою, синтезованою з використанням МГУА, було отримано підсумкові результати, наведені в табл. 3.3 [3.30].

У таблиці півгрубим шрифтом відзначено найкращі результати. Як неважко помітити, неофаззі-нейронна мережа, синтезована за допомогою МГУА, дає набагато кращі результати, ніж звичайна мережа з повною структурою. Це можна пояснити використанням механізму самоорганізації в мережі, синтезованій за МГУА. Разом з тим мережа з оптимальною структурою, синтезованою за МГУА, має недолік — повільнішу збіжність. Для зручності аналізу отриманих експериментальних результатів нижче наведено рисунки з графіками відповідних залежностей. На рис. 3.35 зображено залежність похибки *MAPE* від числа точок прогнозування.

Таблиця 3.3

Порівняльний аналіз експериментальних результатів *CNFNN* з повною та оптимальною структурою

Тип серії експериментів	Тип експериментів	Мережа, синтезована МГУА	Повна мережа
Варіація відношення навчальна / тестова вибірка	75 % : 25 %	**0,0484**	0,0501
	50 % : 50 %	0,0532	0,0536
	25 % : 75 %	0,0608	0,0684
Варіація кількості шарів	1	0,0628	0,0626
	3	**0,0381**	0,0544
	5	0,0434	0,0652
Кількість ітерацій	1000	0,0588	0,0674
	10 000	0,0479	0,0485
	100 000	**0,0459**	0,0482
Кількість прогнозованих точок	1	**0,0495**	0,0587
	3	0,4469	1,0844
	5	1,0418	1,3901
Значення максимальної похибки (поріг зупинення)	0,01	**0,0507**	0,0881
	0,09	0,0899	0,0898

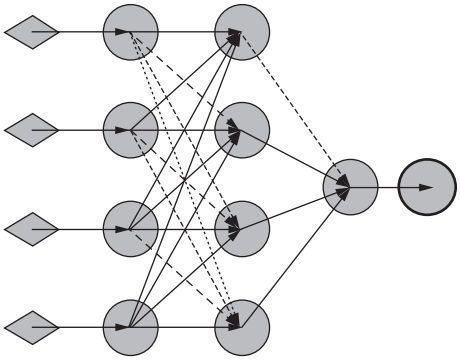


Рис. 3.33. Повна структура неофаззі-нейронної мережі

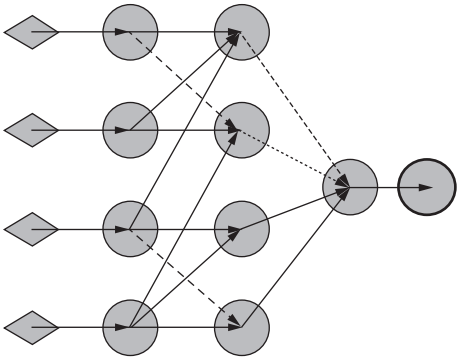


Рис. 3.34. Оптимальна структура мережі, синтезованої за МГУА

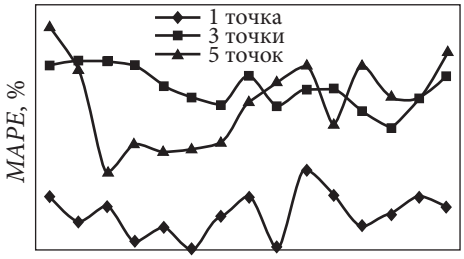


Рис. 3.35. Залежність похибки *MAPE* від числа точок прогнозу

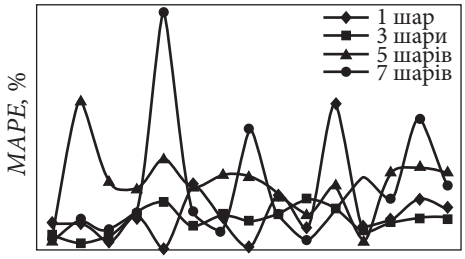


Рис. 3.36. Залежність похибки *MAPE* від числа шарів *CNFNN*

Як можна помітити, у разі прогнозування лише однієї точки отримуємо дуже високу точність: середня відносна похибка становить менш як 15 %. Зі збільшенням числа прогнозованих точок точність падає, похибка перебуває в діапазоні 15—45 %.

Аналізуючи наведені результати на рис. 3.36, на якому показано залежність похибки прогнозування *MAPE* від числа шарів неофаззі-нейронної мережі, можна дійти висновку, що найкращі результати досягаються за кількості шарів, що дорівнює трьом. У цьому разі середня відносна похибка становить менш як 10 %. За одного прихованого шару похибка невисока, але нерівномірно розподілена і може перевищувати 30 %. Для п'яти прихованих шарів похибка зростає і може досягати 35 %. Нарешті, за семи шарів умова *MAPE* сягає 60 %.

Крім того, в ході експериментальних досліджень було визначено оптимальні параметри алгоритмів для повної неофаззі-нейронної мережі:

- найкраще співвідношення навчальної / перевірної вибірки — 75—25 %;
- найкраще число шарів — три;
- найкращі результати навчання за числа ітерацій — 100 000.

3.9. Експериментальні дослідження каскадної нечіткої (неофаззі) мережі

Мета досліджень полягала в аналізі роботи каскадної неофаззі-нейронної мережі для прогнозування поведінки ринку цінних паперів Німеччини, динаміку біржового індексу якого наведено в табл. 3.4 [3.31].

На рис. 3.37 представлено відповідну залежність точності прогнозу від числа каскадів — критерій *MAPE* (%).

Далі було проведено дослідження залежності *MAPE* від числа входів. Відповідні результати наведено на рис. 3.38 для співвідношення $N_{навч} / N_{перев} = 50 / 50$.

Як видно з рис. 3.38, оптимальна кількість входів для цієї задачі дорівнює 4.

Важливим параметром для каскадної неофаззі-мережі є число значень лінгвістичної змінної (нечітких множин). Результати проведених відповідних досліджень наведено на рис. 3.39.

Результати свідчать, що оптимальна кількість значень лінгвістичних змінних дорівнює чотирьом для цієї задачі прогнозування. Далі було проведено дослідження залежності співвідношення $N_{навч} / N_{перев}$ на якість прогнозу. Відповідні результати залежності точності прогнозу від кількості каскадів для різних співвідношень $N_{навч} / N_{перев}$ наведено на рис. 3.40 [3.31].

Знайдені залежності критерію *MAPE* від числа входів для різних співвідношень $N_{навч} / N_{перев}$ подано на рис. 3.41.

Таблиця 3.4

Динаміка біржового індексу ринку ФРН

Місяць	2010	2011	2012	2013	2014	2015	2016
Січень	100	112,503	96,884	123,189	154,842	140,477	127,378
Лютий	91,3225	119,059	107,396	122,1897	155,330	148,933	123,294
Березень	96,7425	116,464	109,99	122,595	154,272	152,447	131,169
Квітень	99,6720	124,913	105,95	120,2768	156,623	154,271	135,750
Травень	89,3813	125,513	98,049	128,8783	159,246	154,696	135,114
Червень	88,7395	123,183	92,7264	127,6049	161,332	150,781	132,484
Липень	92,7467	124,579	96,202	127,7148	157,758	148,275	131,650
Серпень	94,3905	101,466	103,01	132,616	147,500	144,488	141,021
Вересень	97,2455	88,7916	111,93	135,6962	148,519	134,082	140,811
Жовтень	106,911	96,4015	112,95	143,4353	136,024	136,962	139,760
Листопад	109,919	94,3967	111,04	147,9172	141,470	140,838	136,086
Грудень	110,482	92,2427	118,68	151,2206	144,601	138,877	139,783

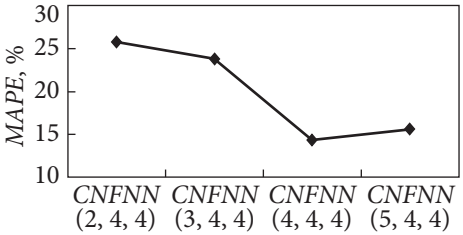


Рис. 3.37. Залежність критерію *MAPE* для $N_{навч}$ / $N_{перев}$ = 50 / 50 для різного числа каскадів

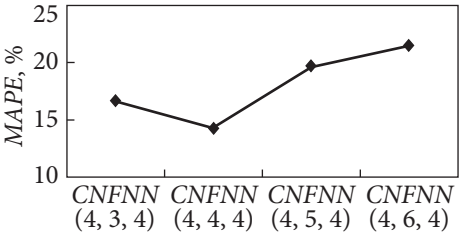


Рис. 3.38. Залежність критерію *MAPE* від числа входів для $N_{навч}$ / $N_{перев}$ = 50 / 50

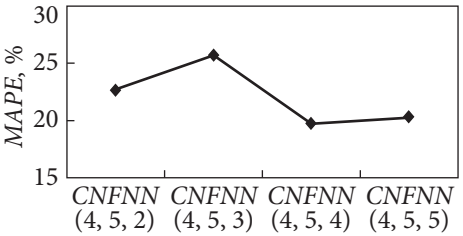


Рис. 3.39. Залежність критерію *MAPE* від числа значень лінгвістичної змінної

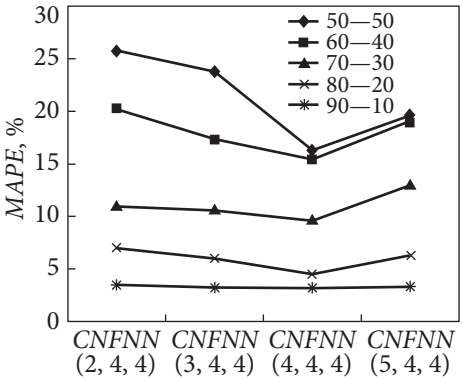


Рис. 3.40. Залежності точності прогнозу від кількості каскадів

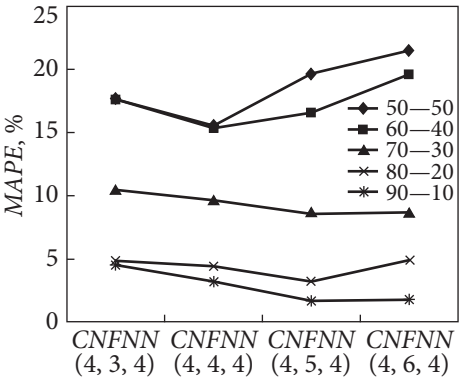


Рис. 3.41. Залежність критерію *MAPE* від числа входів

Таблиця 3.5

Порівняння результатів прогнозування каскадних мереж *CNFNN* та *ANFIS*

MAPE, %	50—50	60—40	70—30	80—20	90—10
ННМ *ANFIS*	19,7	17,65	12,54	6,9554	4,5614
CNFNN (4, 4, 4)	15,5	15,4	9,6584	4,4325	3,1952

Як випливає з наведених результатів, для кожного класу прогнозованих фінансових процесів є значення оптимального числа каскадів. З подальшим їх збільшенням похибка прогнозу або припиняє змінюватися, або починає зростати.

Зі зростанням числа значень лінгвістичних змінних похибка прогнозу спочатку спадає, досягає мінімуму, а потім починає зростати. Така сама залежність простежується і для числа входів моделі та числа значень лінгвістичних змінних.

Далі було виконано аналогічні експерименти щодо прогнозування з використанням ННС *ANFIS.* Число входів та число значень лінгвістичних змінних було обрано таким, що дорівнює 4. Здійснили порівняння ефективності прогнозування з каскадною мережею з аналогічними значеннями параметрів *CNFNN* (4,4,4). Відповідні результати для обох мереж за варіювання співвідношення $N_{навч}$ / $N_{перев}$ наведено в табл. 3.5.

Як показують результати порівняння, каскадна неонечітка нейронна мережа (*CNFNN*) демонструє вищу точність прогнозування. Додатковими перевагами *CNFNN* порівняно з нейромережею *ANFIS* є відсутність необхідності налаштовувати параметри функцій належності, що дає змогу суттєво зменшити обчислювальні витрати та час навчання мережі.

Висновки

Цей розділ присвячено аналізу систем нечіткої логіки та нечітких нейронних мереж (ННМ). Описано основні етапи нечіткого логічного висновку та алгоритми нечіткого логічного висновку: Мамдані, Цукамото, Сугено та Ларсена, а також здійснено аналіз їхніх властивостей.

Сформульовано важливу теорему, яка обґрунтовує універсальні властивості систем з нечіткою логікою — *FAT (Fat Approximation Theorem),* що є аналогом теореми універсальної апроксимації для звичайних нейронних мереж. Проаналізовано основні властивості та недоліки систем нечіткої логіки (СНЛ).

Далі розглянуто класи нечітких нейронних мереж, які об'єднують дві технології: звичайні нейронні мережі та СНЛ. Описано нейронну мережу на основі алгоритмів Мамдані та Цукамото, а також алгоритм її навчання. Наведено структуру та алгоритм навчання відомої мережі *ANFIS,* а також описано ННМ *TSK*, її структуру та гібридний алгоритм навчання, який значно скорочує обчислювальні витрати порівняно зі стохастичним градієнтним спуском.

Проведено порівняльний аналіз ННМ *ANFIS* та *TSK* у задачах прогнозування на ринку цінних паперів. Основним недоліком ННМ *TSK* є необхідність навчання як вхідних параметрів — функцій належності нечітких

множин вхідних змінних, так і вихідних параметрів — ваг правил, що потребує значних обчислювальних витрат і тривалого часу навчання.

Як альтернативу ННМ *TSK* розглянуто неонечіткий нейрон і каскадну неонечітку нейронну мережу на його основі. У цій мережі потрібно навчати лише вихідні параметри правил, що істотно зменшує обчислювальні витрати та час навчання. Ефективність нео-нечіткої нейронної мережі досліджено на завдання прогнозування індексу на ринку цінних паперів ФРН. У результаті експериментів визначено оптимальні параметри мережі, зокрема кількість каскадів, кількість входів і кількість фазифікаторів.

СПИСОК ЛІТЕРАТУРИ

3.1. Russell S., Norvig P. Artificial Intelligence: A Modern Approach. Pearson, 2010. 1152 p.

3.2. Zgurovsky M. Z., Zaychenko Yu.P. Fundamentals of computational intelligence: System approach. Springer, 2016. 275 p.

3.3. Klir G.J., Yuan B. Fuzzy Sets and Fuzzy Logic: Theory and Applications. Prentice Hall, 1995. 592 p.

3.4. Pedrycz W. Fuzzy Control and Fuzzy Systems. Wiley, 1993. 350 p.

3.5. Kosko B. Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence. Prentice Hall, 1992. 447 p.

3.6. Dubois D., Prade H. Fuzzy Sets and Systems: Theory and Applications. Academic Press, 1980. 393 p.

3.7. Haykin S. Neural Networks and Learning Machines. Prentice Hall, 2008. 936 p.

3.8. Huang G.B., Zhou H., Ding X., Zhang R. Extreme Learning Machine for Regression and Multiclass Classification. *IEEE Transactions on Systems, Man, and Cybernetics.* 2012. Vol. 42 (2). P. 513—529.

3.9. Зайченко Ю.П. Основи проектування інтелектуальних систем. Київ: ВД «Слово», 2004. 352 с.

3.10. Dubois D., Prade H. Possibility Theory: An Approach to Computerized Processing of Uncertainty. Springer, 1988. 262 p.

3.11. Mendel J.M. Uncertain Rule-Based Fuzzy Systems: Introduction and New Directions. Springer, 2017. 684 p.

3.12. Dubois D., Prade H. Fuzzy Information and Engineering Systems. Springer, 1996. 299 p.

3.13. Bishop C. Pattern Recognition and Machine Learning. Springer, 2006. 738 p.

3.14. Terano T., Asai K., Sugeno M. Fuzzy Systems Theory and Its Applications. Academic Press, 1992. 288 p.

3.15. Rutkowski L. Computational Intelligence: Methods and Techniques. Springer, 2008. 261 p.

3.16. Jang J.-S. R., Sun C.-T., Mizutani E. Neuro-Fuzzy and Soft Computing. Prentice Hall, 1997. 614 p.

3.17. Cao L. Data Science Thinking: The Next Scientific, Technological and Economic Revolution. Springer, 2018. 350 p.

3.18. Pal S.K., Mitra S. Neuro-Fuzzy Pattern Recognition: Methods in Soft Computing. Wiley, 1999. 400 p.

3.19. Jang J.-S. R., Sun C.-T., Mizutani E. Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence. Prentice Hall, 1997. 614 p.

3.20. Kosko B. Fuzzy Systems as Universal Approximators. *IEEE Trans. Comput.* 1994. No. 11. P. 1329—1333.

3.21. Kosko B. Fuzzy Engineering. New Jersey: Prentice Hall, 1997.

3.22. Zadeh L.A. Fuzzy sets as a basis for a theory of possibility. *Fuzzy sets and Syst.* 1978. No. 1. P. 3—28.

3.23. Zadeh L.A. The Concept of a Linguistic variable and its application to approximate reasoning. *Inform. Sci.* 1975. No. 8. Part 1, 2. P. 199—249, 301—357.

3.24. Zadeh L.A.Theory of commonsense knowledge: aspects of vagueness. Dordrecht: D. Reidel, 1984. P. 257—296.

3.25. Lotfi A. Fuzzy logic, Neural Networks and Soft Computing. *Communications of the ACM* . March 1994. Vol. 37(3). P. 77—84.

3.26. Wang F. Zadeh. Neural Networks Genetic Algorithms and Fuzzy Logic for Forecasting. *Proc. Intern. Conf. Advanced Trading Technologies.* New York, 1992. P. 504—532.

3.27. Nayak J., Dash R., Majhi B. Adaptive Neuro-Fuzzy Inference System: A Survey of Applications in Engineering. *Applied Soft Computing*. 2015. Vol. 27. P. 251—259.

3.28. Bodyanskiy Ye., Viktorov Ye., Pliss I. The cascade NFNN and its learning algorithm. *Вісник Ужгород. нац. ун-ту. Сер. Математика і інформатика.* 2008. Вип. 17. С. 48—58.

3.29. Yamakawa F., Uchino E., Miki T., Kusanagi H. A neo-fuzzy neuron and its applications to system identification and prediction of the system behavior. *Proc. 2nd Intern. Conf. Fuzzy Logic and Neural Networks «LIZUKA-92»*. Lizuka, 1992. P. 477—483.

3.30. Bodyanskiy Ye., Zaychenko Yu., Pavlikovskaya E. et al. Neo-fuzzy neural network structure optimization using GMDH for solving forecasting and classification problems. *Proc. Int. Workshop on Inductive Modeling*. 2009. P. 77—89.

3.31. Atsalakis G.S., Protopapadakis E.E., Valavanis K.P. Stock Trend Forecasting in Turbulent Market Periods Using Neuro-Fuzzy Systems. *Operational Research.* 2016. Vol. 16. P. 245—269.

РОЗДІЛ 4

МЕТОД ІНДУКТИВНОГО МОДЕЛЮВАННЯ В ЗАДАЧАХ ІНТЕЛЕКТУАЛЬНОГО АНАЛІЗУ ДАНИХ

4.1. Загальна характеристика та основні принципи МГУА

4.2. Нечіткий МГУА. Основні ідеї. Математична модель НМГУА

4.3. Опис алгоритму НМГУА

4.4. НМГУА з гауссовими та дзвоноподібними функціями належності

4.5. Нечіткий МГУА з різними видами часткових описів

4.6. Адаптація коефіцієнтів лінійної інтервальної моделі

4.7. Порівняльний аналіз МГУА та нечіткого МГУА

Висновки

Список літератури

Цей розділ присвячено дослідженню методу індуктивного моделювання, відомого як *метод групового урахування аргументів* (МГУА), у задачах інтелектуального аналізу даних (*Data Mining*), зокрема в задачах прогнозування у макроекономіці та фінансовій сфері. Основна проблема полягає у створенні прогнозних моделей та визначенні невідомої функціональної залежності між прогнозованою величиною та заданим набором макроекономічних показників за експериментальними даними, коли аналітичний вигляд моделі заздалегідь невідомий.

Перевагою методу індуктивного моделювання МГУА є можливість автоматизованої побудови об'єктивної моделі в процесі роботи алгоритму, а також ефективна робота з невеликими обсягами даних. Особливістю нечіткого варіанта МГУА є формування інтервальних оцінок для прогнозованої змінної, що дає змогу оцінювати точність отриманих прогнозів.

У розділі наведено огляд ключових результатів, досягнутих у галузі нечіткого методу самоорганізації, виконано аналіз застосування різних видів функцій власності та оцінено перспективи використання нечіткого МГУА для задач прогнозування у макроекономіці та фінансовій сфері.

4.1. Загальна характеристика та основні принципи МГУА

Постановка задачі. Задано множину даних: вхідні змінні $\{X(1), X(2), \ldots, X(N)\}$ та вихідні змінні $\{Y(1), Y(2), \ldots, Y(N)\}$, де $X = [x_1, x_2, \ldots, x_n]$ — n-вимірний вектор; N — число точок спостереження.

Потрібно на основі спостережуваних даних побудувати модель $Y = F(x_1, x_2, \ldots, x_n)$, адекватну цим даним, причому отримана модель у певному розумінні повинна мати найменшу складність. В окремому випадку під час вирішення задачі прогнозування як вихідну змінну використовують модель $X(N + K) = f(X(1), \ldots, X(N))$, де K — довжина інтервалу упередження.

Відмінними ознаками цієї задачі є:

1) вид функціональної залежності невідомий, а визначено лише клас моделей, наприклад, поліномів довільного ступеня нелінійності чи гармонійний ряд (Фур'є);

2) коротка вибірка даних;

3) часові ряди $x_i(t)$, у загальному випадку, нестаціонарні.

У разі, якщо застосування класичних методів статистичного аналізу, як-от регресійний чи дисперсійний, неможливе, потрібно звертатися до «нестандартних» методів, заснованих, наприклад, на використанні ідей штучного інтелекту. До них належить метод групового урахування аргументів, запропонований О.Г. Івахненком. МГУА був розвинений у численних роботах О.Г. Івахненка та його учнів [4.17—4.20] і є способом індуктивного моделювання складних систем.

МГУА запозичує ідеї з біології, а саме — механізми еволюції:

1) схрещування або гібридизацію батьківських пар (аргументів) та генерацію нащадків;

2) селекцію та відбір кращих.

Основні принципи МГУА. Наведемо основні принципи МГУА, справедливі як для чіткого, так і для нечіткого його варіантів. Повну залежність між входами $X(i)$ та виходами $Y(i)$ у класі поліноміальних моделей можна зобразити за допомогою узагальненого полінома Колмогорова — Габора [4.2, 4.17].

Нехай є $X = (x_1, x_2, \ldots, x_n)$, тоді поліном Колмогорова — Габора має вигляд

$$Y = a_0 + \sum_{i=n}^{n} a_i x_i + \sum_{i=1}^{n} \sum_{i \le j}^{n} a_{ij} x_i x_j + \sum_{i=n}^{n} \sum_{i \le j}^{n} \sum_{i \le j}^{n} a_{ijk} x_i x_j x_k + \ldots,$$

де всі коефіцієнти a_0, a_i, a_{ij} невідомі.

Під час побудови моделі (визначення значень коефіцієнтів) як критерій використовують критерій регулярності (точності):

$$\overline{\varepsilon}^2 = \frac{1}{N} \sum_{i=n}^{N} (y_i - f(X_i))^2,$$

де N — обсяг вибірки спостережень.

Необхідно забезпечити мінімум $\overline{\varepsilon}^2$.

Принцип множинності моделей. Існує множина моделей на даній вибірці, що забезпечують нульову похибку (для цього досить підвищувати ступінь полінома моделі).

Тобто, якщо є N вузлів інтерполяції, то можна побудувати ціле сімейство моделей, кожна з яких, проходячи через експериментальні точки, даватиме нульову або мінімальну похибку ($\overline{\varepsilon}^2 = 0$).

Принцип самоорганізації. Позначимо S — складність моделі (визначається числом членів полінома Колмогорова — Габора). Значення похибки залежить від складності моделі. Причому зі зростанням складності вона спочатку зменшується, досягає мінімуму, а потім лишається незмінною або починає зростати. Потрібно вибрати таку оптимальну складність, за якої похибка буде мінімальною. Крім того, якщо враховувати дію перешкод, можна виділити такі моменти [4.17— 4.20]:

1) за різного рівня завад залежність $\overline{\varepsilon}^2$ від складності S буде змінюватися, зберігаючи при цьому загальне спрямування (мається на увазі, що з підвищенням складності вона спочатку зменшуватиметься, а потім зростатиме);

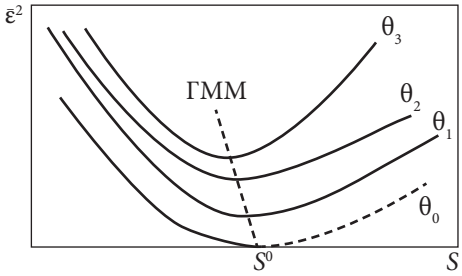


Рис. 4.1. Залежність точності моделі від складності S

2) у разі збільшення рівня перешкод величина $\min\limits_{S}\overline{\varepsilon}^2$ зростатиме;

3) зі збільшенням рівня перешкод оптимальна складність $S_0 = \overline{\varepsilon}^2(s)$ буде зменшуватися (оптимальне значення складності зміщуватиметься ліворуч). Причому $\overline{\varepsilon}^2(S_0) > 0$, якщо рівень перешкод не нульовий (див. рис. 4.1, де ГММ — геометричне місце мінімумів; θ — рівень перешкод, $\theta_0 < \theta_1 < \theta_2 < \theta_3$).

Теорема про неповноту Геделя. У будь-якій формальній логічній системі є ряд тверджень і теорем, які не можна ні спростувати, ні довести, залишаючись у межах цієї системи аксіом. Тому для їх доведення необхідно вийти за рамки системи і використати зовнішню інформацію (метаінформацію).

У разі задачі побудови моделі за вибіркою маємо ситуацію, що вибірка завжди неповна і тому неможливо оцінити якість моделі і знайти адекватну модель. Один зі способів подолання неповноти вибірки, яка є наслідком теореми про неповноту Геделя, — це *принцип зовнішнього доповнення*. Він полягає ось у чому. Всю вибірку поділяють на дві частини — навчальну та перевірну. При цьому перевірна — це вибірка, точки якої не використовувалися під час навчання системи та побудови моделі (тобто в пошуку значень коефіцієнтів полінома Колмогорова — Габора). Пошук найкращої моделі здійснюється в такий спосіб [4.2, 4.17—4.22]:

- всю вибірку поділяють на навчальну та перевірну: $N_{\text{виб}} = N_{\text{навч}} + N_{\text{перев}}$;
- на навчальній вибірці $N_{\text{навч}}$ визначають оцінки коефіцієнтів $\overline{a}_0$, $\overline{a}_i$, $\overline{a}_{ij}$.
- на перевірній вибірці $N_{\text{перев}}$ відбирають найкращі моделі.

Принцип свободи вибору (неостаточність проміжних рішень).

1. Для кожної пари x_i і x_j будуємо часткові описи (всього C_n^2): або $y_S = \varphi(x_i, x_j) = a_0 + a_i x_i + a_j x_j$, $s = 1...C_n^2$ (лінійні), або $y = \varphi(x_i, x_j) = a_0 + a_i x_i + a_j x_j + a_{ii} x_i^2 + a_{ij} x_i x_j + a_{jj} x_j^2$, $s = 1...C_n^2$ (квадратичні).

2. Визначаємо коефіцієнти цих моделей за МНК, використовуючи навчальну вибірку, тобто знаходимо оцінки $\overline{a}_0, \overline{a}_1, \ldots, \overline{a}_j, \ldots, \overline{a}_N, \overline{a}_{11}, \ldots, \overline{a}_{ij}, \ldots, \overline{a}_{NN}$.

3. Далі на перевірній вибірці для кожної з цих моделей шукаємо оцінку за *критерієм регулярності*:

$$\overline{\delta}_s^2 = \frac{1}{N_{\text{перев}}} \sum_{i=1}^{N_{\text{перев}}} [Y(k) - \overline{Y}_S(k)]^2,$$

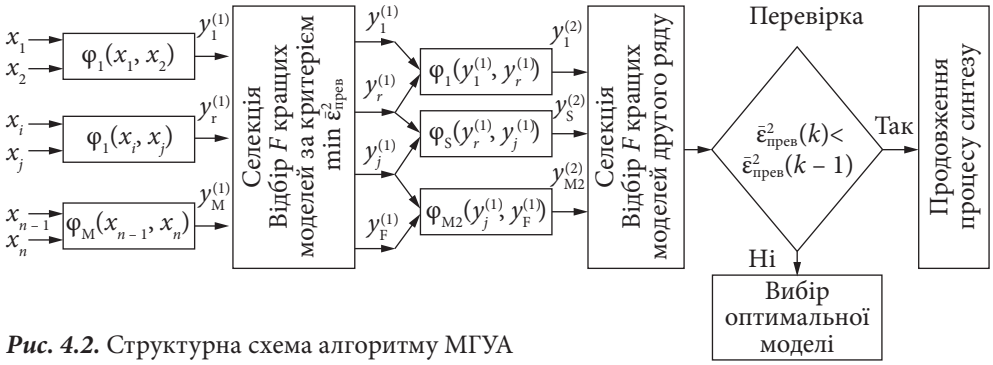


Рис. 4.2. Структурна схема алгоритму МГУА

де $Y(k)$ — дійсне значення вихідної змінної в k-й точці перевірної вибірки; $\overline{Y}_s(k)$ — вихідне значення у k-й точці перевірної вибірки відповідно до s-ї моделі; $N_{\text{перев}}$ — число точок перевірної вибірки), або за *критерієм незміщеності*:

$$N_{\text{зм}} = \frac{1}{N_1 + N_2} \sum_{k=1}^{N_{\text{перев}}} (y_k^* - y_k^{**})^2,$$

де вся вибірка поділяється на дві частини — N_1 і N_2; y_k^* — виходи моделі, побудованої на вибірці N_1; y_k^{**} — виходи моделі, побудованої на вибірці N_2.

4. Визначаємо F найкращих моделей за одним із зазначених критеріїв. Вибрані моделі y_i подаємо на другий ряд. Шукаємо $z_I = \varphi^{(2)}(y_i, y_j) = a_0^{(2)} + a_1^{(2)} y_i + a_2^{(2)} y_j + a_3^{(2)} y_i^2 + a_4^{(2)} y_i y_j + a_5^{(2)} y_j^2$.

Оцінка тут така сама, як у першому ряді. Відбір найкращих здійснюється знову так само, але $F_2 < F_1$. Процес конструювання рядів повторюється доти, доки середній квадрат похибки не зменшуватиметься. Коли на шарі m отримуємо збільшення похибки $\bar{\varepsilon}^2$, процес синтезу моделі припиняємо.

Структурну схему алгоритму МГУА наведено на рис. 4.2.

Основні переваги методу МГУА, які зумовили його популярність та широке використання не лише в Україні, а й за кордоном:

1) немає потреби у завданні моделі в явному вигляді, модель конструюється сама у процесі роботи алгоритму;

2) робота на коротких вибірках (коли число визначених коефіцієнтів моделі n більше від кількості точок спостереження N).

Водночас класичний метод МГУА має низку недоліків:

1) у разі близьких експериментальних точок можливе явище виродженості матриці нормальних рівнянь Гаусса («індуцит» у термінології О.Г. Івахненка), внаслідок чого виникає потреба у застосуванні спеціальних методів регуляризації;

2) отримання точкової моделі (прогнозу), в деяких випадках бажано мати довірчий інтервал, який характеризує точність прогнозу.

Тому останніми роками проводяться інтенсивні дослідження для розроблення нових методів, позбавлених зазначених недоліків. Таким методом є нечіткий МГУА (НМГУА), який будує інтервальну модель регресії і для знаходження моделі (прогнозу) не використовує метод найменших квадратів (МНК). Тому явища виродженості тут немає.

4.2. Нечіткий МГУА. Основні ідеї. Математична модель НМГУА

Цей метод було розроблено для роботи з нечіткою інформацією і побудови нечітких моделей. Він використовує основні ідеї та принципи МГУА і вільний від його недоліків.

У роботах [4.2, 4.23, 4.24] розглянуто лінійну інтервальну модель регресії:

$$Y = A_0 Z_0 + A_1 Z_1 + ... + A_n Z_n. \tag{4.1}$$

де A_i — трикутні нечіткі числа, що описуються парою параметрів:

$$A_i = (\alpha_i, c_i),$$

де α_i — центр інтервалу; c_i — його ширина, $c_i = 0$.

Тоді Y — нечітке число, параметри якого визначаються так:

центр інтервалу $\alpha_y = \sum \alpha_i z_i = \alpha^T z$, (4.2)

ширина інтервалу $c_y = \sum c_i |z_i| = c^T |z|$. (4.3)

Для того щоб інтервальна модель була коректною, необхідно, щоб дійсне значення вихідної величини належало інтервалу невизначеності, що описується обмеженнями:

$$\begin{cases} \alpha^T z - c^T |z| \le y; \\ \alpha^T z + c^T |z| \ge y. \end{cases} \tag{4.4}$$

Наприклад, для часткового опису виразу

$$f(x_i, x_j) = A_0 + A_1 x_i + A_2 x_j + A_3 x_i x_j + A_4 x_i^2 + A_5 x_j^2 \tag{4.5}$$

у загальній моделі (4.1) потрібно покласти $z_0 = 1$, $z_1 = x_i$, $z_2 = x_j$, $z_3 = x_i x_j$, $z_4 = x_i^2$, $z_5 = x_j^2$.

Припустимо, що спостерігаємо навчальну вибірку $\{z_1, z_2, \ldots, z_M\}$, $\{y_1, y_2, \ldots, y_M\}$.

Тоді для адекватності моделі виду (4.1) необхідно знайти такі , для яких виконувались би співвідношення

$$\begin{cases} \alpha^T z_k - c^T |z_k| \le y_k; \\ \alpha^T z_k + c^T |z_k| \ge y_k. \end{cases} \quad k = \overline{1, M}. \tag{4.6}$$

Сформулюємо основні вимоги до лінійної оцінної інтервальної моделі для часткового опису виразу (4.5) [4.2, 4.24].

Знайти такі значення параметрів (α_i, c_i) нечітких коефіцієнтів, за яких:
1) значення, що спостерігаються, y_k, потрапляли б в оцінний інтервал для Y_k;
2) «сумарна ширина» оцінного інтервалу була мінімальною.

Ці вимоги можна звести до такої задачі лінійного програмування (ЛП):

$$(C_0 M + C_1 \Sigma_{k=1}^{M} |x_{ki}| + C_2 \Sigma_{k=1}^{M} |x_{kj}| + C_3 \Sigma_{k=1}^{M} |x_{ki} x_{kj}| + C_4 \Sigma_{k=1}^{M} |x_{ki}^2| + C_5 \Sigma_{k=1}^{M} |x_{kj}^2|) \quad (4.7)$$

за умов:

$$\alpha_0 + \alpha_1 x_{ki} + \alpha_2 x_{kj} + \alpha_3 x_{ki} x_{kj} + \alpha_4 x_{ki}^2 + \alpha_5 x_{kj}^2 - (C_0 + C_1 |x_{ki} + C_2 |x_{kj}| + \\ + C_3 |x_{ki} x_{kj}| + C_4 |x_{ki}^2| + C_5 |x_{kj}^2|) \le y_k;$$

$$\alpha_0 + \alpha_1 x_{ki} + \alpha_2 x_{kj} + \alpha_3 x_{ki} x_{kj} + \alpha_4 x_{ki}^2 + \alpha_5 x_{kj}^2 + (C_0 + C_1 |x_{ki}| + C_2 |x_{kj}| + \\ + C_3 |x_{ki} x_{kj}| + C_4 |x_{ki}^2| + C_5 |x_{kj}^2|) \ge y_k; \quad k = \overline{1, M}; \quad (4.8)$$

$$C_p \ge 0, \; p = \overline{0,5}, \quad (4.9)$$

де k — номер точки вимірювання.

Як бачимо, задача (4.7)—(4.9) є задачею ЛП. Однак незручність форми (4.7), (4.8) для застосування стандартних методів ЛП полягає в тому, що немає обмежень невід'ємності для змінних α_i. Тому для її розв'язання переходимо до двоїстої задачі, ввівши двоїсті змінні $\{\delta_k\}$ і $\{\delta_{k+M}\}$, $k = \overline{1, M}$.

Розв'язавши двоїсту задачу симплекс-методом і знайшовши оптимальні значення двоїстих змінних $\{\delta_k\}$, $\{\delta_{k+M}\}$, визначимо також оптимальні значення шуканих змінних α_i, c_i, $i = \overline{0,5}$, а також нечітку модель для часткового опису (4.5).

4.3. Опис алгоритму НМГУА

Наведемо короткий опис алгоритму НМГУА [4.2, 4.24].

1. Вибір загального виду моделі, яким описуватиметься шукана залежність.

2. Вибір зовнішніх критеріїв оптимальності (критерію регулярності $\overline{\delta}^2$ або незміщеності $N_{зм}$).

3. Вибір загального виду опорної функції (виду часткових описів), наприклад лінійного або квадратичного.

4. Розбиття вибірки на навчальну $N_{навч}$ і перевірну $N_{перев}$.

5. Присвоюємо нульові значення лічильнику числа моделей k та лічильнику числа рядів r.

6. Генеруємо нову часткову модель f_r виду (4.5) на навчальній вибірці. Вирішуємо задачу ЛП (4.7)—(4.9) і знаходимо шукані значення α_i, c_i.

7. Визначаємо за перевірною вибіркою $N_{перев}$ значення зовнішнього критерію ($N_{зм\,k}^{(r)}$) або ($\delta_k^{(2)}(r)$).

8. $k = k + 1$. Якщо $k < C_n^2$ для $r = 1$ або $k < C_F^2$ для $r > 1$, переходимо на крок 6.

9. Обчислюємо найкраще значення критерію для моделей r-ї ітерації ($N_{зм}^{(r)}$ або $\delta^{(2)}(r)$). Якщо $r = 1$, то переходимо на крок 6, поклавши $r = r + 1$, $k = 1$, інакше — на крок 10.

10. Якщо $|N_{зм}(r) - N_{зм}(r-1)| < \varepsilon$ для досить малого ε, або $\delta^{(2)}(r) > \delta^{(2)}(r-1)$, переходимо на крок 11, інакше — відбираємо F кращих моделей і, поклавши $r = r + 1$, $k = 1$, переходимо на крок 6 і виконуємо наступну $(r + 1)$-ітерацію.

11. З F моделей попереднього ряду за критерієм регулярності знаходимо найкращу модель.

4.4. НМГУА з гауссовими та дзвоноподібними функціями належності

У перших роботах, присвячених нечіткому МГУА [4.23, 4.24], розглядалися функції належності (ФН) нечітких коефіцієнтів трикутного вигляду. Оскільки нечіткі числа (НЧ) можуть мати й інший вид ФН, цікавий розгляд інших класів ФН у завданнях моделювання на основі МГУА. У роботі [4.10] описано нечіткі моделі з гауссовими та дзвоноподібними ФН.

Нечіткі числа з гауссовою ФН

Назвемо нечітким числом B з гауссовою ФН нечітку множину з ФН:

$$\mu_B(x) = e^{-\frac{1}{2}\frac{(x-a)^2}{c^2}}.$$

Таке НЧ задається парою чисел $\beta = (a, c)$, де a — центр, c — величина, що характеризує ширину інтервалу (рис. 4.3).

Нехай шукана лінійна інтервальна модель для часткового опису НМГУА має вигляд (4.5). Тоді завдання ставиться так: знайти такі нечіткі числа A_i, тобто параметри (α_i, c_i), щоб:

1) спостереження y_k належало даній нечіткій множині Y_k зі ступенем, не меншим ніж α, $0 < \alpha < 1$;

2) сумарна ширина оцінного інтервалу рівня α була б мінімальною.

Ширина оцінного інтервалу рівня α дорівнює (див. рис. 4.3) $d_a = y_2 - y_1 = 2(y_2 - a)$, де $(y_2 - a)$ визначається за умови:

$$\alpha = \exp\left\{-\frac{1}{2}\frac{(y_2 - a)^2}{c^2}\right\}. \tag{4.10}$$

Звідси $d_\alpha = 2c\sqrt{-2\ln\alpha})$. Для поточної точки k маємо $d_\alpha^k = 2c_k\sqrt{-2\ln\alpha})$, і тоді цільову функцію можна записати у вигляді

$$\sum_{k=1}^{M} d_\alpha^k = \sum_{k=1}^{M} 2C_k\sqrt{-2\ln\alpha)} = 2\sqrt{-2\ln\alpha)}\sum_{k=1}^{M}\sum_{i=1}^{n} C_i\,|z_{ik}|. \tag{4.11}$$

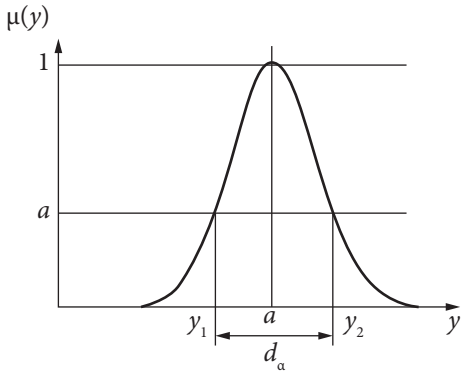


Рис. 4.3. Ширина оцінного інтервалу рівня α

Оскільки $2\sqrt{-2\ln\alpha}$ — додатна константа, що не впливає на набір c_i, який мінімізує цільову функцію (4.11), можна поділити цільову функцію на цю константу і привести цільову функцію до вихідного вигляду.

Тепер розглянемо першу вимогу: $\mu(y_k) \geq \alpha$. Вона еквівалентна виразу

$$\exp\left\{-\frac{1}{2}\frac{(y_k - a_k)^2}{c_k^2}\right\} \geq \alpha.$$

Ця нерівність приводиться до системи нерівностей:

$$\begin{aligned} a_k + c_k\sqrt{-2\ln\alpha} &\geq y_k; \\ a_k - c_k\sqrt{-2\ln\alpha} &\leq y_k. \end{aligned} \qquad (4.12)$$

З урахуванням того що $a_k = \sum_{i=n}^{n} a_i z_{ki}$; $c_k = \sum_{i=n}^{n} c_i |z_{ki}|$, задача знаходження нечіткої моделі остаточно зводиться до задачі ЛП вигляду за [4.2]

$$(C_0 M + C_1 \textstyle\sum_{k=n}^{M}|x_{ki}| + C_2 \sum_{k=n}^{M}|x_{kj}| + C_3 \sum_{k=n}^{M}|x_{ki}x_{kj}| + C_4 \sum_{k=n}^{M}|x_{ki}^2| + C_4 \sum_{k=n}^{M}|x_{kj}^2|) \quad (4.13)$$

за умов:

$$\left.\begin{aligned} \alpha_0 + \alpha_1 x_{ki} + \ldots + \alpha_5 x_{kj}^2 + (C_0 + C_1|x_{ki}| + \ldots + C_5|x_{kj}^2|)\sqrt{-2\ln\alpha} \geq y_k; \\ \alpha_0 + \alpha_1 x_{ki} + \ldots + \alpha_5 x_{kj}^2 - (C_0 + C_1|x_{ki}| + \ldots + C_5|x_{kj}^2|)\sqrt{-2\ln\alpha} \leq y_k; \end{aligned}\right\} \quad (4.14)$$

$$k = \overline{1, M}.$$

Для розв'язання цієї задачі, як і у випадку ФН трикутного вигляду, можна перейти до двоїстої задачі:

$$(\textstyle\sum_{k=1}^{M} y_k \cdot \delta_{k+M} - \sum_{k=1}^{M} y_k \cdot \delta_k) \qquad (4.15)$$

за умов рівностей:

$$\begin{aligned} \textstyle\sum_{k=1}^{M} \delta_{k+M} - \sum_{k=1}^{M} \delta_k &= 0; \\ \textstyle\sum_{k=1}^{M} X_{ki} \cdot \delta_{k+M} - \sum_{k=1}^{M} X_{ki_i} \cdot \delta_k &= 0; \end{aligned} \qquad (4.16)$$

…

$$\Sigma_{k=1}^{M} X_{kj}^2 \cdot \delta_{k+M} - \Sigma_{k=1}^{M} X_{kj}^2 \cdot \delta_k = 0$$

та умов нерівностей:

$$\left.\begin{array}{l} \Sigma_{k=1}^{M} \delta_k + \Sigma_{k=1}^{M} \delta_{k+M} \le \dfrac{M}{\sqrt{-2\ln\alpha}}; \\ \dots \\ \Sigma_{k=1}^{M} |X_{kj}^2| \cdot \delta_k + \Sigma_{k=1}^{M} |X_{kj}^2| \delta_{k+M} \le \dfrac{\Sigma_{k=1}^{M} |X_{kj}^2|}{\sqrt{-2\ln\alpha}}; \end{array}\right\} \qquad (4.17)$$

$$\delta_k \ge 0, \quad k = \overline{1,2M}.$$

Ця задача розв'язується стандартними методами.

Нечіткі моделі з дзвоноподібною функцією належності

Поширений клас нечітких чисел становлять НЧ із дзвоноподібною ФН:

$$\mu_B(x) = \frac{1}{1 + \left(\dfrac{x-a}{c}\right)^2}. \qquad (4.18)$$

Таке НЧ задається парою чисел $B = (a, c)$, де a — центр, c — величина, що характеризує ширину функції (рис. 4.4).

Для нечіткої моделі $Y = A_0 + A_1 x_i + A_2 x_j + A_3 x_i x_j + A_4 x_i + A_5 x_j$ відповідна задача ЛП для знаходження невідомих параметрів (α_i, c_i) має вигляд [4.2, 4.24]

$$(C_0 M + C_1 \Sigma_{k=1}^{M} |x_{ki}| + C_2 \Sigma_{k=1}^{M} |x_{kj}| + C_3 \Sigma_{k=1}^{M} |x_{ki} x_{kj}| + C_4 |x_{ki}^2| + C_5 |x_{kj}^2|) \qquad (4.19)$$

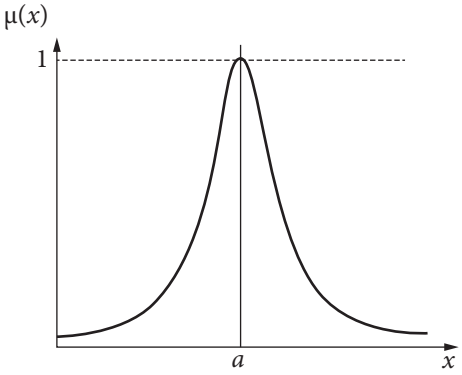


Рис. 4.4. Дзвоноподібна функція належності

за умов:

$$\alpha_0 + \alpha_1 x_{ki} + ... + \alpha_5 x_{kj}^2 + (C_0 + C_1 |x_{ki}| + ... + C_5 |x_{kj}^2|)\sqrt{\frac{1-\alpha}{\alpha}} \geq y_k;$$

$$\alpha_0 + \alpha_1 x_{ki} + ... + \alpha_5 x_{kj}^2 - (C_0 + C_1 |x_{ki}| + ... + C_5 |x_{kj}^2|)\sqrt{\frac{1-\alpha}{\alpha}} \leq y_k, \qquad (4.20)$$

$$k = \overline{1, M},\ C_i \geq 0,\ i = \overline{0,5}.$$

Перейшовши від неї до двоїстої задачі, можна знайти потрібне рішення стандартними методами ЛП.

4.5. Нечіткий МГУА з різними видами часткових описів

Дослідження ортогональних поліномів як часткових описів алгоритмів НМГУА

Як відомо із загальної теорії МГУА, моделі-претенденти генеруються на основі так званих часткових описів — моделей певного виду (наприклад, лінійних поліномів), які, використовуючи як свої аргументи часткові описи попередніх етапів синтезу моделі оптимальної складності, створюють моделі-претенденти все складнішої структури. Але завдання вибору виду часткових описів не формалізовано, і цей вибір зумовлений такими перевагами ортогональних поліномів у задачах моделювання нелінійних процесів, характеристики яких апріорі невідомі:

- завдяки властивостям ортогональності обчислення коефіцієнтів поліноміального рівняння, що апроксимує процес, відбувається швидше, ніж для неортогональних поліномів;
- коефіцієнти поліноміального апроксимувального рівняння не залежать від порядку вихідної поліноміальної моделі.

Отже, за відсутності апріорної інформації про порядок полінома можна перевірити поліноми кількох порядків, при цьому коефіцієнти, отримані для нижчого порядку, залишаються такими самими в разі переходу до полінома вищого порядку. Ця властивість найбільш важлива для дослідження порядку полінома апроксимації; однією з властивостей ортогональних поліномів, що використовуються в цій роботі і найбільш широко — у задачах нелінійної апроксимації, є властивість майже однакових похибок. Ця властивість полягає в тому, що похибка апроксимації коливається в середині діапазону вимірів між двома майже однаковими межами. Завдяки цій властивості не виникають дуже великі похибки, наприклад викиди за межі діапазону даних, для яких виконується апроксимація; навпаки, здебільшого похибки малі, тобто відбувається «демпфування» похибок апроксимації.

Основні положення. Розглянемо рівняння апроксимації для одновимірної системи:

$$\bar{y}(x) = b_0F_0(x) + b_1F_1(x) + ... + b_mF_m(x), \tag{4.21}$$

де $\bar{y}$ — вихідна величина, яка оцінюється; $F_\nu(x)$ — ортогональний поліном порядку ν, який має властивість ортогональності, тобто

$$\Sigma_{i=0}^{r} F_\mu(x_i)F_\nu(x_i) = 0 \ \forall \ \mu \neq \nu, \tag{4.22}$$

де r — довжина навчальної вибірки.

Умову ортогональності подано в загальному вигляді (для неперервних змінних):

$$\int_a^b \omega(x)F_\mu(x)F_\nu(x)dx = 0, \tag{4.23}$$

де μ, ν — невід'ємні цілі числа; $\omega(x)$ — ваговий коефіцієнт.

Дослідження ортогональних поліномів Чебишева як часткових описів алгоритму НМГУА. Ортогональні поліноми Чебишева у такому випадку мають вигляд [4.2]

$$F_\nu(\xi) = T_\nu(\xi) = \cos(\nu \cdot \arccos\xi), \quad -1 \le \xi \le 1. \tag{4.24}$$

Ці поліноми мають такі властивості ортогональності:

$$\int_{-1}^{1} \frac{T_\mu(\xi)T_\nu(\xi)d\xi}{\sqrt{1-\xi^2}} \begin{cases} 0 \text{ при } \mu \neq \nu; \\ \dfrac{\pi}{2} \text{ при } \mu = \nu \neq 0; \\ \pi \text{ при } \mu = \nu = 0, \end{cases} \tag{4.25}$$

де $\sqrt{1-\xi^2}$ — ваговий коефіцієнт $\omega(\xi)$ рівняння (4.23).

Наведемо кілька поліномів Чебишева низького порядку:

$$T_0(\xi) = 1,$$
$$T_1(\xi) = \xi,$$
$$T_2(\xi) = 2\xi^2 - 1,$$
$$T_{\nu+1}(\xi) = 2\xi T_\nu(\xi) - T_{\nu-1}(\xi).$$

Поліном апроксимації для отримують на основі мінімізації функціонала S:

$$S = \int_{-1}^{1} \omega(\xi)(\bar{y}(\xi) - \Sigma_{i=0}^{m} b_iT_i(\xi))^2 d\xi. \tag{4.26}$$

Звідси випливає:

$$b_k = \begin{cases} \dfrac{1}{\pi}\displaystyle\int_{-1}^{1} \frac{\bar{y}(\xi)}{\sqrt{1-\xi^2}}\, d\xi, \ k = 0; \\ \dfrac{2}{\pi}\displaystyle\int_{-1}^{1} \frac{\bar{y}(\xi)T_k(\xi)}{\sqrt{1-\xi^2}}\, d\xi, \ k \neq 0. \end{cases} \tag{4.27}$$

З (4.27) отримуємо рівняння апроксимації:

$$\bar{y}(\xi) = \sum_{k=0}^{m} b_k T_k(\xi). \tag{4.28}$$

Як видно з наведених співвідношень, b_k у рівнянні (4.28) не залежить від вибору m. Тобто змінна b_j не потребує перерахунку $\forall j \le m$, тоді як це необхідно в разі неортогональної апроксимації, що потребує великих витрат часу.

Визначення найкращого порядку полінома апроксимації. Найкращий порядок m^* апроксимувального полінома можна отримати на основі гіпотези про те, що результати експериментів $y(i)$, $i = 1, 2 \ldots, r$ мають незалежний гауссів розподіл біля деякого поліноміального співвідношення $\bar{y}$-го порядку, наприклад $m^* + \mu$, де

$$\bar{y}_{m^*+\mu}(x_i) = \sum_{j=0}^{m^*+\mu} b_j x_i^j, \tag{4.29}$$

а дисперсія σ^2 розподілу $y - \bar{y}$ не залежить від μ.

Зрозуміло, що для дуже малих m ($m = 0, 1, 2, ...$) дисперсія зменшується зі збільшенням m. Оскільки відповідно до прийнятої раніше гіпотези дисперсія не залежить від μ, то найкращим порядком m^* є найменше значення m, для якого $\sigma_m \cong \sigma_{m+1}$.

Для отримання m^* потрібне обчислення поліномів апроксимації різних порядків. Оскільки коефіцієнти b_j у рівнянні (4.30) не залежать від m, то визначення найкращого порядку полінома значно прискорюється.

Побудова лінійної інтервальної моделі. Нехай маємо прогнозовану змінну Y та вхідні змінні $x_1, x_2, \ldots, x_n$. Шукатимемо залежність між ними у вигляді (4.30)

$$Y = A_1 f_1(x_1) + A_2 f_2(x_2) + ... + A_n f_n(x_n), \tag{4.30}$$

де A_i — нечіткі числа трикутного вигляду, $A_i = (a_i, C_i)$, а функції f_i визначаються як [4.23, 4.24]:

$$f_i(x_i) = \sum_{j=0}^{m_i} b_{i_j} T_j(x_i). \tag{4.31}$$

Порядок функції m_i визначається за допомогою гіпотези, викладеної в попередньому підрозділі. Отже, якщо покласти $z_i = f_i(x_i)$, то отримаємо лінійну інтервальну модель у її класичному вигляді.

Дослідження тригонометричних поліномів як часткових описів алгоритму МГУА

Загальні положення. Нехай маємо періодичну функцію з періодом P ($P > 0$):

$$g(x + P) = g(x). \tag{4.32}$$

Без втрати загальності можна розглянути будь-яку періодичну функцію як таку, що має період 2π:

$$f(x) = g(\frac{Px}{2\pi}); \tag{4.33}$$

$$f(x + 2\pi) = g(\frac{Px}{2\pi} + P) = f(x). \tag{4.34}$$

Нехай функція $f(x)$ — періодична з періодом 2π, визначена на інтервалі [–π; π], та її перша похідна $f(x)$ також визначена на [–π; π]. Тоді справедлива рівність

$$S(x) = f(x) \quad \forall\, x \in [-\pi; \pi], \tag{4.35}$$

де

$$S(x) = \frac{a_0}{2} + \Sigma_{j=1}(a_j \cos(jx) + b_j \sin(jx)). \tag{4.36}$$

а коефіцієнти a_j, b_j обчислюються за формулами Ейлера:

$$a_j = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x)\cos(jx)dx; \tag{4.37}$$

$$b_j = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x)\sin(jx)dx. \tag{4.38}$$

Тригонометричні поліноми. Тригонометричним поліномом порядку M називається такий вираз:

$$T_M(x) = \frac{a_0}{2} + \Sigma_{j=1}^{M}(a_j \cos(jx) + b_j \sin(jx)). \tag{4.39}$$

Визначення лінійної інтервальної моделі. Теорема стверджує, що $\exists M$, $2M < N$, що мінімізує вираз

$$\Sigma_{j=1}^{N}(f(x_i) - T_M(x_i))^2. \tag{4.40}$$

При цьому коефіцієнти тригонометричного полінома визначаються за формулами:

$$a_j = \frac{2}{N} \Sigma_{j=1}^{N} f(x_i)\cos(jx_i), \tag{4.41}$$

$$b_j = \frac{2}{N} \Sigma_{j=1}^{N} f(x_i)\sin(jx_i). \tag{4.42}$$

Побудова лінійної інтервальної моделі. Нехай маємо прогнозовану змінну Y та вхідні змінні $x_1, x_2, \ldots, x_n$. Шукатимемо залежність між ними у вигляді

$$Y = A_1 f_1(x_1) + A_2 f_2(x_2) + \ldots + A_n f_n(x_n), \tag{4.43}$$

де A_i — нечіткі числа трикутного вигляду, $A_i = (a_i, C_i)$, а функції f_i визначаються так:

$$f_i(x_i) = T_{M_i}(x_i). \tag{4.44}$$

Порядок M_i полінома визначається за допомогою теореми, викладеної вище. Отже, якщо покласти $z_i = f_i(x_i)$, то отримаємо лінійну інтервальну модель у її класичному вигляді. Слід зазначити, що перед початком моделювання вихідні дані слід пронормувати на інтервалі $[-\pi; \pi]$.

4.6. Адаптація коефіцієнтів лінійної інтервальної моделі

Основні положення

У разі прогнозування з використанням методів самоорганізації (зокрема, нечіткого МГУА) виникає проблема, пов'язана з необхідністю виконання великого обсягу повторних обчислень за зміни кількості точок навчальної послідовності хоча б на одну, а також у випадках прогнозування в реальному часі, коли потрібно швидко коригувати наявну модель відповідно до нових даних.

Для вирішення цієї проблеми у роботі запропоновано застосування покрокової адаптації коефіцієнтів нечіткої моделі прогнозування за допомогою методів стохастичної апроксимації та рекурентного методу найменших квадратів (РМНК).

Необхідність дослідження декількох алгоритмів адаптації коефіцієнтів зумовлена тим, що алгоритм стохастичної апроксимації, попри його переваги, є дещо штучним доповненням до алгоритму НМГУА. Він має важливий недолік: під час формування першої оцінки з послідовності коригувальних коефіцієнтів не враховується інформація, отримана раніше під час оцінювання вектора параметрів моделі у ході роботи НМГУА. Крім того, наявність вибору між кількома алгоритмами провести дає можливість виконати ширші експериментальні дослідження та сформулювати практичні рекомендації щодо їх використання у завданнях прогнозування.

Інформацію, отриману під час оцінювання параметрів моделі, можна враховувати двома способами для ініціалізації алгоритму покрокової адаптації. Перший спосіб передбачає використання рекурсивних методів ідентифікації. У цьому випадку вихід алгоритму НМГУА містить модель оптимальної складності разом із накопиченими даними, які можна використати для модифікації параметрів відповідно до нових вимірювань. Отже, оцінка параметрів на кожному наступному кроці ґрунтується на попередній оцінці, похибці моделі та інформаційній матриці, яка модифікується у процесі ідентифікації, накопичуючи дані для подальших кроків.

Цей підхід значно спрощує адаптацію: якщо зберегти інформаційну матрицю, отриману під час ідентифікації параметрів моделі оптимальної складності, то для адаптації параметрів достатньо виконати одну ітерацію відповідним методом рекурсивної ідентифікації. Проте рекурсивні методи

мають недолік стосовно МГУА — вони не забезпечують виконання однієї з умов коректності інтервальної моделі прогнозування, зокрема умови (4.6).

Отже, для забезпечення коректної адаптації параметрів моделей, отриманих за допомогою класичного алгоритму НМГУА, бажано модифікувати алгоритми РМНК так, щоб для адаптації параметрів було достатньо однієї ітерації відповідного методу рекурсивної ідентифікації.

Застосування методу стохастичної апроксимації для адаптації параметрів лінійної інтервальної моделі

Розглянемо дискретну стаціонарну систему:

$$y_k = \Phi(U_k, p), \tag{4.45}$$

де U_k — вектор вхідних змінних моделей; p — вектор параметрів, який потрібно оцінити (ідентифікувати); Ф — задана функція.

Оцінка вектора p на кроці $(n + 1)$ визначається як

$$\hat{p}_{n+1} = \hat{p}_n - \rho_n \Psi_n, \tag{4.46}$$

де Ψ_n — функція, яка залежить від U_n та y; ρ_n — послідовність скалярних коригувальних коефіцієнтів.

Векторна функція Ψ_n визначається так. Нехай $J_n(\hat{p}_n)$ — скалярний показник якості ідентифікації, що визначається як [4.2]

$$J_n(\hat{p}_n) = \frac{1}{2}(y_n - \Phi(U_n, \hat{p}_n))^2. \tag{4.47}$$

Тоді виконується рівність

$$\Psi_n = \frac{\partial J_n(\hat{p}_n)}{\partial p_n} = \begin{pmatrix} \frac{\partial J_n(\hat{p}_n)}{\partial p_{n_1}} \\ \dots \\ \frac{\partial J_n(\hat{p}_n)}{\partial p_{n_m}} \end{pmatrix}. \tag{4.48}$$

де $\hat{p}_{n_1}, \dots, \hat{p}_{n_m}$ — компоненти вектора $\hat{p}_n$.

Зі співвідношення (4.47) отримуємо

$$\frac{\partial J_n(\hat{p}_n)}{\partial p_n} = (\Phi(U_n, \hat{p}_n) - y_n) g(U_n, \hat{p}_n), \tag{4.49}$$

де

$$g(U_n, \hat{p}_n) = \frac{\partial \Phi(U_n, \hat{p}_n)}{\partial \hat{p}_n}. \tag{4.50}$$

У разі інтервальної моделі маємо:

$$\Phi(U_n, \hat{p}_n) = \hat{p}_n^{\mathrm{T}} U_n, \tag{4.51}$$

де $\hat{p}_n = (\alpha_1, \alpha_2, \ldots, \alpha_m)$ або $\hat{p}_n = (C_1, C_2, \ldots, C_m)$; $U_n = (z_1, z_2, \ldots, z_m)$ або $U_n = (|z_1|, |z_2|, \ldots, |z_m|)$ для адаптації коефіцієнтів α_i або C_i відповідно. Тоді

$$g(U_n, \hat{p}_n) = \frac{\partial \Phi(U_n, \hat{p}_n)}{\partial \hat{p}_n} = U_n, \tag{4.52}$$

внаслідок чого отримуємо

$$\Psi_n = (\Phi(U_n, \hat{p}_n) - y_n) U_n. \tag{4.53}$$

Перший із послідовності скалярних коригувальних коефіцієнтів визначається як

$$\rho_1 = \frac{1}{||g(U_n, \hat{p}_n)||^2}. \tag{4.54}$$

У разі лінійної інтервальної моделі маємо

$$\rho_1 = \frac{1}{||U_n||^2} = \frac{1}{U_n^T U_n}. \tag{4.55}$$

Отже, метод стохастичної апроксимації є зручним інструментом для рекурсивної ідентифікації параметрів моделей, які допускають лінеаризацію, а значить, може бути застосований для вирішення задачі адаптації коефіцієнтів моделей, отриманих за допомогою нечіткого методу групового урахування аргументів (НМГУА). Процедура адаптації коефіцієнтів застосовується як до параметрів α_i, так і до параметрів c_i.

У програмній реалізації адаптація коефіцієнтів моделі може виконуватись як користувачем у режимі покрокового прогнозування, так і автоматично в процедурі прогнозування на кілька періодів уперед. Принцип роботи алгоритму адаптації полягає в такому: коригування коефіцієнтів моделі здійснюється в разі, якщо фактичне значення прогнозованої змінної виходить за межі довірчого інтервалу, передбаченого моделлю.

Основним недоліком цього алгоритму є те, що під час формування оцінки першого з послідовності коригувальних коефіцієнтів не враховується інформація, отримана в процесі оцінювання вектора параметрів моделі за алгоритмом НМГУА.

З метою усунення цього недоліку доцільно модифікувати алгоритм оцінювання параметрів лінійної інтервальної моделі. Така модифікація має забезпечити, щоб вихід алгоритму НМГУА містив модель оптимальної складності разом із даними, накопиченими під час оцінювання вектора параметрів.

Застосування рекурентного методу найменших квадратів (РМНК) для адаптації параметрів лінійної інтервальної моделі

Розглянемо модель виду

$$y(k) = \theta^T \Psi(k) + \nu(k), \tag{4.57}$$

де $y(k)$ — залежна змінна; $\Psi(k)$ — вектор вимірювань; $\nu(k)$ — обурення випадкового характеру; θ — вектор параметрів, який потрібно оцінити.

Шукану оцінку вектора параметрів θ на кроці N виконують за формулою [4.2]

$$\hat{\theta}(N) = \hat{\theta}(N-1) + \gamma(N)[y(N) - \hat{\theta}^T(N-1)\Psi(N)], \tag{4.58}$$

де $\gamma(N)$ — вектор коефіцієнтів, який визначається за такою формулою:

$$\gamma(N) = \frac{P(N-1)\Psi(N)}{1 + \Psi^T(N)P(N-1)\Psi(N)}, \tag{4.59}$$

де $P(N-1)$ — інформаційна матриця, яка обчислюється за формулою

$$P(N-1) = P(N-2) - \frac{P(N-2)\Psi(N-1)\Psi^T(N-1)P(N-2)}{1 + \Psi^T(N-1)P(N-2)\Psi(N-1)}. \tag{4.60}$$

Як видно з виразу (4.60), інформаційну матрицю можна отримати в результаті оцінки параметрів. Адаптацію обох векторів параметрів виконують:

$$\theta_1^T = [\alpha_1, \ldots, \alpha_m]; \; \theta_2^T = [C_1, \ldots, C_m];$$

Необхідно паралельно адаптувати параметри α_i і c_i. Тоді одночасно оцінюватимемо два вектори параметрів за такими формулами:

$$\hat{\theta}_1(N) = \hat{\theta}_1(N-1) + \gamma_1(N)[y(N) - \hat{\theta}_1^T(N-1)\,\Psi_1(N)]; \tag{4.61}$$

$$\hat{\theta}_2(N) = \hat{\theta}_2(N-1) + \gamma_2(N)[y_c(N) - \hat{\theta}_2^T(N-1)\,\Psi_2(N)], \tag{4.62}$$

де

$$\Psi_1^T = [z_1 \ldots z_m]; \; \Psi_2^T = [|z_1| \ldots |z_m|]; \tag{4.63}$$

$$y_c(N) = |y(N) - \theta_1^T(N-1)\,\Psi_1(N)|.$$

4.7. Порівняльний аналіз МГУА та нечіткого МГУА

Порівняння на завданнях макроекономічного прогнозування для України. Як вихідні дані для прогнозування вибирали дані за макроекономічними показниками України з січня 2011 р. до листопада 2012 р. Прогнозованим показником слугував ВВП. У процесі експериментів варіювалося число оптимальних моделей, переданих на наступний рівень F = 2, 3, 4, а також відсоток співвідношення між навчальною і перевірною вибірками (50 %—50 %; 60 %—40 %; 70 %—30 %; 80 %—20 %; 90 %—10 %).

Нечіткий метод застосовували з використання трьох ФН: трикутної, гауссової і дзвоноподібної. Останні дві мали рівень значущості 0,8. Методи розглядали на двох моделях:

лінійна модель $f(x_i, x_j) = A_{01} + A_{11} * x_i + A_{12} * x_j$;

квадратична модель $f(x_i, x_j) = A_{01} + A_{11} * x_i + A_{12} * x_j + A_{21} * x_{i2} + A_{22} * x_{j2} + A_{23} x_i * x_j$.

Було проведено експерименти з усіма можливими параметрами моделей — усього 120 експериментів. Результати цих експериментів наведено у табл. 4.1 і на рис. 4.5 та 4.6.

Таблиця 4.1

Порівняння похибок за різних методів

Модель	Показник статистики	Навчальна вибірка, %				
		50	60	70	80	90
Лінійна трикутна	*MSE*	6,266	6,2712	6,3648	6,3568	6,1712
	MAPE	5,8455	5,88	5,796	4,0588	4,0004
Квадратична трикутна	*MSE*	5,4356	5,2	5,2416	6,1312	6,3228
	MAPE	5,2875	5,4663	5,2197	4,239	4,0506
Лінійна гауссова	*MSE*	4,3566	4,0299	4,0698	5,0128	5,1092
	MAPE	4,587	4,293	4,1715	3,7736	3,7736
Квадратична гауссова	*MSE*	4,11	4,05	3,99	4,82	4,82
	MAPE	4,11	4	3,99	4,03	4,03
Лінійна дзвоноподібна	*MSE*	4,6138	4,484	4,522	5,9396	5,6522
	MAPE	4,646	4,4312	4,393	4,25	4,393
Квадратична дзвоноподібна	*MSE*	4,4965	4,522	4,408	5,7001	5,4606
	MAPE	4,6287	4,4844	4,848	4,584	4,4312
Чіткий МГУА, лінійна	*MSE*	5,5074	5,4264	5,187	6,3624	6,5552
	MAPE	5,7963	5,67	5,3865	4,9484	4,6992
Чіткий МГУА, квадратична	*MSE*	5,22572	5,0787	4,8906	6,21215	6,06989
	MAPE	5,1375	5,3466	5,3865	5,1987	5,239

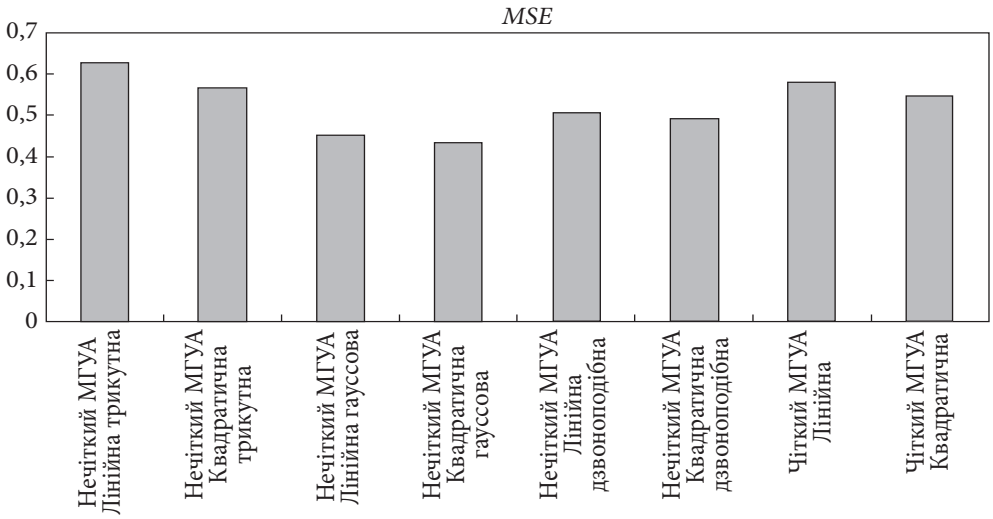


Рис. 4.5. Значення середньоквадратичної похибки за різних методів МГУА

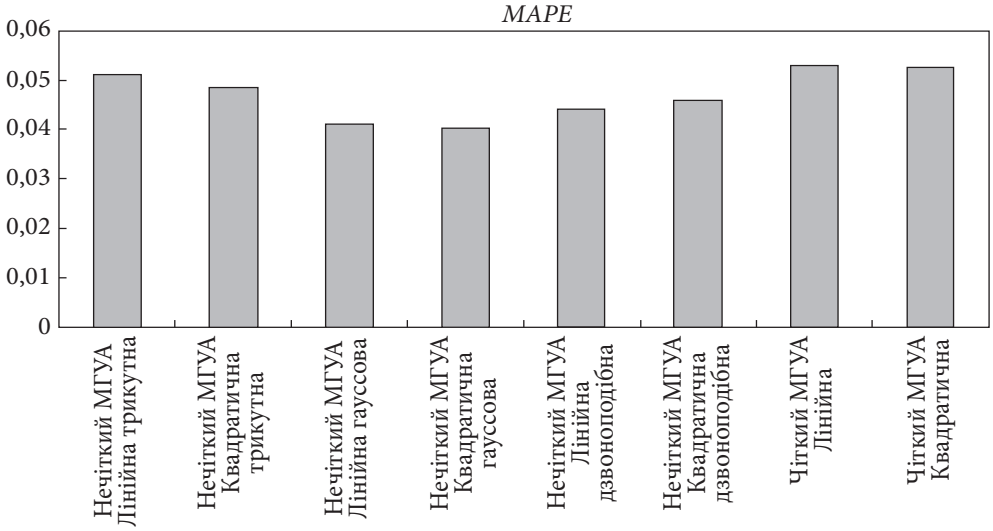


Рис. 4.6. Значення середньоабсолютної відсоткової похибки для різних методів

Отримані значення похибок свідчать про те, що деякі реалізації нечітких методів групового урахування аргументів дають якісніші результати, ніж чіткий МГУА.

Найкращим виявився нечіткий МГУА з гауссовою функцією належності — він дає мінімальне значення як середньоквадратичної похибки, так і средньоабсолютної відсоткової. За ним з невеликим відставанням йде нечіткий МГУА з дзвоноподібною ФН, далі — нечіткий МГУА з трикутною

ФН і чіткий МГУА, які дали близькі результати. Цікавим є той факт, що моделі з лінійним видом часткового опису показали дещо гірший результат, ніж моделі з квадратичним видом. Оптимальним виявився поділ вибірки на тренувальну і перевірну у співвідношенні 70 : 30.

Експериментальні дослідження МГУА та НМГУА в завданнях прогнозування на ринку цінних паперів *NYSE*

Вхідними величинами було обрано ціни провідних західних нафтових компаній, які котируються на фондовому ринку *NYSE:* ціни закриття компаній *Hess Corporation*, *Repsol YPF, S.A. (ADR), Eni S.p.A. (ADR), Exxon Mobil Corporation, Chevron Corporation, TOTAL S.A. (ADR).* Прогнозованими величинами слугували ціни закриття компанії *British Petrolium BP plc (ADR),* а також індекс *Dow Jones Industrial Average.* Для перевірки прогнозу використано дані щодо компанії *BP plc (ADR)* та дані щодо *Dow Jones Industrial Average* з 15 листопада 2011 р. по 17 листопада 2011 р. (табл. 4.2).

Класичний МГУА. У цій серії експериментів наведено результати, отримані в результаті використання класичного МГУА як методи прогнозу. Для дослідження моделлю часткових описів слугували лінійна, квадратична, тригонометричний поліном Фур'є та поліном Чебишева.

Усі прогнози виконували зі зміною обсягу навчальної вибірки, а саме 50, 70, 90 %. Розрахунки здійснювали зі зміною свободи вибору $F = 5, 6, 7$. Але, як показали результати дослідження, через те що коливання величин прогнозу невеликі, значення прогнозу відрізняються лише після 3—4-го знака після коми. Тож у наступних експериментах свободу вибору приймали такою, що дорівнювала $F = 6$.

З метою аналізу було обчислено похибки прогнозів, і в табл. 4.3 і 4.4 наведено такі критерії оцінки якості прогнозів: середньоквадратична похибка за перевірочною вибіркою, середньоквадратична похибка за всією вибіркою, середньоабсолютна відсоткова помилка по перевірочній вибірці, середньоабсолютна відсоткова похибка за всією вибіркою, критерій Дурбіна — Уотсона (Джеймс Дурбін — Джеффрі Уотсон), статистика *R-square*, критерій Акаїке (Хіроцуґу Акаїке), байєсівський інформаційний критерій та критерій Шварца (Гідон Шварц). Критерій середньоабсолютної відсоткової похибки взято у відсотках.

Результати прогнозів, отримані в результаті застосування НМГУА. Для дослідження моделей часткових описів використовували лінійну, квадратичну, поліном Фур'є, поліном Чебишева. Як функцію належності було обрано трикутну, лінійну та дзвоноподібну. Критерій середньоабсолютної відсоткової похибки взято у відсотках. Усі прогнози викону-

вали зі зміною співвідношення обсягу навчальної вибірки, а саме 50, 70, 90 % [4.2].

Результати експериментів наведено в табл. 4.3.

На рис. 4.7 наведено графіки прогнозів за НМГУА для лінійних часткових описів на три кроки вперед за $F = 6$ та обсягу навчальної вибірки 70 %, при цьому взято гауссову функцію належності, рівень значущості $\alpha = 0{,}7$.

Таблиця 4.2

Перевірні дані за індексом *Dow Jones Industrial Average*

Дата	*Dow Jones Industrial Average*
15.11.2011	12 096,16
16.11.2011	11 905,59
17.11.2011	11 770,73

Таблиця 4.3

Результати експериментів прогнозування на два кроки з використанням НМГУА

Крок	Критерій	Навчальна вибірка, %		
		50	70	90
1	*MSE test*	1,248355	0,793523	0,317066
	MSE	0,748864	0,612827	0,485599
	MAPE test	2,041366	2,065155	1,386096
	MAPE	1,452656	1,723657	1,505965
	DW	0,686478	1,763043	1,839065
	R-square	1,064526	0,938058	1,008042
	AIC	2,456985	2,228267	2,053752
	BIC	–3,268440	–3,063	–2,892253
	SC	2,506693	2,277	2,103459
2	*MSE test*	1,256828	1,871883	2,440575
	MSE	0,728499	0,989650	0,795699
	MAPE test	2,072085	2,367006	2,426847
	MAPE	1,468987	1,845798	1,656084
	DW	0,686478	1,795021	1,839065
	R-square	1,064526	0,874163	1,008042
	AIC	2,456985	2,147717	2,053752
	BIC	–3,268440	–2,986448	–2,892253
	SC	2,506693	2,197424	2,103459

Таблиця 4.4

Порівняльні результати прогнозування для індексу *Dow Jones*

Крок прогнозу	Критерії оцінки	Нечітка НМ	Чіткий МГУА			Нечіткий МГУА			
			Лінійна МЧО	Квадратична МЧО	Поліном Фур'є як МЧО	Поліном Чебишева як МЧО	Лінійна МЧО	Квадратична МЧО	Поліном Фур'є як МЧО	Поліном Чебишева як МЧО
1	*MSE*	514561	26900	38225	40142	23188	25176	21332	42205	24464
	MAPE	5,231	1,149	1,298	1,445	1,111	1,137	1,046	1,381	1,117
2	*MSE*	584371	27793	39460	40903	23978	25793	223491	45905	24767
	MAPE	5,992	1,167	1,322	1,445	1,119	1,159	1,064	1,518	1,12
3	*MSE*	624501	33670	50471	41740	27337	31942	38291	45190	27437
	MAPE	6,179	1,23	1,386	1,46	1,157	1,176	1,099	1,623	1,16

З метою аналізу прогнозу було прораховано критерії похибки, які наведено в табл. 4.4 і 4.5: середньоквадратична похибка (СКП) за перевірною вибіркою, СКП за всією вибіркою, середньоабсолютна відсоткова похибка (САВП) за перевірною вибіркою, САВП за всією вибіркою, критерій Дурбіна — Уотсона (*DW*), статистика *R-square*, критерій Акаіке (*AIC*), байєсівський інформаційний критерій (*BIC*) та критерій Шварца (*SC*). Критерії розраховувалися на кожному кроці прогнозу на перевірній вибірці [4.2, 4.11].

На рис. 4.8 показано прогноз на три кроки для квадратичних часткових описів для *BP plc* (*ADR*), при $F = 6$ і відсотка навчання 70 %, було взято дзвоноподібну функцію належності, рівень $\alpha = 0{,}7$.

Як випливає з результатів, наведених у табл. 4.4, каскадна нечітка нейромережа показала результати гірші, ніж обидва методи МГУА. Найкращі результати отримано за нечіткого МГУА з квадратичною моделлю часткових описів (МЧО) за 50%-ї навчальної вибірки і дзвоноподібної функції належності. Найгірші результати за обох методів показало використання полінома Фур'є як МЧО. Обидва методи МГУА продемонстрували високу результативність.

Якщо порівнювати ефективність обох методів — МГУА та

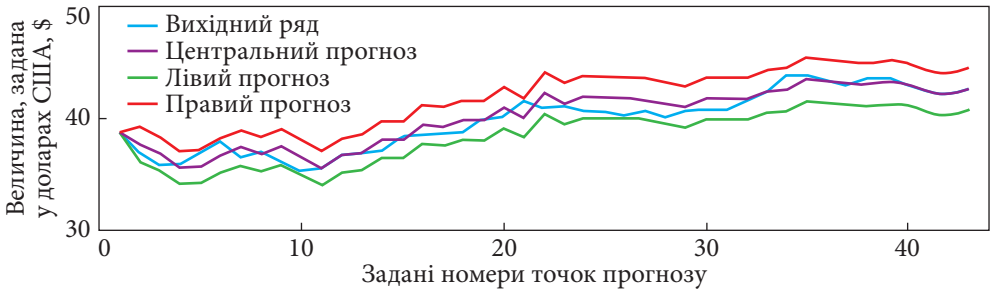


Рис. 4.7. Графіки прогнозів за НМГУА для лінійних часткових описів на три кроки вперед

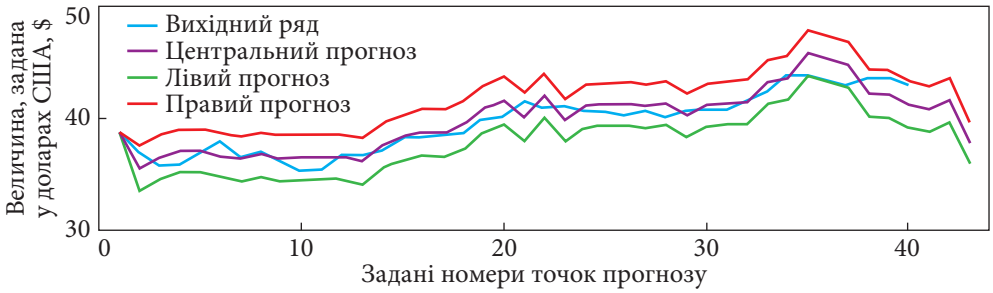


Рис. 4.8. Прогноз за НМГУА для квадратичних часткових описів цін акцій *BP plc (ADR)*

НМГУА за використаними моделями часткових описів, то поліноми Чебишева як МЧО в класичному МГУА дали більш точні результати, а за використання лінійної та квадратичної моделей часткових описів точніші оцінки отримано методом НМГУА [4.1, 4.2, 4.11].

Висновки

Розглянуто метод групового урахування аргументів (МГУА), який широко використовується для побудови моделей складних систем на основі експериментальних даних. Наведено і проаналізовано ключові принципи МГУА: принцип множинності моделей; самоорганізації; зовнішнього доповнення; свободи вибору.

Описано поліноміальний алгоритм МГУА, детально проаналізовано його властивості, а також підкреслено його переваги над іншими методами моделювання, такими як регресійний аналіз. Зокрема, МГУА вирішує задачу структурної ідентифікації, даючи можливість синтезувати оптимальну структуру моделі. Однак класичний алгоритм МГУА має недоліки: він використовує метод найменших квадратів (МНК) для визначення коефіцієнтів моделі. У разі, якщо матриця коефіцієнтів нормальних рівнянь Гаусса

вироджена, це може призводити до нестійких значень коефіцієнтів та високої волатильності моделі. Крім того, класичний МГУА не підтримує роботу з неповними або нечіткими даними.

Для усунення зазначених недоліків було розроблено нечіткий МГУА (НМГУА). У розділі описано основи теорії НМГУА, запропоновано інтервальну модель регресії, сформульовано вимоги до її адекватності, а також побудовано математичну модель для визначення нечітких коефіцієнтів часткових описів поліноміальної моделі, що ґрунтується на методах лінійного програмування.

Наведено алгоритм НМГУА, який є розширенням класичного МГУА для роботи з нечіткими моделями, і сформульовано його основні властивості. Здійснено порівняльний аналіз класичного та нечіткого МГУА.

Також розглянуто варіант МГУА з використанням ортогональних поліномів як часткових описів. Підкреслено основні переваги ортогональних поліномів порівняно з класичними поліноміальними моделями, зокрема можливість легко підвищувати ступінь полінома без перевизначення коефіцієнтів нижчого ступеня, що спрощує пошук оптимального ступеня.

Досліджено моделі НМГУА з ортогональними поліномами Чебишева та тригонометричними поліномами, їхні властивості та застосування. Розглянуто задачу адаптації моделей МГУА і запропоновано алгоритми для адаптації коефіцієнтів моделей НМГУА на основі стохастичної апроксимації та рекурентного МНК.

Проведено експериментальні дослідження роботи МГУА та НМГУА із застосуванням різних часткових описів у завданнях прогнозування курсів акцій та індексу Доу-Джонса на ринку цінних паперів. Виконано порівняльний аналіз, у результаті якого визначено оптимальні типи часткових описів для цих задач.

Нарешті, проведено порівняльний аналіз МГУА та НМГУА у завданнях прогнозування макроекономічних показників України, що дало змогу оцінити їхню ефективність і сформулювати рекомендації щодо застосування в різних контекстах.

СПИСОК ЛІТЕРАТУРИ

4.1. Russell S., Norvig P. Artificial Intelligence: A Modern Approach. Pearson, 2010. P. 693—750.

4.2. Zgurovsky M., Zaychenko Yu. Fundamentals of computational intelligence — System approach. Springer. 2016. 275 p.

4.3. Zgurovsky M., Zaychenko Yu. Big Data: Conceptual Analysis and Applications. Springer Nature Switzerland AG. 2019. 306 p.

4.4. Klir G. J., Yuan B. Fuzzy Sets and Fuzzy Logic: Theory and Applications. Prentice Hall, 1995. 591p.

4.5. Tanaka K. Fuzzy Modeling and Control: Theory and Applications. John Wiley & Sons, 2004. 320 p.

4.6. Kohonen T. Self-Organizing Maps. 3rd ed. 2001. Springer. 502 p.

4.7. Ross T. J. Fuzzy Logic with Engineering Applications. Wiley, 2010. 606 p.

4.8. Yager R. R., Filev D. P. Essentials of Fuzzy Modeling and Control. Wiley, 1994. 388 p.

4.9. Kruse R., Moewes C., Borgelt C. Computational Intelligence: A Methodological Introduction. Springer, 2016. 406 p.

4.10. Kasabov N. Fuzzy Logic and Neural Networks: Basic Concepts and Applications. 1998. 289 p.

4.11. Yamakawa T., Matsumoto M. Fuzzy Systems Engineering: Toward Human-Centric Computing. Wiley, 2009. 426 p.

4.12. Pedrycz W., Gomide F. An Introduction to Fuzzy Sets: Analysis and Design. MIT Press, 1998. 456 p.

4.13. Dubois D., Prade H. Possibility Theory: An Approach to Computerized Processing of Uncertainty. Springer, 1988. 352 p.

4.14. Taha H. A. Operations Research: An Introduction. Pearson, 2016. 848 p.

4.15. Zgurovsky M. Z., Pankratova N. D. System Analysis: Theory and Applications. Springer. 2007. 447 p.

4.16. Lootsma F. A. Fuzzy Logic for Planning and Decision Making. Springer, 1997. 210 p.

4.17. Ivakhnenko A.G., Ivakhnenko G.A., Mueller J.A. Self-organization of the neural networks with active neurons. *Pattern Recognition and Image Analysis*. 1994. 4 (2). P. 177—188.

4.18. Ivakhnenko A.G., Stepashko V.S. Disturbance tolerance of modeling. Kyiv: Naukova dumka, 1985.

4.19. Ivakhnenko A.G., Wuensch D., Ivakhnenko G.A. Inductive sorting-out GMDH algorithms with polynomial complexity for active neurons of neural networks. *Neural Networks*. 1999. Vol. 2. P. 1169—1173.

4.20. Ivakhnenko G.A. Self-organization of neuronet with active neurons for effects of nuclear test explosions forecasting. *System Analysis Modeling Simulation*. 1995. Vol. 20. P. 107—116.

4.21. Ivakhnenko O. G., Zaychenko J. P., Dimitrov V. D. Self-Organization of Predictive Modeling Systems. Translated version. Moscow: Soviet Radio, 1976. 363 p.

4.22. Haken H. Introduction to the Theory of Self-Organization. Springer, 1983. 300 p.

4.23. Zaychenko Yu.P., Zaets I.O. The Fuzzy Group Method of Data Handling and Its Application to the Tasks of the Macroeconomic Indexes Forecasting. *SAMS*. 2001. P. 1—11.

4.24. Zaychenko Yu. The Fuzzy Group Method of Data Handling and Its Application for Economical Processes forecasting. *Sci. Inquiry*. 2006. Vol. 7 (1). P. 83—98.

РОЗДІЛ 5

НЕЧІТКІ НЕЙРОННІ МЕРЕЖІ В ЗАДАЧАХ КЛАСИФІКАЦІЇ

5.1. ННМ *NEFClass*: архітектура, властивості,
алгоритм навчання бази правил і параметрів ФН

5.2. Аналіз властивостей *NEFClass*.
Модифікована система нечіткої класифікації *NEFClass-M*

5.3. Експериментальні дослідження ННМ *NEFClass*

5.4. Розпізнавання об'єктів електрооптичних зображень
з використанням нечітких НМ

5.5. Застосування системи *NEFClass* у завданні розпізнавання
об'єктів електрооптичних зображень

5.6. Експерименти з розпізнавання об'єктів
на електрооптичних зображеннях

5.7. Метод опорних векторів

5.8. Алгоритм налаштування *SVM*

5.9. Застосування методу опорних векторів

5.10. Метод класифікації *Random Forest*

Висновки

Список літератури

Метою цього розділу є аналіз нечітких нейронних мереж у задачах класифікації, які широко застосовують у техніці, економіці, соціології, медицині та інших галузях.

У розділі розглянуто базову нечітку нейронну мережу для класифікації *NEFClass (Neuro-Fuzzy Classifier)*, детально описано алгоритм навчання бази правил і функцій належності (ФН) нечітких множин. Проаналізовано переваги та недоліки системи *NEFClass* і наведено опис її модифікації — системи *NEFClass-M*, яка усуває недоліки базової системи.

Розглянуто результати численних порівняльних експериментів, виконаних для базової та модифікованої систем *NEFClass.* Окрему увагу приділено актуальному завданню розпізнавання об'єктів на електрооптичних зображеннях (ЕОЗ). Наведено приклади вирішення цього завдання за допомогою ННМ *NEFClass*, а також проведено порівняльний аналіз ефективності різних алгоритмів навчання цієї моделі у складних умовах дії перешкод.

5.1. ННМ *NEFClass:* архітектура, властивості, алгоритм навчання бази правил і параметрів ФН

На сьогодні класифікація даних є однією з найактуальніших сфер застосування систем обчислювального інтелекту. Для вирішення цих задач було запропоновано низку підходів, серед яких особливої популярності набули методи, що інтегрують нейронні мережі та нечіткі системи логічного виведення. Одним із таких методів є система *NEFClass (Neuro-Fuzzy Classifier)*, заснована на узагальненій архітектурі нечіткого перцептрона [5.4, 5.5].

Як базова, так і модифікована моделі *NEFClass* є похідними від загальної архітектури нечіткого перцептрона. Основна мета цих моделей — автоматичне виведення нечітких правил на основі навчальних даних, які розподіляються між класами, що не перетинаються. Нечіткість у системі виникає через неповноту або неточність вимірювань властивостей об'єктів, які потребують класифікації.

Нечіткі правила, які описують дані, формулюються у вигляді, що відображає зв'язки між властивостями об'єктів та їхньою належністю до кон-

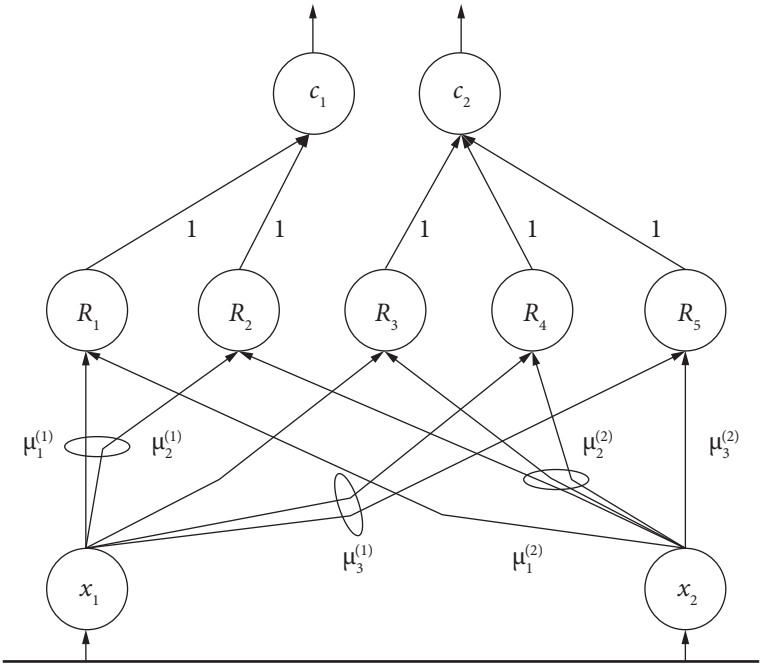


Рис. 5.1. Архітектура *NEFClass*

кретних класів. У розділі наведено методологію навчання бази правил та параметрів ФН, які є ключовими для точності класифікації.

Нечіткі правила, що описують дані, мають таку форму: якщо $x_1 \in \mu_{1i}$, $x_2 \in \mu_{2i}$, … $x_n \in \mu_{ni}$, то зразок (x_1, x_2, … x_n) належить до класу i, де μ_1, …, μ_n — ФН нечітких множин.

Завдання *NEFClass* — визначити ці правила, а також вид функцій належності для нечітких множин. Тут приймається, що перетин двох різних множин є порожнім.

Система *NEFClass* має тришарову послідовну архітектуру (рис. 5.1).

Перший шар U_1 містить вхідні нейрони, в яких представлено вхідні зразки. Активація вхідного нейрона a_x зазвичай змінює вхідне значення. Прихований шар U_2 містить нечіткі правила, третій шар складається з вихідних нейронів кожного класу. Активація для нейронів правил a_R і для нейронів вихідного шару a_c зі зразком p обчислюється як вирази

$$a_R^{(p)} = \min_{x \in U_1} \{W(x, R)(a_x^{(p)})\}, \tag{5.1}$$

$$a_c^{(p)} = \sum_{R \in U_2} W(R, c) a_R^{(p)}, \tag{5.2}$$

або альтернативно, як у виразі

$$a_c^{(p)} = \min_{R \in U_2} \{(a_R^{(p)})\}, \tag{5.3}$$

де $W(x, R)$ — нечітка вага з'єднання вхідного нейрона x з нейроном правил R, а $W(R, c)$ — нечітка вага з'єднання нейрона правил R з нейроном вихідного шару c.

Замість застосування операцій взяття максимуму та мінімуму можна використовувати інші функції t-норми і t-конорми відповідно.

База правил є апроксимацією невідомої функції φ: $R^n \to \{0; 1\}^m$ і описує вирішальне правило класифікації $\varphi(x) = (c_1, c_2, \ldots, c_m)$, де $c_i = 1$, $c_j = 0$ ($j = 1, \ldots, m, \forall j \neq i$), якщо x належить до класу C_i. Кожна нечітка множина маркується лінгвістичним терміном, як-от «великий», «малий», «середній» і т. ін.

Нечіткі множини та лінгвістичні правила становлять апроксимацію класифікаційної функції і визначають результат системи *NEFClass*. Їх отримують із вибірки за допомогою навчання. Обов'язково має виконуватися правило, що для кожного лінгвістичного значення (наприклад, «x_1 додатне і велике») може існувати тільки одне подання нечіткої множини.

Навчання у системі *NEFClass*. Навчання бази правил. Система *NEFClass* може бути побудована за частковими знаннями про зразки. Користувач має визначити кількість початкових нечітких множин для кожної з ознак об'єкта і задати значення $k_{\max}$ — максимальної кількості вузлів правил, які можуть бути створені у прихованому шарі. Для навчання застосовують трикутні функції належності. Розглянемо безпосередньо алгоритм навчання основи правил.

Використаємо систему *NEFClass* з n вхідними нейронами ($x_1, \ldots, x_n$), k нейронами правил, де $k \le k_{\max}$, і m вихідними нейронами $c_1, \ldots, c_m$.

Також задано навчальну множину зразків $L = \{(p_1, t_1), \ldots, (p_s, t_s)\}$, кожен із яких містить вхідний зразок $p \in R^n$ та бажаного зразка $t \in \{0, 1\}^m$. Алгоритм навчання складається з двох етапів.

Етап 1. Генерація бази правил. Перший етап, метою якого є створення нейронів правил системи *NEFClass*, полягає у таких діях [5.2, 5.4, 5.5].

1. Вибираємо зразок (p, t) із вибірки L.

2. Для кожного вхідного нейрона $x_i \in U_1$ знаходимо таку функцію належності $\mu_{J_i}^{(p)}$, що

$$\mu_{J_i}^{(p)} = \max_{j \in 1, \ldots, q_1} \{\mu_j^{(p)}(p_i)\}, \tag{5.4}$$

де $x_i = p_i$, p_i — значення x_i для зразка p.

3. Якщо, як і раніше, кількість вузлів правил k менша від $k_{\max}$ і не існує вузла правила R, такого, що

$$W(x_1, R) = \mu_{J_1}, \ldots, W(x_n, R) = \mu_{J_n},$$

тобто всі ваги набувають максимальних значень, створюємо такий вузол і з'єднуємо його з вихідним вузлом c_i, якщо $t_i = 1$, а також з усіма вхідними вузлами.

4. Якщо ще залишилися необроблені зразки в L і $k < k_{\max}$, то переходимо на крок 1. Інакше — зупинка.

5. Визначаємо базу правил за однією з трьох процедур:

а) «просте» навчання правил: залишаємо лише перші k правил (зупиняємо створення правил, якщо було створено $k = k_{\max}$ правил);

б) «краще» навчання правил: обробляємо зразки в L та накопичуємо активації кожного нейрона правил для кожного класу зразків, які були поширені. Якщо нейрон правил R показує більше накопичення активації для класу C_j, ніж для класу C_t, який було специфіковано для наслідку правила, змінюємо висновок нейрона R на C_j, тобто з'єднуємо R з нейроном виходу C_j. Продовжуємо оброблення зразків у L далі і обчислюємо для кожного нейрона правил функцію (накопичену активацію):

$$V_R = \sum_{p \in L} a_R^{(p)} \cdot e_p, \tag{5.5}$$

де $e_p = \begin{cases} 1, \text{ якщо зразок } p \text{ класифіковано правильно;} \\ -1 \text{ — інакше.).} \end{cases}$

Залишаємо k нейронів правил з найвищими значеннями V_R та видаляємо інші нейрони правил із системи *NEFClass*;

в) «найкращий для кожного класу» алгоритм навчання: діємо так само, як і в попередньому випадку, але залишаємо для кожного класу C_j ті найкращі $[k/m]$ правил, наслідки яких становлять клас C_j (де $[x]$ — ціла частина x).

Етап 2. Навчання нечітких множин. На другому етапі відбувається навчання параметрів функцій належності (ФН) нечітких множин. Алгоритм навчання з учителем системи *NEFClass* має адаптувати ФН його нечітких множин; алгоритм циклічно проходить через усю навчальну вибірку L, виконуючи наведені нижче кроки, поки не виконається один із критеріїв зупинки [5.2, 5.4, 5.5, 5.7].

Кроки алгоритму:

1. Вибираємо зразок (p, t) з вибірки L, подаємо його в систему *NEFClass* і визначаємо вихідний вектор.

2. Для кожного вихідного нейрона c_i визначаємо помилку класифікації δ_{c_i}:

$$\delta_{c_i} = t_i - a_{c_i},$$

де t_i — бажаний вихід; a_{c_i} — фактичний вихід нейрона c_i.

3. Для кожного нейрона правил R, для якого вихід $a_R > 0$:

а) визначаємо значення δ_R, що дорівнює

$$\delta_R = a_R(1 - a_R) \sum_{C \in U_3} W(R, C)\, \delta_C; \tag{5.6}$$

б) знаходимо таке x', що

$$W(x', R)(a_{x'}) = \min_{x \in U_1}(W(x, R)(a_x)); \tag{5.7}$$

в) для нечітких множин $W(x', R)$ визначаємо зміщення параметрів ФН Δ_a, Δ_b, Δ_c, використовуючи швидкість навчання $\sigma > 0$, як показано

$$\Delta_b = \sigma\delta_R(c - a)\mathrm{sgn}(a_{x'} - b,); \tag{5.8}$$

$$\Delta_a = -\sigma\delta_R(c - a) + \delta_b; \tag{5.9}$$

$$\Delta_c = \sigma\delta_R(c - a) + \delta_b, \tag{5.10}$$

та виконуємо зміни $W(x', R)$;

г) обчислюємо похибку правила E:

$$E = a_R(1 - a_R)\sum_{C\in U_3}(2W(R, c) - 1)|\delta_c|. \tag{5.11}$$

Кінець ітерації. Повторюємо зазначені ітерації до виконання умов зупинки.

Як критерії зупинки можна взяти, наприклад, такі:

1) похибка протягом n ітерацій не зменшується;

2) припинити навчання після досягнення похибкою певного (бажано, близького до нуля) значення.

5.2. Аналіз властивостей *NEFClass*. Модифікована система нечіткої класифікації *NEFClass-M*

ННМ *NEFClass*

ННМ *NEFClass* має низку безперечних переваг, що виділяють її серед інших систем класифікації. Особливо слід зазначити такі, як простота реалізації, висока швидкість роботи алгоритмів навчання і найважливіше — висока точність класифікації даних — на рівні найкращих систем у цій галузі. Водночас базова система *NEFClass* має низку недоліків: використовувані формули для навчання параметрів мають емпіричний характер, крім того, незрозуміло, як вибирати в алгоритмі навчання параметр швидкості навчання σ. Наведемо запропоновану модифікацію базової моделі нечіткої нейронної системи *NEFClass-M* [5.2].

Рандомізація та ретельний підбір константи швидкості навчання σ є визначальними властивостями реалізації модифікованої системи *NEFClass*, подальший розвиток якої є однією з цілей цієї роботи. Ці властивості були покликані зменшити вплив деяких недоліків оригінальної моделі і дали змогу досягти помітного покращення якості класифікації.

Рандомізація. Завдяки особливостям алгоритму навчання «простих» правил і алгоритму навчання нечітких множин результати навчання мережі за цими алгоритмами значною мірою залежать від порядку, в якому подано зразки навчальної вибірки. Так, якщо порядок зразків буде відсортований за класами, то отримана в результаті система краще поводитиметься

на зразках одного класу і значно гірше — на зразках іншого. В ідеалі зразки в навчальній вибірці мають бути перемішані випадково, щоб уникнути цього негативного ефекту.

Реалізація системи *NEFClass-M* [5.2] обходить цю складність за допомогою випадкового «перемішування» порядку обробки зразків навчальної вибірки після завантаження. Більше того, таке випадкове «перемішування» відбувається перед кожною ітерацією алгоритму навчання нечітких множин. Як показують подальші експерименти, це дає можливість досягти стабільніших і найчастіше кращих результатів класифікації, що не залежать від того, в якому порядку зразки навчальної вибірки були подані користувачем.

Вибір швидкості навчання. В алгоритмі навчання нечітких множин моделі *NEFClass* фігурує швидкість навчання σ. Як показали експерименти, проведені в ході розроблення реалізації *NEFClass-M*, цей параметр відіграє виключно важливу, якщо не одну з визначальних, роль для успіху всього процесу навчання.

Проведені експерименти свідчать, що за інших однакових умов певного конкретного завдання навчання є певне значення, за якого досягається мінімальний відсоток помилкових класифікацій після навчання загалом. На жаль, аналітичну залежність отримати дуже важко, бо алгоритми навчання *NEFClass* загалом дуже емпіричні як такі. І все ж, методом спроб і помилок було встановлено, що для багатьох завдань оптимальне значення σ перебуває в інтервалі [0,05; 0,1]. Це значення було встановлено для програми, яка реалізує модифіковану модель *NEFClass-M* [5.2].

Модифікована модель *NEFClass*

Розглянемо основні недоліки алгоритму навчання у системі *NEFClass*.

Емпіричність алгоритму навчання нечітких множин. Оригінальний алгоритм навчання нечітких множин у системі *NEFClass*, описаний вище, є емпіричним, його (суб)оптимальність ніяк не обґрунтовано і не доведено його збіжність до шуканого рішення. Експерименти, наведені далі, показують, що хоча алгоритм дає результати, не надто далекі від оптимальних, усе ж таки його оптимальність для трикутних функцій належності можна поставити під сумнів.

Структура модифікованої моделі. Аналіз наведених недоліків свідчить, що їх причиною значною мірою є недосконалість алгоритму навчання нечітких множин *NEFClass*. Тому природним підходом, покликаним виправити ситуацію, стала заміна емпіричного алгоритму навчання на строгий алгоритм чисельної оптимізації з усіма наслідками для архітектури та алгоритмів мережі. Подальші експерименти підтвердили правильність саме такого рішення.

Як і оригінальна, так і модифікована модель *NEFClass* ґрунтуються на архітектурі нечіткого перцептрона [5.4]. Архітектурні відмінності оригінальної та модифікованої моделей складаються у вигляді функцій належності нечітких множин, функції *t*-норми для обчислення активацій нейронів правил, а також у вигляді функції, що агрегує (*t*-конорми) виходи правил.

Застосування чисельних методів оптимізації потребує диференційованості функцій належності нечітких множин — умова, яку трикутні функції не задовольняють. Тому в модифікованій моделі запропоновано використання гауссових функцій належності, які описуються так:

$$\mu(x) = \exp\left\{-\frac{(x-a)^2}{2b^2}\right\}.$$

Конкретна функція належності визначається, отже, двома параметрами — a і b.

Вимога диференційованості диктує також вид операції перетину для обчислення активації нейронів правил. У системі *NEFClass* для цього використовується функція мінімуму; у модифікації це добуток відповідних значень ФН входів правила [5.2].

Нарешті, вид агрегаційної функції (*t*-конорми) для модифікованої моделі обмежений лише зваженою сумою. Причина полягає в тому, що функція максимуму, яка використовується в оригінальній системі, не відповідає умові диференційованості.

Основна зміна стосується алгоритму навчання нечітких множин. Цільовою функцією в модифікованій системі *NEFClass* виступає мінімізація середньоквадратичної похибки на навчальній вибірці за аналогією з класичними (чіткими) нейромережами:

$$E = \frac{1}{N}\sum_{p=1}^{N}\left\|a_c^{(p)} - a_c^{(p)^*}\right\|,$$

де N — кількість зразків навчальної вибірки; $a_c^{(p)}$ — вектор активації нейронів вихідного шару для чергового навчального зразка p; $a_c^{(p)^*}$ — цільове значення цього вектора для даного патерну (зразка).

Компоненти цільового вектора для патерну p дорівнюють

$$a_c^{(p)^*} = \begin{cases} 0, & i \neq j, \\ 1, & i = j, \end{cases}$$

де c — номер класу, до якого належить цей патерн p.

Змінними чисельної оптимізації, спрямованої на зменшення середньоквадратичної похибки за навчальною вибіркою, є сукупний вектор параметрів a і b всіх нечітких множин мережі. Конкретним методом може бути будь-який метод безумовної оптимізації, як, наприклад, градієнтний метод або метод спряжених градієнтів; саме ці два методи було реалізовано у цій роботі.

5.3. Експериментальні дослідження ННМ *NEFClass*

Експерименти класифікації проводили на двох наборах даних — *IRIS* і *WBC*. Вибір їх як тестових наборів зумовлений двома міркуваннями: по-перше, ці набори можна вважати стандартними для завдань класифікації, по-друге, в оригінальних роботах авторів моделі *NEFClass* виконується тестування моделі саме на цих наборах даних. Це дає можливість порівняти результати роботи базової системи *NEFClass* з модифікованою *NEFClass-M*, а також простежити вплив рандомізації на процес навчання.

Набір даних *IRIS*. Набір *IRIS* містить 150 зразків, що складаються з трьох різних видів родини Півникових (*Iris setosa, Iris versicolour, Iris virginica*), по 50 зразків кожного виду. Кожен зразок характеризується чотирма властивостями [5.10].

Набір *IRIS* є єдиним за своєю класифікаційною простотою, для якого навіть звичайна стратегія відбору правил дає задовільні результати.

Для першого експерименту в модифікованій моделі *NEFClass-M* було обрано «простий» алгоритм навчання правил, число яких було обмежено 10, за кількості множин на змінну — 3 (для всіх інших параметрів встановлено значення за усталеним налаштуванням). У результаті система створила 10 правил і досягла результату в 4 помилкових класифікації з 150 (тобто 97,3 % правильних).

Найкращий результат, якого вдалося досягти за «простого» алгоритму навчання правил, — три правила за двох істотних змінних x_3 і x_4, з таким порядком помилкових класифікацій [5.1, 5.2]:

R1: IF (any, any, large, large) THEN Class 3;

R2: IF (any, any, medium, medium) THEN Class 2;

R3: IF (any, any, small, small) THEN Class 1.

Такий самий результат спостерігається і для «кращого», і «кращого на клас» алгоритмів навчання правил. Однак для останніх двох алгоритмів можливе подальше скорочення кількості множин розбиття для змінних x_3 та x_4 за таких правил (шість помилкових класифікацій):

R1: IF (any, any, small, small) THEN Class 1;

R2: IF (any, any, large, small) THEN Class 2;

R3: IF (any, any, large, large) THEN Class 3.

Авторами моделі *NEFClass* було отримано близькі результати, за винятком того, що у своїх експериментах вони зупинилися на трьох множинах, що розбивають x_3 і x_4. Наведені дані свідчать, що для ефективного вирішення проблеми насправді досить двох множин на змінну.

Отже, для набору даних *IRIS* вдалося досягти результату кращого, ніж в оригінальних роботах, — виключно простого набору правил з двома змінними з використанням двох нечітких множин для кожної змінної.

Набір даних *WBC*. Під час оброблення набору *Wisconsin Breast Cancer* у системі *NEFClass-M* було отримано цікаві результати, які не завжди збігаються (як правило, на краще) з результатами роботи моделі *NEFClass*.

Повторюючи хід експериментів авторів системи *NEFClass* [5.2, 5.5], для навчання системи було задано максимум із чотирьох правил з алгоритмом навчання «кращі на клас» (три множини на змінну). В результаті було отримано показники помилкової класифікації для системи *NEFClass-M* близько 28 із 663 (95,7 % правильних). Дуже цікавим є той факт, що у моделі *NEFClass* для аналогічних параметрів показник правильності становив лише 80,4 % (135 неправильних класифікацій). Цю істотну перевагу модифікованої системи *NEFClass-M* можна пояснити тим, що відрізняє реалізовану модель *NEFClass* від канонічної, а саме — використанням алгоритму рандомізації та вибору швидкості навчання.

Найкращий результат, який вдалося отримати для набору даних *WBC*, — база з восьми правил із п'ятьма суттєвими змінними x_1, x_2, x_4, x_6 і x_9 (помилкових класифікацій 19) [5.10]:

R1: IF (small, small, any, small, any, small, any, any, small) THEN Class 1;
R2: IF (small, small, any, large, any, small, any, any, small) THEN Class 1;
R3: IF (small, small, any, small, any, small, any, any, large) THEN Class 1;
R4: IF (large, large, any, small, any, large, any, any, small) THEN Class 2;
R5: IF (large, large, any, large, any, small, any, any, small) THEN Class 2;
R6: IF (small, large, any, small, any, large, any, any, small) THEN Class 2;
R7: IF (large, small, any, small, any, small, any, any, small) THEN Class 2;
R8: IF (large, small, any, small, any, small, any, any, large) THEN Class 2,

де Class 1 — доброякісні, Class 2 — злоякісні пухлини.

Порівнянні результати (24 помилкових класифікацій) дає використання всього двох правил («кращі на клас») з усіма суттєвими змінними, крім x_5 і x_7:

R1: IF (small, small, small, small, any, small, any, small, small) THEN Class 1;
R2: IF (large, large, large, small, any, large, any, large, small) THEN Class 2.

Наведені результати перевершують результати, отримані авторами базової моделі *NEFClass*, як за кількістю правил / істотних змінних, так і за точністю класифікації. Це знову підтверджує доцільність модифікацій, внесених до моделі *NEFClass-M*, — рандомізації та правильного вибору швидкості навчання.

5.4. Розпізнавання об'єктів електрооптичних зображень з використанням нечітких НМ

Загальна характеристика роботи системи. Дистанційне зондування є одним із ключових методів отримання зображень у всіх діапазонах електромагнітного спектра — від ультрафіолетового до інфрачервоного. Аналіз зображень, отриманих з метеорологічних і геостаціонарних супутників, які охоплюють значні площі, або детальних аерофотознімків ділянок, дає

змогу дійти висновку, що цей метод є основним інструментом для отримання якісної інформації про поверхню Землі та досліджувані об'єкти.

Зазвичай дистанційне зондування слугує ланкою вимірювань, фіксуючи параметри об'єктів або величини, пов'язані з ними. Наприклад, для оцінки стану сільськогосподарських посівів супутникова апаратура реєструє інтенсивність світлового потоку, відбитого від об'єктів у різних ділянках оптичного спектра. Для інтерпретації цих даних необхідно провести попередні дослідження, зокрема експерименти з вивчення стану посівів контактними методами, аналіз відбивної здатності води, ґрунту та рослин за різних умов освітлення, положення супутника та параметрів вимірювальної апаратури. Лише після цього можна зробити висновки про стан об'єктів або поверхні.

Методи дистанційного зондування належать до високих технологій завдяки не лише використанню сучасних оптоелектронних пристроїв і потужних комп'ютерів, а й новим підходам до інтерпретації результатів вимірювань. Хоча дослідження із супутників з невеликим кутом огляду є досить громіздкими, вони дають змогу комбінувати зображення, створюючи якісні широкомасштабні карти великих територій і навіть усієї земної кулі. Ширина охоплення поверхні під час сканування є однією з ключових переваг супутникових методів, що дає можливість отримувати якісні зображення за короткий час.

Інформація з супутників зазвичай надходить у цифровому вигляді. Системи дистанційного зондування складаються з декількох сенсорів, чутливих до світлових або звукових хвиль, і комп'ютера, який зчитує й записує дані. Сенсори встановлюють на рухомих платформах, що забезпечують огляд необхідної території. Наприклад, за допомогою системи з цифровою камерою для моніторингу морських хвиль отримують дані про глибину океану, тип рослинності та придатність земель для використання. Ці дані є безцінними для аналізу таких явищ як ерозія гір, зміни ґрунтів, урбанізація та зміна типу зеленого покриву Землі.

Дистанційне зондування дає також можливість проводити оперативні спостереження й створювати великі архіви аерофотознімків. Сьогодні майже вся поверхня суші та значна частина водної поверхні зафіксовані космічними знімками, отриманими за різних умов спостереження (різні сезони, час доби тощо).

Концепція мультиспектральних електрооптичних систем

Основні визначення. Концепцію мультиспектральної системи наведено на рис. 5.2.

Система знімає зображення поверхні Землі за допомогою сенсора з двовимірними детекторними ґратками. У результаті дані є практично неперервним спектром для кожного елемента зображення (пікселя). Схема-

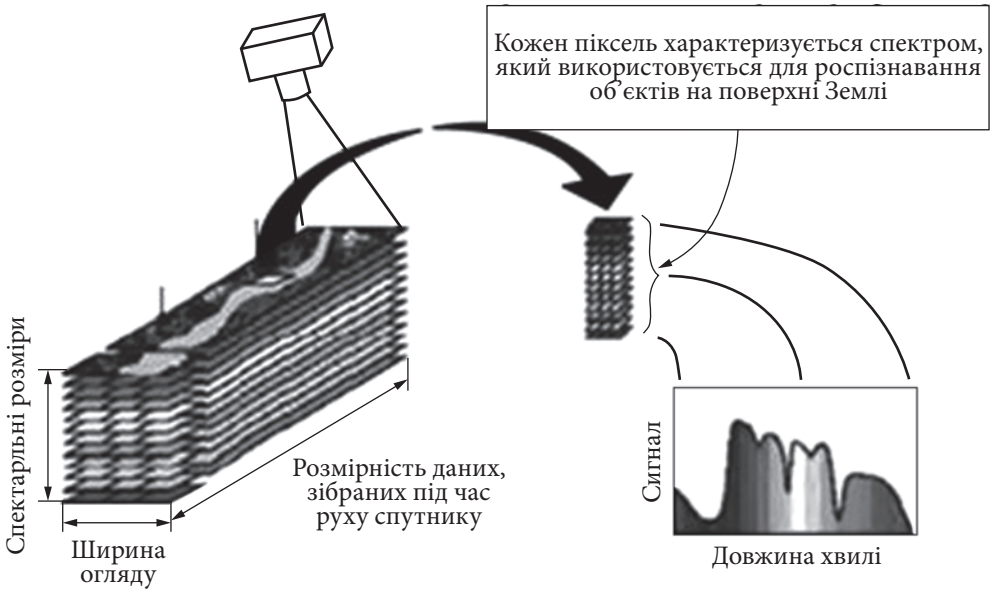


Рис. 5.2. Концепція мультиспектральної системи

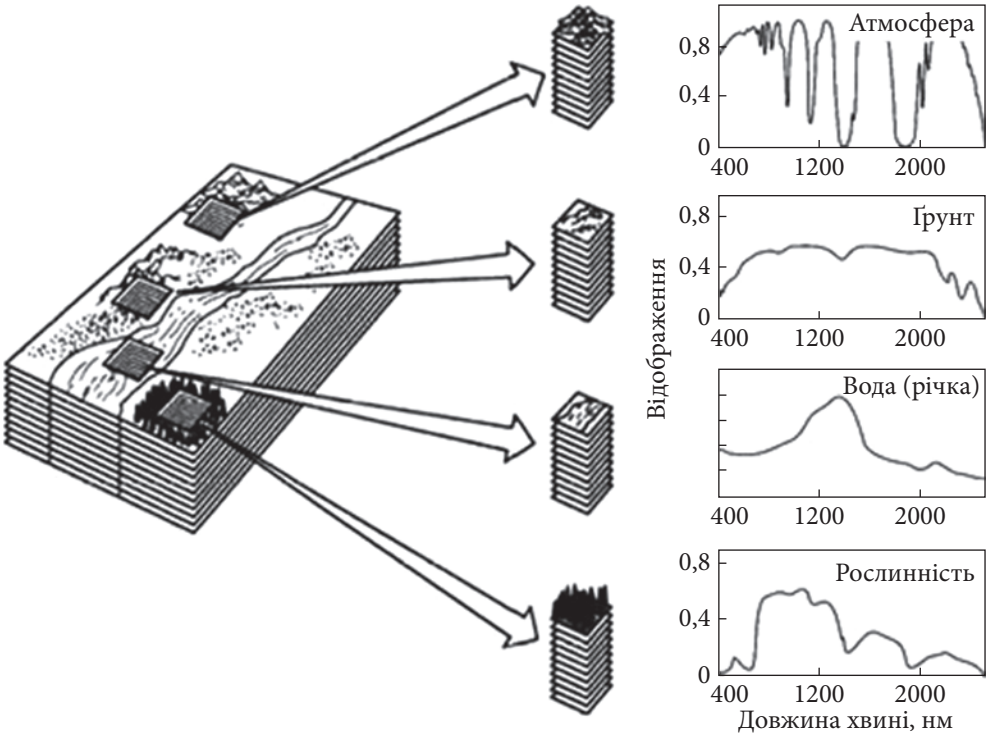


Рис. 5.3. Приклад розпізнавання об'єктів за формою спектральної кривої

тично складові системи можна подати у вигляді 3D-куба — тривимірного набору даних, зображення, що характеризуються трьома ознаками — *x*, *y* і *z*-координатами і довжиною хвилі (спектром) розділяючої [5.10].

Спектральна крива відображує залежність між довжиною хвилі та енергією і відбивну здатність об'єкта спостереження. За формою спектральної кривої можна розрізняти об'єкти. Наприклад, рослинність має високу відбивну здатність у ближньому інфрачервоному діапазоні і низьку — в середньому порівняно з ґрунтом. Приклад розпізнавання об'єктів за формою спектральної кривої наведено на рис. 5.3.

За допомогою бортових комп'ютерів отримано зображення, що передаються до наземних станцій для подальшого оброблення. Алгоритм оброблення зображення полягає у калібруванні, картографуванні виправлень геометричного викривлення та класифікації поверхонь. У деяких випадках використовується алгоритм оброблення даних у реальному часі.

Види сенсорів. Мультиспектральні та гіперспектральні системи

Мультиспектральна система — це система, що має на борту супутника або літака сенсор, який знімає зображення у певному спектральному діапазоні і налічує від 2 до 15 спектральних частот. Частотна смуга визначається як частина спектра (кольорів) з певною шириною, наприклад 10 або 50 нм [5.10].

Приклад спектрального діапазону мультиспектрального сенсора наведено на рис. 5.4. Мультиспектральний набір даних характеризується 5—10 смугами з великою шириною спектра — 70—400 нм. Зазвичай у мультиспектральних системах частотні смуги спектра зображень знаходяться поруч одна з одною, тому є суміжними. Вони можуть бути широкими або вузькими, а значна кількість — дуже вузькими (близько 10 нм). Сучасні мультиспектральні зображення мають більш високе спектральне розширення. Наприклад, у сенсорі *«Aster»* спектральне розширення для ближнього діапазону становить від 0,07 до 0,17 мкм. Чим вужчий спектральний діапазон, до якого чутливий сенсор, тим вужче спектральне розширення.

Принцип роботи ***гіперспектральних систем*** такий самий, як і мультиспектральних, але вони відрізняються кількістю спектральних частот. Гіперспектральні системи налічують десятки сотень вузьких суміжних частотних смуг. Взагалі такі системи мають від 100 до 200 частот на відносно невеликій ширині діапазону (5—10 нм). Приклад спектрального діапазону гіперспектрального сенсора наведено на рис. 5.5.

Спектральні криві зображення зазвичай починаються з 400 нм від кінця видимого спектра. Такі системи вимірюють діапазон до 1100 і навіть 2500 нм.

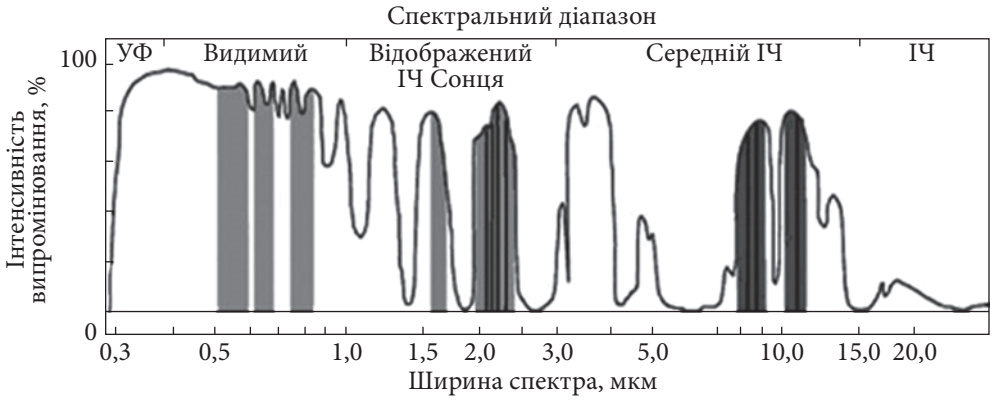


Рис. 5.4. Взаємозв'язок спектрального діапазону та спектрального розширення

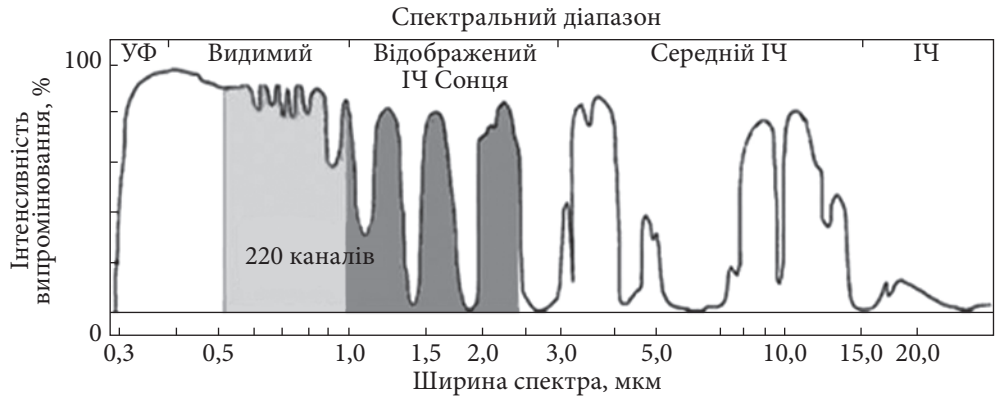


Рис. 5.5. Приклад спектрального діапазону гіперспектрального сенсора

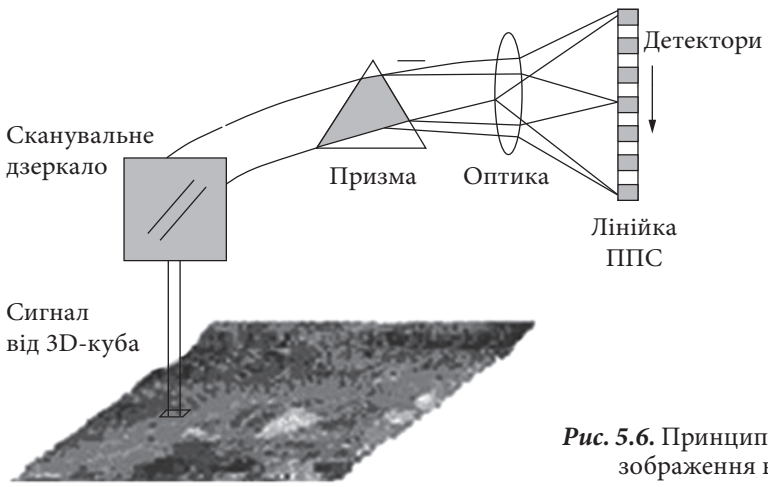


Рис. 5.6. Принципова схема побудови зображення в спектрометрі

Принципи формування зображень системи

У мульти- та гіперспектральній системах електромагнітне випромінювання розбивається за допомогою призми на безліч вузьких сумісних смуг. Енергія у кожній смузі вимірюється окремим детектором. На рис. 5.6 зображено принципову схему побудови зображення в сенсорі — спектрометрі.

За допомогою сканувального дзеркала, призми та набору оптичних лінз зображення надходить до приймально-синхронізувальної системи (ПСС) або детекторної матриці, з якої зображення спрямовується до комп'ютера, записується на жорсткий диск і передається для подальшого оброблення.

5.5. Застосування системи *NEFClass* у завданні розпізнавання об'єктів електрооптичних зображень

За допомогою мультиспектральної електрооптичної системи, що працює у трьох діапазонах — червоному, зеленому та синьому, було отримано знімки прибережної смуги океану. Потрібно було розпізнати об'єкти у вигляді геометричних фігур на водній поверхні та на піску. Для цього, враховуючи складність завдання, а також великий рівень перешкод, запропоновано використовувати нечіткі нейронні мережі, зокрема *NEFClass* [5.9].

З метою організації навчання ННМ *NEFClass* було розроблено низку алгоритмів: градієнтний, спряжених градієнтів та генетичний [5.9], проведено дослідження їх ефективності та порівняння з базовим алгоритмом навчання системи *NEFClass*, запропонованим Д. Науком та Р. Крузе [5.7], який описано у п. 5.1.

Градієнтний алгоритм навчання у системі *NEFClass*. Для першого етапу цього алгоритму навчання бази правил було використано перший етап базового алгоритму *NEFClass*. Другий етап використовує градієнтний алгоритм навчання нейронної мережі прямої дії, який описано далі.

Нехай критерій навчання нечіткої нейромережі, яка має три шари (один прихований шар), такий:

$$e(W) = \sum_{i=1}^{M} (t_i - NET_i(W))^2 \to \min, \tag{5.12}$$

де t_i — бажане значення i-го виходу нейромережі; $NET_i(W)$ — фактичне значення i-го виходу нейромережі для вагової матриці $W = [W^I, W^0]$, $W^I = W(x, R) = \mu_j(x)$, $W^0 = W(R, C)$. Тобто критерій $e(W)$ є середнім квадратом похибки апроксимації.

Нехай функції активації для нейронів прихованого шару (нейронів правил) становлять

$$O_R = \prod_{i=1}^{N} \mu_{j_i}^{(i)}(x_i), \quad j = 1, \ldots, q_i, \tag{5.13}$$

де $\mu_{j_i}^{(i)}(x_i)$ — функція належності, яка має вигляд

$$\mu_{j_i}^{(i)}(x_i) = e^{\frac{-(x-a_{ji})^2}{b_{ji}^2}}, \tag{5.14}$$

а функція активації нейронів вихідного шару (зважена сума) дорівнює

$$O_C = \frac{\sum_{R\in U_2} W(R, C)\cdot O_R}{\sum_{R\in U_2} W(R, C)}, \tag{5.15}$$

або функція максимуму

$$O_C = W(R, C)\cdot O_R. \tag{5.16}$$

Розглянемо градієнтний алгоритм навчання нечіткого перцептрона [5.10].

1. Нехай $W(n)$ — поточне значення матриці ваги. Алгоритм має такий вигляд:

$$W(n+1) = W(n) - \gamma_{n+1}\nabla_w e(W(n)), \tag{5.17}$$

де γ_n — розмір кроку на n-й ітерації; $\nabla_w e(W(n))$ — градієнт критерію (5.12).

2. На кожній ітерації спочатку навчаємо (коригуємо) вхідні ваги W (вирази (5.18)—(5.19)), які залежать від параметрів a і b (див. ФН (5.14)):

$$a_{ji}(n+1) = a_{ji}(n) - \gamma_{n+1}\frac{\partial e(W)}{\partial a_{ji}}; \tag{5.18}$$

$$b_{ji}(n+1) = b_{ji}(n) - \gamma'_{n+1}\frac{\partial e(W)}{\partial b_{ji}}, \tag{5.19}$$

де γ'_{n+1} — розмір кроку для параметра b;

$$\frac{\partial e(W)}{\partial a_{ji}} = -2\sum_{k=1}^{M}((t_k - NET_k(w))W(R, C)\cdot O_R\frac{(x-a_{ji})}{b_{ji}^2}; \tag{5.20}$$

$$\frac{\partial e(W)}{\partial b_{ji}} = -2\sum_{k=1}^{M}((t_k - NET_k(w))W(R, C)\cdot O_R\frac{(x-a_{ji})^2}{b_{ji}^3}. \tag{5.21}$$

3. Знаходимо (навчаємо) вихідні ваги:

$$\frac{\partial e(W^0)}{\partial W(R, C)} = -((t_k - NET_k(W^0))\cdot O_R; \tag{5.22}$$

$$W_k^0(n+1) = W_k^0(n) - \gamma''_{n+1}\frac{\partial e(W^0)}{\partial W(R, C_k)}. \tag{5.23}$$

4. $n := n + 1$ і переходимо на наступну ітерацію.

Градієнтний метод є першим запропонованим алгоритмом навчання, він простий у реалізації, але має такі недоліки:

а) повільно сходиться;

б) знаходить лише локальний екстремум.

Крім градієнтного методу навчання було реалізовано метод спряжених градієнтів та генетичний алгоритм (див. розд. 1).

5.6. Експерименти з розпізнавання об'єктів на електрооптичних зображеннях

Для вибору та оброблення даних електрооптичних зображень використовують систему *ENVI* та її можливість картографувати, тобто поєднувати зображення за контрольними точками (зображення, отримані з різних спектральних камер) [5.9]. Це дає можливість отримати мультиспектральні зображення. Вихідні дані для картографування наведено на рис. 5.7, *а*.

Після вибору 15 контрольних точок на зображеннях різних спектрів (ця функція не автоматизована) зображення поєднують і отримують так званий мультиспектральний куб. Результат (мультиспектральні зображення) наведено на рис. 5.7, *б*.

Було визначено дев'ять типів різних поверхонь, які потрібно класифікувати. Для визначення характеристик використовували так звані зони інтересу *ROI (Region of Interest)*. На зображенні визначали однорідну зону, наприклад пісок, вода, піна, ціль червоного кольору, ціль білого кольору тощо. Результат такого виділення зображення з *ROI* показано на рис. 5.8.

Далі, використовуючи систему оброблення, обчислюють середнє значення для вибраного регіону та дисперсію. Дані, отримані в такий спосіб, було зведено в таблицю. Ці дані характеризують дев'ять класів поверхні:

- біла ціль; червона ціль; зелена ціль; синя ціль; жовта ціль;
- піна; вода; сухий пісок; мокрий пісок.

Для класифікації об'єктів запропоновано використовувати ННМ *NEFClass-M*. Ці види поверхонь відповідають дев'ятьом вихідним вузлам у системі *NEFClass-M*.

Загальна кількість ознак, які використовують для класифікації видів поверхонь, становить чотири, а саме:

- яскравість у червоному спектрі (ЧС);
- яскравість у блакитному спектрі (БС);
- яскравість у зеленому спектрі (ЗС);
- яскравість в інфрачервоному діапазоні (ІС).

Загальна кількість даних становить 99, по 11 на кожен клас.

Наведемо основні статистичні характеристики для набору даних, отриманих за допомогою мультиспектральної системи *Mantis* (табл. 5.1, 5.2) [5.2].

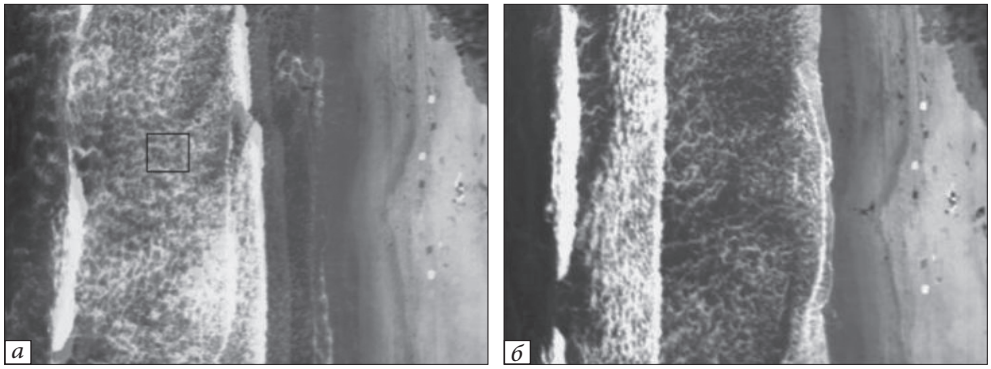
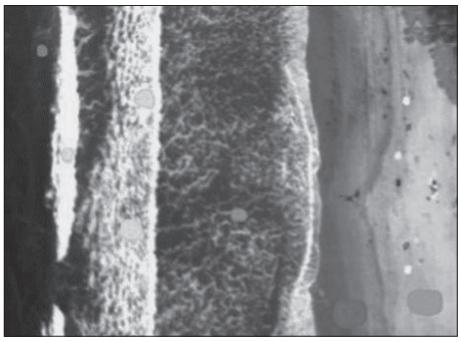


Рис. 5.7. Отримання мультиспектральних зображень: *а* — вхідні дані; *б* — результат

Рис. 5.8. Результат виділення зображення ► з *ROI*



Таблиця 5.1

Основні статистичні характеристики для набору даних, отриманих за допомогою мультиспектральної системи *Mantis*

Яскравість, відносні одиниці	Мінімум	Максимум	Середнє	Стандартне відхилення	Кореляція між ознаками та класом
ЧС	28,81	255,00	165,40	76,14	–0,46
БС	72,93	255,00	165,43	68,62	–0,32
ЗС	44,34	254,89	121,57	57,64	–0,52
ІС	17,03	255,00	140,84	81,58	–0,49

Таблиця 5.2

Кореляційна матриця

Яскравість, відносні одиниці	Яскравість, відносні одиниці			
	ЧС	БС	ЗС	ІС
ЧС	1	0,7	0,58	0,95
БС	—	1	0,77	0,7
ЗС	—	—	1	0,59
ІС	—	—	—	1

Для дослідження ефективності різних алгоритмів навчання системи *NEFClass* у завданні розпізнавання електрооптичних зображень було розроблено програму *NEFClass BGCGG (Basic Gradient Conjugate Gradient Genetic)* [5.2].

Далі було проведено експерименти з програмою моделювання *NEFClass BGCGG*. Відповідно до основного принципу дослідження моделей, експерименти проводили зі зміною послідовно лише одного параметра. З наявних 99 зразків 54 завантажували як навчальну вибірку. Інші 45 зразків було використано для тестування.

Значення основних параметрів алгоритму моделювання було встановлено у вихідне положення (табл. 5.3).

У процесі навчання було згенеровано 15 правил нечіткого класифікатора, наведених у табл. 5.4 [5.2].

Було досліджено залежність якості навчання від кількості правил, що генеруються на першому етапі. Для об'єктивної оцінки результатів проведемо тестування на перевірній вибірці. Для цього змінимо кількість правил, починаючи від 9 до 14. Результати дослідження залежності якості навчання від кількості правил наведено в табл. 5.5. Результат закономірний: чим більше правил, тим кращий результат тестової класифікації.

Було досліджено вплив кількості термів у ознаках на якість навчання. Порівняльний аналіз залежності якості навчання від кількості термів наведено в табл. 5.6. Дуже цікавий результат було отримано в цій серії дослідів. З даних табл. 5.6 випливає, що є оптимальна кількість термів, які можна використати для опису набору даних під час навчання. У разі збільшення кількості термів

Таблиця 5.3

Значення параметрів алгоритму *NEFClass BGCGG*

Параметр	Значення
Алгоритм генерації правил	Найкращий для класу
Алгоритм навчання	Класичний
Кількість згенерованих правил	Максимальна
Функція агрегації	Виважена сума
Навчання зважених коефіцієнтів між шаром правил та вихідним шаром	Вільне
Кількість термів (значень) для кожної ознаки	5 для всіх ознак
Обмеження перетину нечітких множин	1
Швидкість навчання для вагових коефіцієнтів між ознаками та правилами	$\sigma_a = 0{,}1$; $\sigma_b = 0{,}1$; $\sigma_c = 0{,}1$
Швидкість навчання для вагових коефіцієнтів між правилами та вихідним шаром	$\sigma = 0{,}1$
Максимальна кількість епох	50

Таблиця 5.4

Згенеровані правила нечіткого класифікатора

Номер правила	Номер ФН для ознаки 1	Номер ФН для ознаки 2	Номер ФН для ознаки 3	Номер ФН для ознаки 4	Номер класу
1	4	4	4	4	0
2	4	0	1	4	1
3	4	0	0	4	1
4	4	1	0	4	1
5	2	3	1	1	2
6	1	0	1	0	3
7	4	4	1	4	4
8	3	4	3	3	5
9	3	3	2	3	5
10	4	4	3	3	5
11	0	0	0	0	6
12	3	2	1	2	7
13	1	0	0	1	8
14	1	1	0	1	8
15	1	0	0	0	8

Таблиця 5.5

Результати дослідження залежності якості навчання від кількості правил

Кількість правил	СКП	ПК, %
9	13,071009	24
10	9,545608	15
11	9,910701	15
12	9,705482	15
13	4,769655	4
14	4,739224	4
15	4,751657	4

Таблиця 5.6

Порівняльний аналіз залежності якості навчання від кількості термів

Кількість термів	СКП	ПК, %
4	5,928639	4
5	4,626252	4
6	4,957257	4
7	5,228448	4
8	5,633563	4
9	6,797175	4
10	7,897521	7

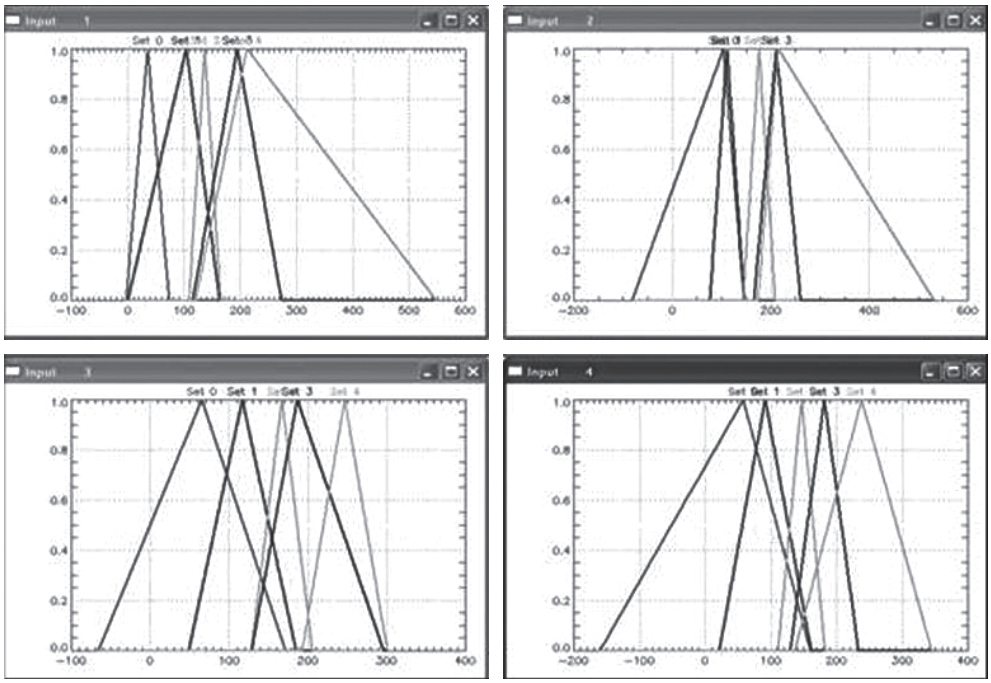


Рис. 5.9. Вигляд функції належності для кожної ознаки за класичним алгоритмом Наука — Крузе

понад це значення зростає кількість помилково класифікованих зразків, тобто з підвищенням складності моделі збільшується помилка (тобто перенавчання).

Здійснено навчання системи класичним алгоритмом з оптимальною кількістю термів у ознаках. Вигляд функції належності для кожної ознаки у процесі навчання класичним алгоритмом наведено на рис. 5.9. Підсумкова сума квадратів похибок становила 2,852081, кількість помилкових класифікацій (ПК) — нуль на навчальній вибірці; на тестовій, як ми бачили, частка помилкових класифікацій становила 4 %, середньоквадратична похибка (СКП) дорівнює 4,6252, що є хорошим результатом.

Проведено також експерименти за градієнтним алгоритмом. Отримані результати наведено на рис. 5.10 (графіки налаштованих ФН нечітких множин для кожної з чотирьох ознак).

Похибка наприкінці навчання становила 2,042015, що дещо краще порівняно з класичним методом. Під час тестування СКП становила 3,786005, частка помилкової класифікації — 4 %.

Було застосовано автоматичне налаштування швидкості адаптації параметрів ФН, для чого використано алгоритм «золотого перетину». Результати роботи градієнтного алгоритму в тандемі із «золотим перетином» наведено на рис. 5.11.

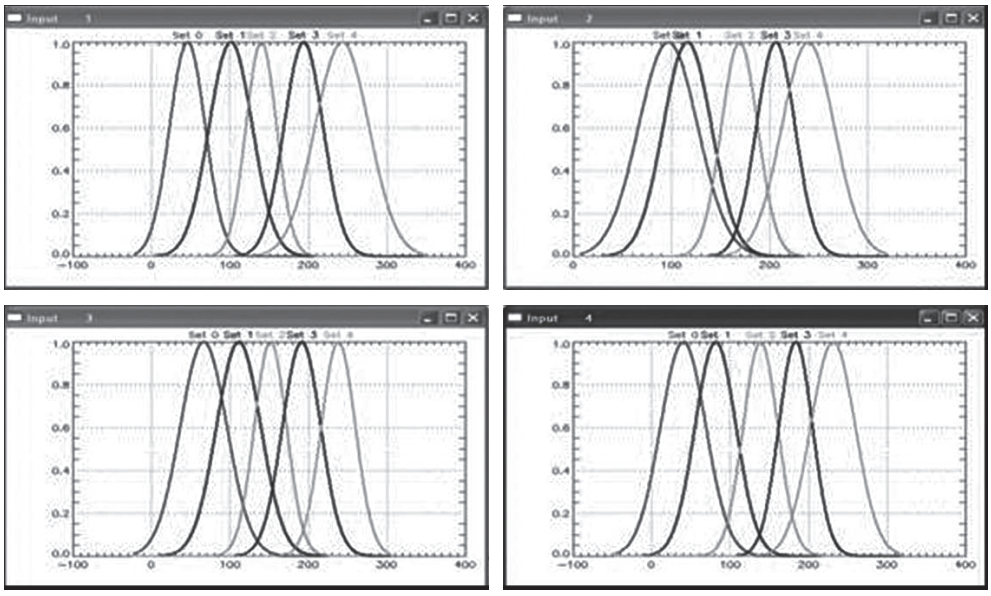


Рис. 5.10. Вигляд функції належності для кожної ознаки за градієнтним алгоритмом

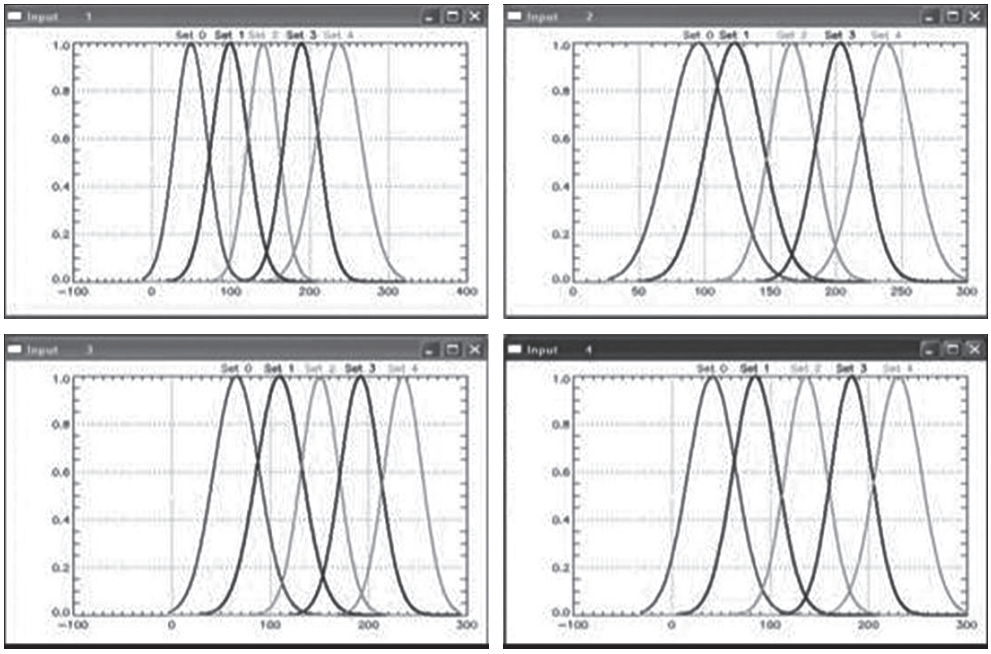


Рис. 5.11. Вигляд функції належності для кожної ознаки за градієнтним алгоритмом із «золотим перетином»

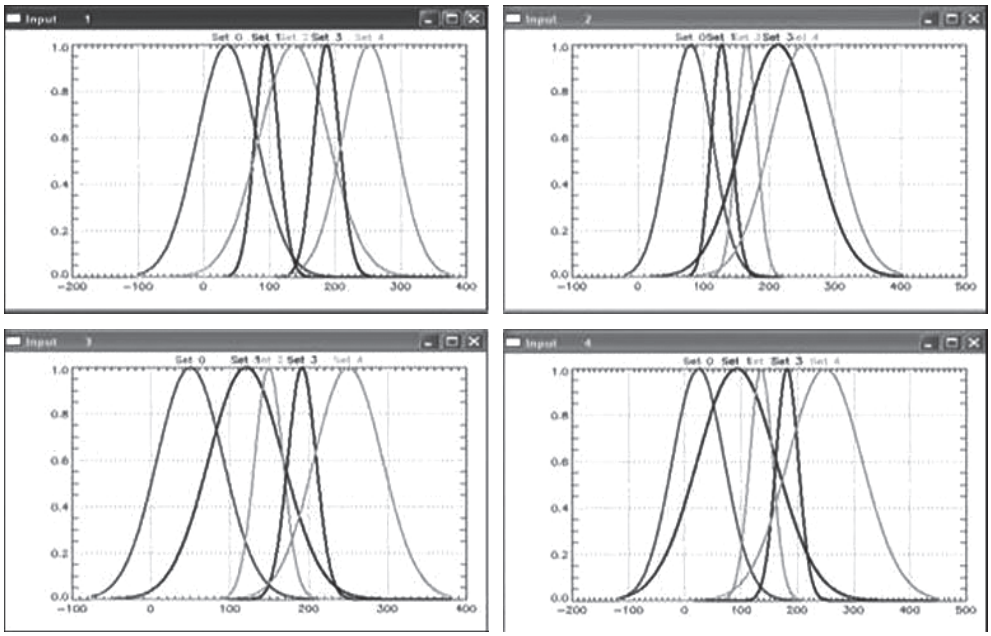


Рис. 5.12. Вигляд функції належності для кожної ознаки за алгоритмом спряжених градієнтів

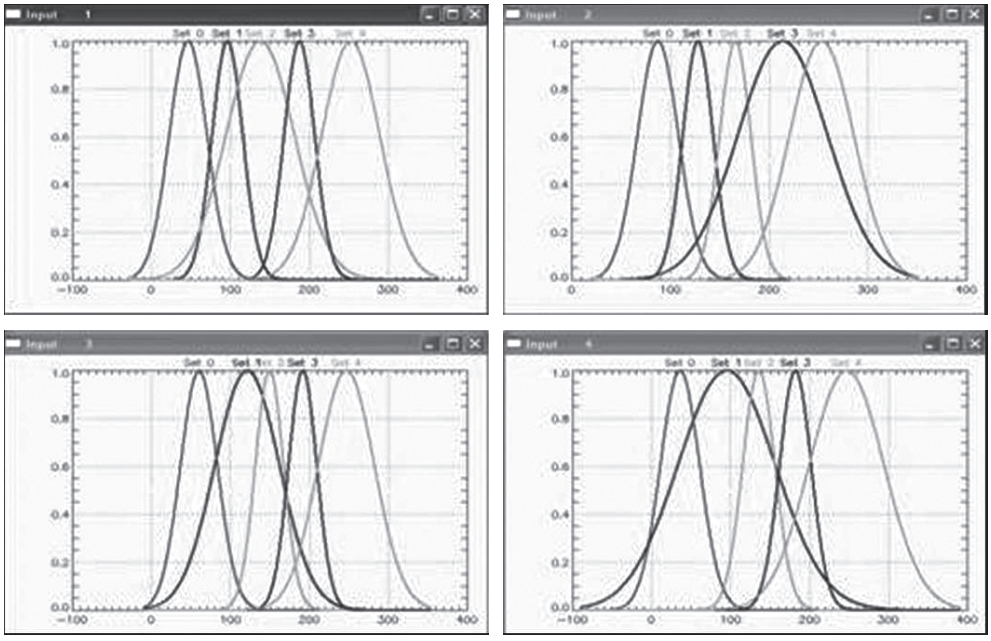


Рис. 5.13. Вигляд функції належності для кожної ознаки за алгоритмом спряжених градієнтів із «золотим перетином»

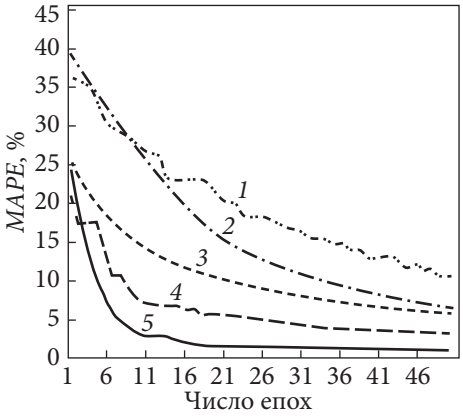


Рис. 5.14. Порівняння графіків швидкості збіжності різних алгоритмів навчання

Далі було проведено такі самі експерименти за алгоритмом спряжених градієнтів. Результати навчання методом спряженого градієнта наведено на рис. 5.12 [5.2].

Застосуємо метод «золотого перетину». Отримані результати навчання методом спряженого градієнта з вибором кроку методом «золотого перетину» показано на рис. 5.13.

На закінчення було проведено експерименти з генетичним алгоритмом з різними ФН — трикутною та гауссовою. Результати навчання ваг за різними алгоритмами наведено на рис. 5.14, де зображено порівняння графіків швидкості збіжності до оптимальної класифікації різних алгоритмів навчання, і в табл. 5.7.

Зауважимо, що для навчальної вибірки для всіх алгоритмів було отримано відмінні результати за критерієм «помилкової класифікації» (ПК) у відсотках. Для всіх алгоритмів цей критерій дорівнює нулю. Однак під час тестування на перевірній вибірці результати виявилися гіршими: як мінімум два зразки помилково класифіковано [5.2]. Також зросла сума квадратів похибок (СКП) за всіх без винятку алгоритмів навчання. Для зручності порівняння кількість ітерацій обмежено до 50.

Таблиця 5.7

Результати навчання ваг за різними алгоритмами

Алгоритм навчання вагових коефіцієнтів	Навчання		Тестування	
	СКП	ПК, %	СКП	ПК, %
Класичний	6,650668	0	7,285827	4
Градієнтний	5,9893	0	6,829068	4
Спряжених градієнтів	1,132871	0	3,314763	4
Генетичний з трикутною ФН	11,110936	0	13,677424	4
Генетичний з гауссовою ФН	3,204446	0	4,568338	4

Як бачимо, результати є цілком задовільними, рівень правильної класифікації на перевірній частині вибірки становить 96 %. Ці результати можна покращити, сформувавши більш представницьку вибірку.

На графіках рис. 5.14 добре видно, що найкращим методом щодо швидкості збіжності є метод спряжених градієнтів (крива *5*). Далі йде генетичний з гауссовою функцією належності (крива *4*). Менш ефективними є градієнтний (крива *3*) та класичний (крива *2*) методи, що використовуються в системі *NEFClass*. І найменш ефективний — генетичний спосіб (крива *1*) із трикутною функцією належності.

Проте критерій СКП, для якого побудовано графіки, неоднозначно відображає якість класифікації. Важливим критерієм оцінки методів є мінімальна кількість помилково класифікованих зразків на тестовій вибірці. З даних табл. 5.7 випливає, що всі алгоритми дали однакові результати щодо цього критерію.

5.7. Метод опорних векторів

Вище було розглянуто нечіткий метод класифікації *NEFClass*, а також приклади його використання в популярних задачах класифікації. Проте крім нечітких методів розроблено також чимало чітких методів класифікації. Серед цієї групи чітких класичних методів класифікації гідне місце посідає метод опорних векторів *(Support Vector Machine, SVM)*, який багато років використовується у різних завданнях класифікації [5.18—5.22].

Розглянемо теорію та застосування цього методу.

Введемо основні позначення. Розглядається завдання навчання за прецедентами (X, Y, y^*, X^L), де X — простір об'єктів, Y — множина відповідей, $y^* : X \to Y$ — цільова залежність, значення якої відомі тільки на об'єктах навчальної вибірки $X^L = (x_i, y_i)_{i=1}^L$, $y_i = y^*(x_i)$. Потрібно побудувати алгоритм $A : X \to Y$, що апроксимує цільову залежність на всьому просторі X.

Метод опорних векторів у задачах класифікації

Розглянемо задачу класифікації на два класи, що не перетинаються, в якій об'єкти описуються n-вимірними дійсними векторами: $X \in \mathbb{R}^n$, $Y = \{-1, +1\}$. Будемо будувати лінійний пороговий класифікатор:

$$A(X) = \text{sign}(\sum_{j=1}^{n} w_j x^j - w_0) = \text{sign}(\langle w, x\rangle - w_0), \qquad (5.24)$$

де $X = (x^1, \ldots, x^n)$ — ознаковий опис об'єкта x; вектор $w = (w^1, \ldots, w^n) \in \mathbb{R}^n$ і скалярний поріг $w_0 \in \mathbb{R}^n$ є параметрами алгоритму.

Рівняння $\langle w, x\rangle = w^0$ описує гіперплощину, що розділяє класи у просторі $\mathbb{R}^n$.

Хоча правило класифікації точно збігається з моделлю нейрона за Маккалоком — Пітсом (Уоррен Маккалок; Warren McCulloch; Волтер Піттс; Walter Pitts), критерій і методи налаштування параметрів у *SVM* радикально відрізняються від градієнтних методів навчання.

Поняття оптимальної відокремлювальної гіперплощини. Припустимо, що вибірку лінійно можна розділити, тобто є такі значення параметрів w, w_0, за яких функціонал числа помилок

$$Q(w, w_0) = \sum_{i=1}^{l} [y_i (\langle w, x\rangle - w_0) < 0] \tag{5.25}$$

набуває нульового значення. Однак тоді розділова гіперплощина не єдина, оскільки є й інші положення такої гіперплощини, що реалізують те саме розбиття вибірки. Ідея методу полягає в тому, щоб розумно використати цю свободу вибору.

Потрібно, щоб розділова гіперплощина була максимально далеко від найближчих до неї точок обох класів. Спочатку цей принцип класифікації виник з евристичних міркувань: цілком природно вважати, що максимізація зазору (*margin*) між класами має сприяти більш чіткій класифікації. Надалі цей принцип набув суттєвого теоретичного обґрунтування [5.10, 5.16, 5.19].

Нормування. Зауважимо, що параметри лінійного порогового класифікатора визначено з точністю до нормування: алгоритм $A(x)$ не зміниться, якщо w і w_0 одночасно множити на ту саму позитивну константу. Зручно вибрати цю константу так, щоб для всіх граничних (тобто найближчих до розділової гіперплощини) об'єктів x_i з X^L виконувались умови $\langle w, x\rangle - w_0 = y_i$.

Зробити це можливо, оскільки за оптимального положення гіперплощини, що розділяє, всі граничні об'єкти розміщені на однаковій відстані від неї. Інші об'єкти знаходяться далі. Отже, для всіх $x_i \in X^L$

$$\langle w, x\rangle - w_0 = \begin{cases} \leq -1, \text{ якщо } y_i = -1; \\ \geq 1, \text{ якщо } y_i = +1. \end{cases} \tag{5.26}$$

Умова $-1 < \langle w, x\rangle - w_0 < 1$ задає смугу, яка розділяє класи. Жодна з точок навчальної вибірки не може перебувати всередині цієї смуги. Межами смуги слугують дві паралельні гіперплощини з напрямним вектором w. Точки, найближчі до розділової гіперплощини, лежать точно на межах смуги. При цьому сама гіперплощина проходить рівно посередині смуги.

Ширина розділової смуги. Щоб гіперплощина була якомога далі від точок вибірки, ширина смуги має бути максимальною.

Нехай x_- і x_+ — дві довільні точки класів -1 та $+1$ відповідно, що лежать на межі смуги. Тоді ширина смуги є

$$\langle (x_+ - x_-), \frac{w}{||w||} = \frac{\langle w, x_+\rangle - \langle w, x_-\rangle}{||w||} = \frac{(w_0 + 1) - (w_0 - 1)}{||w||} = \frac{2}{||w||}.$$

Ширина смуги максимальна, коли норма вектора w мінімальна.

Отже, у разі, якщо вибірка лінійно подільна, досить прості геометричні міркування приводять до такої задачі: потрібно знайти такі значення параметрів w і w_0, за яких норма вектора w мінімальна за умови (5.26). Це задача квадратичного програмування. Її докладно розглянуто далі.

Лінійно відокремлювана вибірка

Побудова оптимальної розділової гіперплощини зводиться до мінімізації квадратичної форми за L обмежень-нерівностей

$$\begin{cases} \langle w, w\rangle \to \min; \\ y_i(\langle w, x_i\rangle - w_0) \geq 1, \quad i = 1, ..., L. \end{cases} \tag{5.27}$$

Згідно з теоремою Куна — Такера (Гарольд Вільям Кун — Harold William Kuhn; Альберт Вільям Такер — Albert William Tucker), ця задача еквівалентна двоїстій задачі пошуку сідлової точки функції Лагранжа (Жозеф-Луї Лагранж — Joseph-Louis Lagrange):

$$\begin{cases} \mathrm{L}(w, w_0; \lambda) = \dfrac{1}{2}\langle w, w\rangle - \sum_{i=1}^{L}(\lambda_i(y_i(\langle w, x_i\rangle - w_0) - 1) \longrightarrow \min\limits_{w, w_0}\max\limits_{\lambda}; \\ \lambda_i \geq 0, \, i = 1, ..., L; \\ \lambda_i = 0, \text{ або } \langle w, x_i\rangle - w_0 = y_i, \, i = 1, ..., \mathrm{L}, \end{cases} \tag{5.28}$$

де $\lambda = (\lambda_1, \ldots, \lambda_L)$ — вектор двоїстих змінних. Остання з трьох умов називається умовою комплементарної нежорсткості.

Обов'язковою умовою для сідлової точки є те, що похідні лагранжіана дорівнюють нулю. Звідси відразу випливають два корисних співвідношення:

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^{L}\lambda_i y_i x_i = 0; \quad \mathrm{w} = \sum_{i=1}^{L}\lambda_i y_i x_i; \tag{5.29}$$

$$\frac{\partial L}{\partial w_o} = -\sum_{i=1}^{L}\lambda_i y_i = 0 \implies \sum_{i=1}^{L}\lambda_i y_i = 0;. \tag{5.30}$$

З (5.29) випливає, що шуканий вектор ваг w є лінійною комбінацією векторів навчальної вибірки, і тільки тих, для яких $\lambda_i \neq 0$. Згідно з умовою доповнювальної нежорсткості, на цих векторах x_i обмеження-нерівності переходять у рівняння $\langle w, x_i\rangle - w_0 = y_i$, відповідно, ці вектори є на кордоні розділової смуги. Всі інші вектори будуть знаходитися далі від межі, для них $\lambda_i = 0$ і вони не роблять свого внеску в суму (5.29). Алгоритм не змінився б, якби цих векторів узагалі не було в навчальній вибірці.

Визначення 5.1. Якщо $\lambda_i > 0$ і $\langle w, x_i\rangle - w_0 = y_i$, то об'єкт навчальної вибірки x_i називається *опорним вектором*.

Підставляючи (5.29) і (5.30) знову в лагранжіан, отримаємо еквівалентну задачу квадратичного програмування, яка містить лише двоїсті змінні:

$$\begin{cases} -\mathrm{L} = -\sum_{i=1}^{L} \lambda_i + \frac{1}{2} \sum_{i=1}^{L} \left(\sum_{j=1}^{L} \lambda_i \lambda_j y_i y_j \left(\langle x_i, x_j \rangle \right) \right) \longrightarrow \min_{\lambda}; \\ \lambda_i \geq 0,\ i = 1, \ldots, L; \\ \sum_{i=1}^{L} \lambda_i y_i = 0. \end{cases} \tag{5.31}$$

Тут квадратичний функціонал зведений до мінімуму, що має невід'ємну певну квадратичну форму, а значить, визначає опуклу. Ділянка, визначена обмеженнями-нерівностями та однією рівністю, також є опуклою. Відповідно, ця проблема має єдине рішення.

Припустимо, ми розв'язали цю задачу. Тоді вектор w обчислюється за формулою (5.29). Для визначення порога w_0 досить узяти довільний опорний вектор x_i і виразити w_0 з рівняння $w_0 = \langle w, x_i \rangle - y_i$. На практиці з метою підвищення чисельної стійкості рекомендується приймати за w_0 середнє арифметичне всіх опорних векторів, а ще краще — медіану:

$$w_0 = \mathrm{med}\{\langle w, x_i \rangle - y_i : \lambda_i > 0,\ i = 1, \ldots, L\}\ . \tag{5.32}$$

У результаті алгоритм класифікації можна записати так:

$$A(x) = \mathrm{sign}\left(\sum_{i=1}^{l} \lambda_i y_i \langle x_i, x \rangle - w_0 \right). \tag{5.33}$$

Звернімо увагу, що фактичне підсумовування відбувається не для всієї вибірки, а лише для опорних векторів, для яких $\lambda_i \neq 0$. Саме ця властивість розрідженості *(sparsity)* відрізняє *SVM* від інших лінійних функцій, таких як дискримінант Фішера (Рональд Фішер; Ronald Aylmer Fisher), логістична регресія та одношаровий перцептрон.

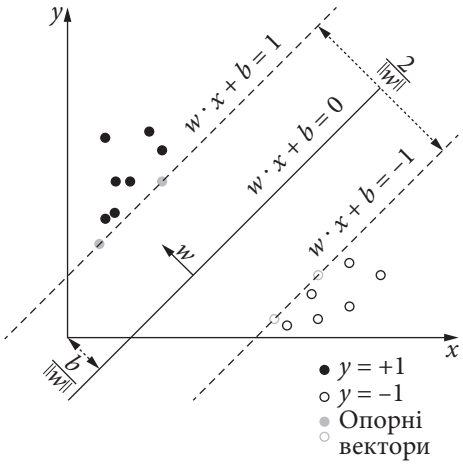


Рис. 5.15. Бінарна «жорстка» класифікація методом *SVM*

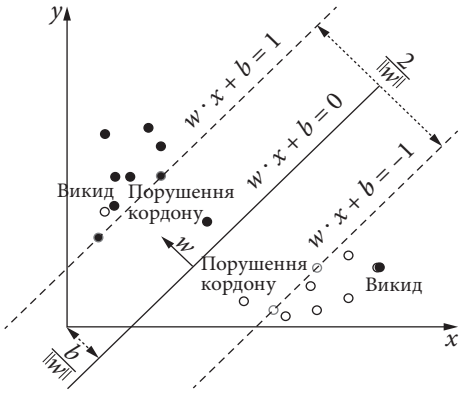


Рис. 5.16. «М'яка» класифікація методом *SVM*

SVM передбачає як «жорстку» класифікацію (рис. 5.15), коли всі об'єкти мають лежати поза розділовою гіперплощиною, так і «м'яку», коли допускається потрапляння окремих об'єктів у гіперплощину та наявність викидів.

Для «м'якої» класифікації допускаються викиди та вихід окремих точок за межу рішення (рис. 5.16).

У підсумку зазначимо, що відкритими залишаються два питання: що робити, якщо класи лінійно неподільні, та як вирішити двоїсту задачу (5.31)?

Почнемо з першого питання. У наступному розділі розглядається узагальнена двоїста задача у разі відсутності лінійної подільності. Після цього буде розглянуто перехід від скалярних добутків до довільних ядер — так званих *kernel trick*, який дає змогу будувати нелінійні подільники.

Лінійно неподільна вибірка

Щоб узагальнити *SVM* у разі лінійної неподільності, дозволимо алгоритму допускати похибки на навчальних об'єктах, але разом із тим постараємось зменшити кількість похибок. Введемо набір додаткових змінних $\xi_i > 0$, що характеризують величину похибки на об'єктах, — x_i, $i = 1, \ldots, L$. Візьмемо за точку відліку задачу (5.28); пом'якшимо обмеження-нерівності в ній, одночасно введемо в функціонал, що мінімізується, штраф за сумарну похибку:

$$\begin{cases} \dfrac{1}{2}\langle w, w\rangle + C\sum_{i=1}^{L}\xi_i \underset{w, w_0 \xi}{\min}; \\ y_i\left(\langle w, x_i\rangle - w_0\right) \geq 1 - \xi_i, \quad i = 1, \ldots, L; \\ \xi_i, \quad i = 1, \ldots, L. \end{cases} \tag{5.34}$$

До тієї ж задачі оптимізації приводить ще один ланцюжок міркувань.

Нагадаємо, що у разі $Y = \{-1, +1\}$, відступом (*margin*) об'єкта x_i від межі класів називається величина

$$m_i = y_i(\langle x_i, x\rangle - w_0).$$

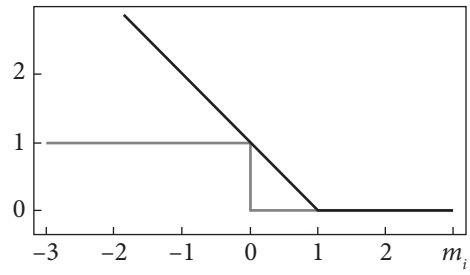


Рис. 5.17. Кусково-лінійна апроксимація порогової функції втрат

Алгоритм допускає похибку на об'єкті x_i тоді і тільки тоді, коли відступ m_i від'ємний. Якщо $m_i \in (-1, +1)$, то об'єкт x_i потрапляє всередину розділової смуги. Якщо $m_i > 1$, то об'єкт x_i класифікується правильно, і знаходиться на деякому віддаленні від смуги. Запишемо функціонал числа похибок алгоритму A на вибірці X^l у термах відступів:

$$Q(A, X^L) = \sum_{i=1}^{L} [m_i < 0].$$

Замінимо в цьому функціоналі функцію порогових втрат на кусково-лінійну. Її верхня оцінка становить $[m_i < 0] \le (1 - m_i)_+$ (як показано на рис. 5.17). Сенс цієї підстановки полягає в тому, щоб зробити функцію втрат чутливою до величини похибки і заодно ввести штраф за наближення об'єкта до межі класів.

Крім того, до функції Q додамо штрафний доданок $\tau||w||^2$. Відповідно до принципу регуляризації неправильно поставлених задач, такий доданок означає, що з усіх векторів w, що мінімізують функціонал Q, найбільшу перевагу віддають векторам з мінімальною нормою. Регуляризацію часто використовують для налаштування лінійних моделей класифікації та регресії. У разі наявності шуму та/або залежних ознак вона підвищує стійкість алгоритму відносно складу вибірки та його узагальнювальну здатність.

З урахуванням обох модифікацій функціонал якості набуває вигляду

$$Q(A, X^L) = \sum_{i=1}^{L} (1 - m_i)_+ + \tau ||w||^2 \longrightarrow \min_{w, w_0}. \qquad (5.35)$$

Отже, принцип оптимальної розділової гіперплощини (або максимізація ширини розділової смуги) збігається з принципом регуляризації за Тихоновим.

Додатна константа τ є керівним параметром методу і дає можливість знайти компроміс між максимізацією розділової смуги і мінімізацією сумарної похибки.

Повернемося до задачі (5.34) і запишемо її функцію Лагранжа:

$$L(w, w_0; \xi; \lambda, \eta) = \frac{1}{2} \langle w, w \rangle - \sum_{i=1}^{L} \lambda_i (y_i (\langle w, x_i \rangle - w_0) - 1) - \sum_{i=1}^{L} \xi_i (\lambda_i + \eta_i - C),$$

де $\eta = (\eta_1, \ldots, \eta_L)$ — вектор змінних, двоїстий до змінних $\xi = (\xi_1, \ldots, \xi_L)$. Як і минулого разу, умови Куна — Такера (Гарольд Вільям Кун; Harold William Kuhn; Альберт Вільям Такер; Albert William Tucker) зводять задачу до пошуку сідлової точки функції Лагранжа (Жозеф-Луї Лагранж; Joseph-Louis Lagrange):

$$\begin{cases} L(w, w_0; \xi; \lambda, \eta) \to \min\limits_{w, w_0 \xi} \max\limits_{\lambda, \eta}(\lambda); \\ \xi_i \geq 0, \lambda_i \geq 0, \eta_i \geq 0, i = 1, ..., L; \\ \lambda_i = 0, \text{ або } y_i (\langle x_i, x \rangle - w_0) = 1 - \xi_i, i = 1, ..., L; \\ \eta_i = 0, \text{ або } \xi_i = 0, i = 1, ..., L. \end{cases}$$

Останні два рядки містять умови доповнювальної нежорсткості. Необхідною умовою для сідлової точки є те, що похідні Лагранжа дорівнюють нулю. Звідси випливають три корисних співвідношення:

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^{l} \lambda_i y_i x_i = 0 \quad \Longrightarrow w = \sum_{i=1}^{l} \lambda_i y_i x_i; \tag{5.36}$$

$$\frac{\partial L}{\partial w_o} = - \sum_{i=1}^{l} \lambda_i y_i = 0 \quad \Longrightarrow \sum_{i=1}^{l} \lambda_i y_i = 0; \tag{5.37}$$

$$\frac{\partial L}{\partial \xi_i} = - \lambda_i - \eta_i + C = 0 \quad \Longrightarrow \eta_i + \lambda_i = C, \quad i = 1, ..., L. \tag{5.38}$$

Перші два співвідношення точно такі самі, як і в лінійно подільному випадку. З третього співвідношення та нерівності $\eta_i \geq 0$ випливає обмеження $\lambda_i \leq C$. Звідси та з умов доповнювальної нежорсткості випливає, що можливі тільки три допустимі поєднання значень змінних ξ_i, λ_i, η_i і відступів m_i.

Відповідно, всі об'єкти x_i, C поділяються на такі три типи:

1. $\lambda_i = 0; \eta_i = 0; \xi_i = 0; m_i > 1$.

Об'єкт x_i класифікується правильно і знаходиться далеко від розділової смуги. Такі об'єкти будемо називати периферійними.

2. $0 < \lambda_i < C; 0 < \eta_i < C; \xi_i = 0; m_i = 1$.

Об'єкт x_i класифікується правильно і лежить точно на межі розділової смуги. Такі об'єкти, як і раніше, будемо називати опорними.

3. $\lambda_i = C; \eta_i = 0; \xi_i > 0; m_i < 1$.

Об'єкт x_i або лежить всередині розділової смуги, але класифікується правильно ($0 < \xi_i < 1; 0 < m_i < 1$), або потрапляє на межу класів ($\xi_i = 1; m_i = 0$), або взагалі належить до чужого класу ($\xi_i > 1; m_i < 0$). У всіх цих випадках об'єкт x_i будемо називати порушником.

Відповідно до співвідношення (5.38) у лагранжіані обнуляються всі члени, що містять змінні ξ_i і η_i, і він набуває того самого вигляду, що й у разі лінійної подільності. Параметри розділової поверхні w і w_0, згідно з формулами (5.26) і (5.27), також виражаються тільки через двоїсті змінні λ_i.

Отже, задача знову зводиться до квадратичного програмування відносно двоїстих змінних λ_i. Єдина відмінність від лінійно подільного випадку полягає у появі обмеження зверху $\lambda_i \le C$:

$$\begin{cases} -L(\lambda) = -\sum_{i=1}^{L} \lambda_i + \frac{1}{2} \sum_{i=1}^{L} \sum_{j=1}^{L} \lambda_i \lambda_j y_i y_j \langle x_i, x_j \rangle \to \min\limits_{\lambda}; \\ 0 \le \lambda_i \le C, \quad i = 1, ..., L; \\ \sum_{i=1}^{L} \lambda_i y_i = 0. \end{cases} \tag{5.39}$$

На практиці для побудови *SVM* вирішують саме цю задачу, а не (5.31), оскільки гарантувати лінійну подільність вибірки в загальному випадку неможливо. Цей варіант алгоритму називають *SVM* з м'яким зазором (*soft-margin SVM*), тоді як у лінійно подільному випадку говорять про *SVM* з жорстким зазором (*hard-margin SVM*).

Для алгоритму класифікації зберігається формула (5.33), з тією лише відмінністю, що тепер ненульові λ_i мають не тільки опорні об'єкти, а й об'єкти-порушники. У певному сенсі це недолік *SVM*, оскільки порушниками часто виявляються шумові викиди, і побудоване на них вирішальне правило, по суті, спирається на шум.

Константу C зазвичай вибирають за критерієм ковзного контролю. Це трудомісткий спосіб, оскільки задачу доводиться вирішувати заново за кожного значення C.

Якщо є підстави вважати, що вибірка майже лінійно подільна, і лише об'єкти-викиди класифікуються неправильно, то можна застосувати фільтрацію викидів. Спочатку задача вирішується за деякого C, і з вибірки видаляється невелика частка об'єктів, що мають найбільше значення похибки ξ_i. Після цього задача вирішується заново за усіченою вибіркою. Можливо, доведеться зробити кілька таких ітерацій, доки решта об'єктів не виявляться лінійно подільними.

Ядра та простори

Існує ще один підхід до вирішення проблеми лінійної неподільності. Це перехід від початкового простору ознакових описів об'єктів X до нового простору H за допомогою деякого перетворення $\psi: X \to H$. Якщо простір H має достатньо високу розмірність, то можна сподіватися, що в ньому вибірка виявиться лінійно подільною (легко показати, що якщо вибірка X^L несуперечлива, то завжди знайдеться простір розмірності не більш як L, в якому вона буде лінійно подільною). Цей результат сформульовано у вигляді теореми Ковера в розділі 1. Простір H називають спрямним.

Якщо припустити, що ознаковими описами об'єктів є вектори $\psi(x_i)$, а не вектори x_i, то побудова SVM проводиться точно так само, як і раніше. Єдина відмінність полягає в тому, що скалярний добуток $\langle x, x' \rangle$ у просторі X всюди замінюється скалярним добутком $\langle \psi(x), \psi(x') \rangle$ у просторі H. Звідси

випливає природна вимога: простір H має бути наділений скалярним добутком, зокрема, придатним є будь-який евклідів, а в загальному випадку й гільбертовий, простір.

Визначення 5.2. Функція $K : X \times X \to R$ називається ядром (*kernel function*), якщо її можна подати у вигляді $K(x, x') = \langle \psi(x), \psi(x') \rangle$ за деякого відображення $\psi: X \to H$, де H — простір зі скалярним добутком.

Постановка задачі (5.39) і сам алгоритм класифікації (5.33) залежать тільки від скалярних добутків об'єктів, але не від самих ознакових описів. Це означає, що скалярний добуток $\langle x, x' \rangle$ можна формально замінити ядром $K(x, x')$. Оскільки ядро в загальному випадку нелінійне, така заміна призводить до суттєвого розширення множини реалізованих алгоритмів $a: X \to Y$.

Більше того, можна взагалі не будувати спрямляючий простір H у явному вигляді і замість підбирання відображення ψ займатися безпосередньо підбиранням ядра.

Можна піти ще далі і зовсім відмовитися від ознакових описів об'єктів. У багатьох практичних задачах об'єкти спочатку задаються інформацією про їхні парні взаємозв'язки, наприклад відношенням подібності. Якщо ця інформація допускає подання у вигляді двомісної функції $K(x, x')$, що задовольняє аксіоми скалярного добутку, то задача може вирішуватися методом *SVM*. Для такого підходу було запропоновано термін «безознакове розпізнавання» (*featureless recognition*), хоча багато давно відомих метричних алгоритмів класифікації (*kNN, RBF* та ін.) також не вимагають задання ознакових описів.

Чи може будь-яка функція двох аргументів $K(x, x')$ виконувати роль ядра? Наведена теорема дає вичерпну відповідь на це питання і показує, що клас допустимих ядер досить широкий.

Теорема 5.1 (Джеймс Мерсер; James Mercer, 1909 [5.13]). *Функція $K(x, x')$ є ядром тоді і тільки тоді, коли вона симетрична, $K(x, x') = K(x, x')$, і невід'ємно визначена, тобто:*

$\int_x \int_x K(x, x')g(x)g(x')dx\,dx' \geq 0$ для будь-якої функції g: $X \to \mathbb{R}$.

Є еквівалентне визначення невід'ємної визначеності.

Визначення 5.3. Функція $K(x, x')$ невід'ємно визначена, якщо для будь-якої скінченної вибірки $X^p = (x_1, \ldots, x_p)$ із X матриця $K = ||K(x_i, x_j)||$ розміру $p \times p$ невід'ємно визначена: $z^T Kz \geq 0$ для будь-якого $z \in \mathbb{R}^p$.

Перевірка невід'ємної визначеності функції у практичних ситуаціях може виявитися нетривіальним завданням. Часто обмежуються перебиранням скінченного числа функцій, про які відомо, що вони є ядрами. Серед них вибирається найкраща, як правило, за критерієм ковзного контролю. Очевидно, що це не оптимальне рішення. На сьогоднішній день проблема вибору ядра, оптимального для певної конкретної задачі, залишається відкритою.

Конструктивні способи побудови ядер. Правила породження, що дають можливість будувати ядра у практичних задачах.

1. Довільний скалярний добуток $K(x, x') = \langle x, x' \rangle$ є ядром.

2. Константа $K(x, x') = 1$ є ядром.

3. Добуток ядер $K(x, x') = K_1(x, x')\, K_2(x, x')$ є ядром.

4. Для будь-якої функції φ: $X \to R$ добуток $K(x, x') = K_0(\varphi(x), \varphi(x'))$ є ядром.

5. Лінійна комбінація ядер з невід'ємними коефіцієнтами $K(x, x') = \alpha_1 K_1(x, x')\, \alpha_2 K_2(x, x')$ є ядром.

6. Композиція довільної функції φ: $X \to X$ і довільного ядра K_0 є ядром: $K(x, x') = K_0(\varphi(x), \varphi(x'))$

7. Якщо s: $X \times X \to$ R — довільна симетрична інтегрована функція, то $\int_x s(x, z)s(x, z)dz$ є ядром.

8. Межа локально-рівномірно збіжної послідовності ядер є ядром.

9. Композиція довільного ядра K_0 і довільної функції f: $R \to R$, зображеної у вигляді збіжного степеневого ряду з невід'ємними коефіцієнтами $K(x, x') = fK_0(x, x')$, є ядром. Зокрема, функції $f(z) = e^z$ і $f(z) = 1/(1 - z)$ від ядра є ядрами.

Приклади ядер. Є декілька «стандартних» ядер, які за детального розгляду приводять до вже відомих алгоритмів: поліноміальних розділових поверхонь, двошарових нейронних мереж, потенціальних функцій (*RBF*-мереж) та ін. Отже, ядра претендують на роль універсальної мови для опису широкого класу алгоритмів навчання за прецедентами. Спостерігається парадоксальна ситуація. З одного боку, ядра — одне з найкрасивіших і найплідніших винаходів у машинному навчанні. З іншого боку, досі не знайдено ефективного загального підходу до їх добору в конкретних задачах.

Приклад 5.1. Нехай $X = \mathbb{R}^2$. Розглянемо ядро $K(u, v) = \langle u, v \rangle$, де $u = (u_1, u_2)$, $v = (v_1, v_2)$, і покажемо, які спрямний простір та перетворення ψ йому відповідають. Розкладемо квадрат скалярного добутку:

$$K(u, v) = \langle (u_1, u_2), (v_1, v_2) \rangle^2 = (u_1 v_1 + u_2 v_2)^2 =$$
$$= u_1^2 v_1^2 + u_2^2 v_2^2 + 2u_1 v_1 u_2 v_2 = \langle (u_1^2, u_2^2, \sqrt{2}u_1 u_2), (v_1^2, v_2^2, \sqrt{2}v_1 v_2) \rangle.$$

Ядро K представлено у вигляді скалярного добутку в просторі $H = \mathbb{R}^3$. Перетворення ψ: $\mathbb{R}^2 \to \mathbb{R}^3$ має вигляд ψ: $(u_1, u_2) \mapsto (u_1^2, u_2^2, \sqrt{2}u_1 u_2)$. Лінійній поверхні в просторі H відповідає квадратична поверхня в початковому просторі X. Зокрема, таке ядро дає можливість лінійно розділити внутрішню і зовнішню частини еліпса (задача $X \circ R$), що неможливо в початковому двовимірному просторі.

Приклад 5.2. Ускладнимо ситуацію. Нехай тепер $X = \mathbb{R}^n$, $K(u, v) = \langle u, v \rangle^d$. Тоді компонентами вектора $\psi(u)$ є різні добутки за всіх можливих цілих невід'ємних $d_1, \ldots, d_n$, що задовольняють умову $d_1 + \ldots + d_n = d$. Кількість таких добутків, а відповідно, і розмірність простору H дорівнює C_{d+n-1}^d.

Простір H ізоморфний простору всіх поліномів, що складаються з добутків степеня d від змінних $u_1, \ldots, u_n$.

Приклад 5.3. Якщо $X = \mathbb{R}^n$, $K(u, v) = (\langle u, v\rangle + 1)^d$, то H — простір усіх мономів степеня не вищого за d від змінних $u_1, \ldots, u_n$. У цьому випадку простір H ізоморфний простору всіх поліномів степеня d. Лінійна подільність множин у цьому просторі еквівалентна поліноміальній подільності множин у початковому просторі X.

Зв'язок *SVM* з двошаровими нейронними мережами. Розглянемо структуру алгоритму $A(x)$ після заміни у виразі (5.33) скалярного добутку $\langle x_i, x_j\rangle$ на ядро $K(x_i, x_j)$. Перенумеруємо об'єкти так, щоб перші h об'єктів виявилися опорними. Оскільки $\lambda_i = 0$ для всіх неопорних об'єктів, $i = h + 1, \ldots, L$, алгоритм $A(x)$ набуває вигляду

$$A(x) = \operatorname{sign}\left(\sum_{i=1}^{L} \lambda_i y_i K\langle x_i, x' - w_0\right).$$

Якщо $X = \mathbb{R}^n$, то алгоритм $A(x)$ можна розглядати як двошарову нейронну мережу, що має n вхідних нейронів і нейронів у прихованому шарі (рис. 5.18). Мережа, налаштована методом *SVM*, має кілька відмінних особливостей:

1) кількість нейронів прихованого шару визначається автоматично;

2) вектори ваг у нейронів прихованого шару збігаються з ознаковими описами опорних об'єктів;

3) прояснюється сенс двоїстих змінних: λ_i — це вага, що виражає ступінь важливості ядра $K(x_i, x)$.

Приклад 5.4. Класичну нейронну мережу з сигмоїдальними функціями активації отримаємо, якщо як ядро взяти функцію

$$K(u, v) = \operatorname{th}(k_0 + k_1 \langle u, v\rangle).$$

Ця функція задовольняє умови Мерсера не за всіх значень параметрів k_0 і k_1. Зокрема, вона їх не задовольняє при $k_0 < 0$ або $k_1 < 0$ [5.11]. Однак це не перешкоджає її успішному практичному застосуванню. Замість гіперболічного тангенса $\operatorname{th}(z)$ часто використовують також логістичну функцію $\sigma(z) = 1/(1 + e^{-z})$. Що може статися, якщо функція $K(u, v)$ не буде задовольняти умови Мерсера? Постановка задачі квадратичного програмування (5.29) залишиться тією самою і в цьому випадку. Однак квадратична форма втратить властивість невід'ємної визначеності, мінімізований функціонал уже не буде опуклим і рішення може виявитися не єдиним. Найнеприємніше те, що на межах гіперпаралелепіпеда $0 \le \lambda_i \le C$ виникне величезна кількість локальних мінімумів і пошук рішення серед них у загальному випадку потребуватиме повного перебору. В цій ситуації багато методів квадратичного програмування будуть видавати якийсь локальний мінімум, зовсім не обов'язково задовільний.

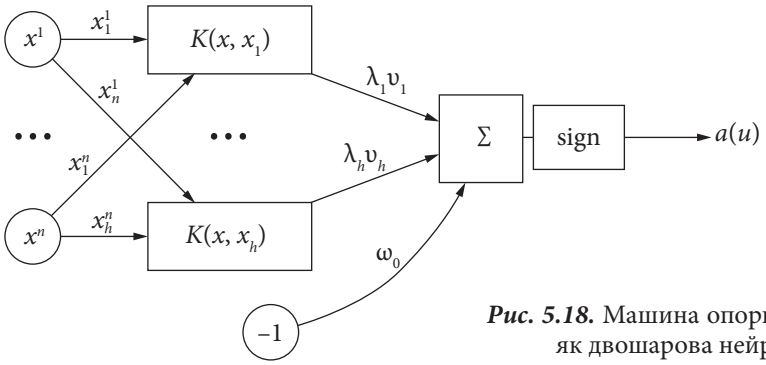


Рис. 5.18. Машина опорних векторів (*SVM*) як двошарова нейронна мережа

Приклад 5.5. Нейронну мережу з радіальними базисними функціями (*radial basis functions, RBF*) отримаємо, якщо взяти гауссове ядро:

$$K(u, v) = \exp(-\beta||u - v||^2),$$

де β — параметр. Ядро $K(x_i, x)$ обчислює оцінку близькості об'єкта x до опорного об'єкта x_i. Чим ближче об'єкти, тим більше значення ядра. Вихідний нейрон додає всі ці оцінки, множачи їх на коефіцієнти λ_i. При цьому близькості до опорних об'єктів класу +1 підсумовуються з додатними вагами, а до об'єктів класу −1 — з від'ємними. Вихідний нейрон проводить «голосування», порівнюючи сумарні близькості розпізнаваного об'єкта x до обох класів.

У розд. 1 розглянуто альтернативний метод навчання *RBF*-мереж (*Radial Basis Function Network*, мережа з радіально-базисними функціями), заснований на *EM*-алгоритмі (*Expectation-Maximization Algorithm*). Гауссові ядра відігравали роль компонентів суміші ймовірнісних розподілів. Центри ядер розміщувалися не в опорних об'єктах, а в місцях локальних скупчень щільності об'єктів. У цьому і полягає основна відмінність *SVM-RBF* від *EM-RBF*. Метод *SVM* зсуває центри гауссіанів ближче до межі класів, у результаті форма розділової поверхні описується чіткіше. Отже, *SVM-RBF* краще придатний для опису класів з межами складної форми. Натомість *EM-RBF* більш стійкий до викидів і зручніший для задач з «розмитими» межами класів.

Переваги *SVM* перед методом стохастичного градієнта:

1. Замість багатоекстремальної задачі вирішується задача квадратичного програмування, яка має єдине рішення. Методи оптимізації в цьому випадку значно ефективніші.

2. Автоматично визначається кількість нейронів прихованого шару. Вона дорівнює кількості опорних векторів.

3. Принцип оптимальної розділової гіперплощини приводить до максимізації ширини розділової смуги між класами, що забезпечує більш впев-

нену класифікацію. Градієнтні нейромережеві методи вибирають положення розділової гіперплощини випадково, «як доведеться».

Недоліки *SVM*

1. Метод опорних векторів нестійкий до шуму у вихідних даних. Якщо навчальна вибірка містить шумові викиди, вони суттєво враховуватимуться під час побудови розділової гіперплощини. Цього недоліку позбавлений метод релевантних векторів (*relevance vector machine, RVM*).

2. Досі не розроблено загальних методів побудови спрямних просторів або ядер, найбільш придатних для конкретної задачі. Побудова адекватного ядра є мистецтвом і зазвичай ґрунтується на апріорних знаннях про предметну галузь. На практиці «цілком розумні» функції $K(x, x')$, виведені зі змістовних міркувань, далеко не завжди виявляються додатно визначеними.

У загальному випадку, коли лінійна роздільність не гарантується, доводиться підбирати керівний параметр алгоритму C.

5.8. Алгоритм налаштування *SVM*

Двоїста задача (5.39) є задачею квадратичного програмування. Загальні методи розв'язування таких задач відомі, але досить трудомісткі, як у сенсі реалізації, так і за часом виконання. Тому для навчання *SVM* застосовуються алгоритми, що враховують специфічні особливості *SVM*. Специфіка полягає в тому, що кількість опорних векторів h, як правило, невелика — $h << L$, і ці вектори знаходяться поблизу межі класів. Саме ці особливості сприяють прискоренню пошуку опорних об'єктів. Спеціалізовані алгоритми налаштування *SVM* успішно впораються з вибірками з десятків тисяч об'єктів.

Розглянемо не найвідоміший, але також досить ефективний алгоритм — послідовний метод активних обмежень (*Incremental Active Set Method, INCAS*), запропонований у [5.12, 5.16].

Перепишемо двоїсту задачу (5.39) в матричних позначеннях. Введемо матрицю $Q = (y_i y_j K(x_i, x_j))_{i=1,\dots,L}^{j=1,\dots,L}$ розміру $L \times L$ і три вектори-стовпці довжиною ℓ: вектор відповідей $y = (y_i)_{i=1,\ell}$, вектор двоїстих змінних $\lambda = (\lambda_i)_{i=1,\dots,L}$ і вектор одиниць $e = (1)_{i=1,\dots,L}$. Тоді задачу (5.39) можна переписати у вигляді

$$\begin{cases} \dfrac{1}{2}\lambda^T Q\lambda = e^T\lambda \to \min\limits_{\lambda}; \\ y^T\lambda = 0; \\ 0 \le e^T\lambda \le Ce. \end{cases} \tag{5.40}$$

Припустимо, що розв'язок λ ще невідомий, але відомо розбиття множини об'єктів на три підмножини об'єктів, що не перетинаються $\{1, \dots, L\} = I_0 \cup I_C \cup I_s$:

$I_0 = \{i: \lambda_i = 0\}$ — периферійні об'єкти, $m_i > 1$;

$I_s = \{i: 0 < \lambda_i < C\}$ — опорні об'єкти, $m_i = 1$;

$I_C = \{i: \lambda_i = C\}$ — об'єкти-порушники, $m_i < 1$,

де $m_i = y_i(\langle w, x_i\rangle - w_0)$ — це відступ об'єкта x_i від межі класу.

Обмеження-нерівності $0 \le \lambda_i \le C$ стають активними, тобто оберненими на рівняння, на периферійних об'єктах ($i \in I_0$, $\lambda_i = 0$) і об'єктах-порушниках ($i \in I_C$, $\lambda_i = C$). Відповідні значення λ_i можна підставити назад у рівняння (5.40) і отримати задачу оптимізації, яка залежить тільки від частини змінних λ_i, $i \in I_s$. Щоб записати цю задачу в матричному вигляді, введемо трьохблокові позначення для матриці Q і векторів y, e, λ:

$$Q = \begin{pmatrix} Q_{SS} & Q_{SO} & Q_{SC} \\ Q_{OS} & Q_{OO} & Q_{OC} \\ Q_{CS} & Q_{CO} & Q_{CC} \end{pmatrix}; \quad y = \begin{pmatrix} y_s \\ y_0 \\ y_C \end{pmatrix}; \quad e = \begin{pmatrix} e_s \\ e_0 \\ e_C \end{pmatrix}; \quad \lambda = \begin{pmatrix} \lambda_s \\ 0 \\ Ce_C \end{pmatrix}.$$

З цими позначеннями задача (5.40) набуде вигляду

$$\begin{cases} \frac{1}{2}\lambda_S^T Q_{SS}\lambda_S + Ce_C^T Q_{CS}\lambda_S - e_S^T\lambda_S \to \min\limits_{\lambda_S}; \\ y_S^T\lambda_S + Ce_C^T y_C = 0. \end{cases} \tag{5.41}$$

Якщо не брати до уваги обмеження-нерівності, то це задача мінімізації квадратичного функціонала від $h = ||I_s||$ змінних з одним лінійним обмеженням типу рівності. Вона легко розв'язується стандартними методами лінійної алгебри і зводиться, фактично, до обернення симетричної додатно визначеної матриці Q_{SS} розміру $h \times h$.

Рішення цієї задачі дає весь вектор λ, що дає змогу обчислити параметри алгоритму w і w_0 і за формулами (5.29) та (5.32). Тепер можна класифікувати об'єкти вибірки x_i, обчислити відступи m_i і перевірити, чи правильно множину об'єктів було розбито на підмножини $I_O \cup I_C \cup I_s$. Якщо умови Куна — Такера не порушуються на жодному об'єкті, значить розв'язок задачі (5.40) знайдено. Якщо ж виявиться хоча б одна суперечність, то відповідний об'єкт має бути переведено з однієї підмножини в іншу.

Всього можливі чотири типи суперечностей:

1) якщо $i \in I_s$ і $\lambda_i \le 0$, то об'єкт x_i переводиться з I_s в I_O;

2) якщо $i \in I_s$ і $\lambda_i \ge C$, то об'єкт x_i переводиться з I_s в I_C;

3) якщо $i \in I_O$ і $m_i \le 1$, то об'єкт x_i переводиться з I_O в I_S;

4) якщо $i \in I_C$ і $m_i \ge 1$, то об'єкт x_i переводиться з I_C в I_S.

Після кожної модифікації множин I_O, I_C, I_s задача (5.40) розв'язується заново. Ітераційний процес переведення об'єктів з однієї множини в іншу триває доти, доки всі об'єкти не будуть задовольняти умови Куна — Такера.

Описаний процес є частковим випадком методу активних обмежень (*Active Sets Method*), який застосовується для розв'язування довільних задач математичного програмування з обмеженнями-нерівностями. У лінійному програмуванні він еквівалентний симплекс-методу. Збіжність цього мето-

ду в загальному випадку не гарантується, однак у задачах налаштування *SVM* зациклення трапляються вкрай рідко і їм легко запобігати за допомогою кількох вдалих евристик [5.12].

Непогана евристика полягає в тому, щоб на кожному кроці вибирати об'єкт *i*, для якого умови Куна — Такера порушуються найбільше. Це сприяє збільшенню швидкості збіжності.

Початкове наближення. Кількість ітерацій суттєво залежить від того, наскільки вдалим виявиться початкове наближення. На практиці застосовуються різні прийоми, щоб одразу точніше «вгадати» множину опорних векторів I_S.

Прийом 1. Вибирають довільну точку вибірки і знаходять найближчу до неї точку іншого класу. Для неї, у свою чергу, знаходять найближчу точку в першому класі і т. д. Цей ітераційний процес, як правило, швидко сходиться до деякої пари граничних точок. У *алгоритмі 5.1* цю пару приймають за початкове наближення I_S, але можна й продовжити процес, побудувавши кілька або навіть усі такі пари.

Прийом 2. Будують кілька досить грубих лінійних класифікаторів. Для цього можна використовувати одношарові перцептрони, проводячи невелику кількість ітерацій методом стохастичного градієнта. Потім для кожної лінійної розділової поверхні знаходять кілька найближчих до неї точок.

Алгоритм 5.1. Навчання *SVM:* метод послідовного активного обмеження *INCAS*.

Вхід: X^L — зразок навчання; C — параметр двоїстої задачі.

Вихід: параметри лінійного класифікатора w, w_0.

Процес:

1) початкове наближення:
   I_s = дві найближчі точки з різних класів;
   I_O = всі інші точки;
   $I_C \neq \emptyset$;
2) повторювати;
3) розв'язок оптимізаційної задачі щодо λ_s:

$$\begin{cases} \frac{1}{2}\lambda_S^T Q_{SS}\lambda_S + Ce_C^T Q_{CS}\lambda_S - e_S^T\lambda_S \to \min\limits_{\lambda_S}; \\ y_S^T\lambda_S + Ce_C^T y_C = 0; \end{cases}$$

4) якщо $|I_s| > 2$ і $\exists i$: $i \in I_s$ і $\lambda_i \leq 0$, то перевести i в I_O;
5) якщо $|I_s| > 2$ і $\exists i$: $i \in I_s$ і $\lambda_i \geq C$, то перевести i в I_C;
6) доки існує $i \in I_s$, якого необхідно перевести в I_O або в I_S;
7) обчислити параметри алгоритму w та w_0 за формулами (5.29) і (5.32);
8) обчислити відступи m_i, $i \in I_O \cup I_C$;
9) якщо $\exists i$: $i \in I_O$ і $m_i \leq 1$, то перевести i в I_S;
10) якщо $\exists i$: $i \in I_C$ і $m_i \geq 1$, то перевести i в I_S;
11) доки існує $i \in I_O \cup I_C$, яке необхідно перевести в I_S.

Ефективність. Висока ефективність алгоритму 5.1 випливає з двох фактів.

По-перше, оптимізаційне завдання, яке вирішується на 4-му кроці, залежить лише від матриць Q_{SS} та Q_{CS}. Це означає, що обчислювати скалярні добутки $K(x, x')$ доводиться тільки для пар об'єктів типу «опорний — опорний» та «опорний — порушник».

По-друге, множина I_S на кожному кроці змінюється лише на один елемент. Це дає змогу виконувати перерахунок зворотної матриці за $O(h^2)$ операцій, тоді як звичайне обертання потребувало б $O(h^3)$ операцій.

Переваги методу *INCAS:*

1. Метод дає можливість вирішувати завдання, в яких немає лінійної роздільності, у тому числі задачі з шумовими викидами.

2. Метод особливо ефективний, якщо кількість опорних векторів $h = |I_s|$ невелика.

3. Метод добре пристосований для розв'язання таких задач, у яких навчальні об'єкти надходять по одному в режимі реального часу. Додавання нового об'єкта реалізується практично так само, як переведення об'єкта з однієї підмножини в іншу, і потребує порядку $O(h^2)$ операцій.

Недоліки методу *INCAS.* Метод стає неефективним у разі великого числа опорних векторів. Разом із тим це означає, що або вибірка суттєво нероздільна, або ядро $K(x, x')$ було обрано невдало. І в тому, і в іншому випадку таку задачу, швидше за все, немає сенсу розв'язувати — потрібно змінювати ядро або саму постановку.

5.9. Застосування методу опорних векторів

Як приклад застосування методу опорних векторів розглянемо завдання розпізнавання образів — віднесення об'єкта до того чи іншого класу залежно від зображення предмета.

Для постановки задачі розпізнавання використовують математичний опис, тобто, на відміну від штучних нейронних мереж, де в основі лежить отримання результату за допомогою експерименту, використовують логічні міркування та математичні докази.

Найчастіше в завданнях класифікації зображень працюють з монохромними зображеннями, за такого підходу мають можливість розглядати зображення як функцію на площині. Тобто, якщо розглядати множину точок на деякій площині T, де функція $\varphi(x, y)$ відображує в кожній точці зображення його характеристики, а саме — прозорість, яскравість, щільність тощо, в такому разі ця функція і буде формальним описом цього зображення.

Задачу розпізнавання образів, зокрема рукописних поштових індексів США, за допомогою методу *SVM* було проаналізовано В. Вапником і

К. Кортес уже в одній із перших статей, присвячених застосуванню цього методу в машинному навчанні (1995) [5.20].

Проведений аналіз дав авторам підстави стверджувати: «Попри простоту дизайну поверхні прийняття рішень, новий алгоритм демонструє дуже хороші показники в порівняльному дослідженні. Інші характеристики, такі як контроль потужності та легкість зміни реалізованої поверхні прийняття рішень, роблять мережу опорних векторів надзвичайно потужною та універсальною машиною для навчання» [5.20, с. 291].

У 1998 р. вийшла стаття Філліпса (Патрік Джон Філліпс; Patrick J. Phillips), присвячена розпізнаванню облич [5.28], у якій автор проаналізував ефективність розв'язання задач верифікації та ідентифікації облич методом *SVM* і дійшов висновку про істотну перевагу застосування методу саме для верифікації облич. Відтоді застосування методу *SVM* для розпізнавання образів стало доволі поширеним.

Окремою проблемою є мультикласова класифікація, адже базова *SVM* — бінарна. Два способи мультикласової класифікації за допомогою *SVM:* один проти одного (*One-versus-One — OvO*) та один проти всіх (*One-versus-All — OvA*), на основі яких пропонуються різні алгоритми реалізації *SVM* залежно від цілей класифікації, карти ознак, характеру та кількості даних.

Поширеним є алгоритм *DAGSVM*, заснований на ациклічному графі — *Decision Directed Acyclic Graph (DDAG)*. Для *N*-класів *DAGSVM* містить $N(N-1)/2$ класифікаторів та індукується ядром простору ознак.

Для мультикласової класифікації застосовують також різні інші методи, зокрема проксимальний (*Fung Glenn*), тернарний (*Chih-Cheng Chang, Li-Jen Chien, Yuh-Jye Lee,* 2011) та ін. У 2006 р. Вапник (Володимир Вапник; Volodymyr Vapnik) з колегами запропонували концепцію *Universum SVM* для мультикласової класифікації.

Відтоді застосування *SVM* для розпізнавання образів стало доволі поширеним.

Для класифікації зображень було обрано пакет *sklearn.svm.SVC (C-Support Vector Classification)*, реалізація якого базується на *LIBSVM*. Час навчання змінюється щонайменше квадратично залежно від кількості зразків і може бути непрактичним у разі виходу за межі десятків тисяч зразків. Мультикласова підтримка виконується за схемою «один проти одного».

Для тренування було обрано датасет *Labeled Facesinthe Wild (LFW)* — загальнодоступний еталон для перевірки облич, також відомий як збіг пар (*pair matching*) [5.28].

Набір даних містить понад 13 000 зображень облич, зібраних з Інтернету. Кожне обличчя було позначено іменем зображеної людини. 1680 із зображених осіб мають дві або більше різних фотографій у наборі даних.

```
Predicting people's names on the test set
done in 0.059s
                   precision    recall  f1-score   support

     Ariel Sharon       0.69      0.69      0.69        13
     Colin Powell       0.75      0.87      0.81        60
  Donald Rumsfeld       0.79      0.70      0.75        27
    George W Bush       0.91      0.93      0.92       146
Gerhard Schroeder       0.83      0.80      0.82        25
      Hugo Chavez       0.80      0.53      0.64        15
       Tony Blair       0.94      0.83      0.88        36

         accuracy                           0.85       322
        macro avg       0.82      0.77      0.79       322
     weighted avg       0.85      0.85      0.85       322
```

```
[1] plot_gallery(X_test, prediction_titles, h, w)
```

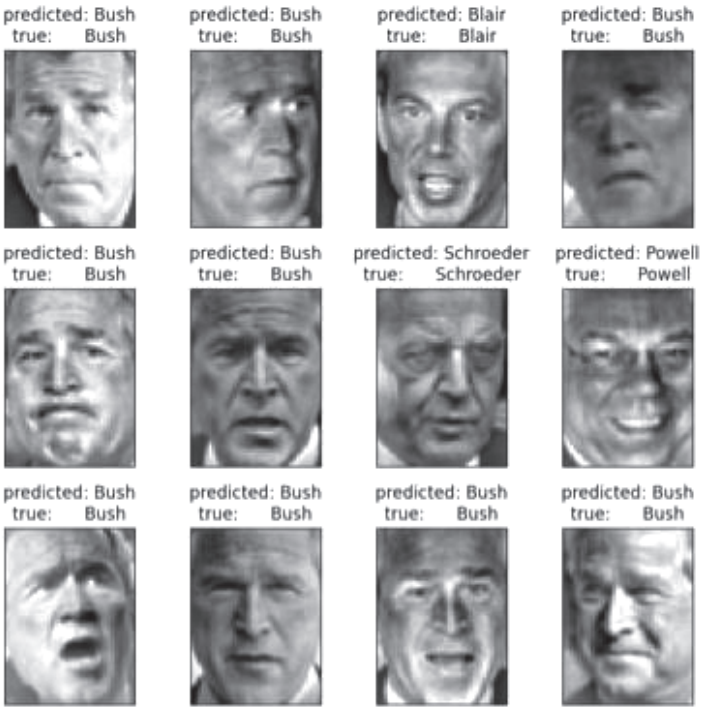


Рис. 5.19. Приклади зображень облич та результати застосування методу опорних векторів

Аналіз отриманих результатів. Навчання моделі методом *SVM* було проведено з різними параметрами, зокрема різними типами ядерної функції — базисною, лінійною, поліноміальною, сигмоїдальною. Для більшої репрезентативності результатів було завантажено екземпляри датасету, які мають понад 70 зображень на особу [5.28].

Результати для стандартного ядра радіальної базисної функції наведено на рис. 5.19.

Порівняння результатів за ключовими показниками для різних типів ядер наведено у табл. 5.20 [5.28].

Як видно з порівняння результатів реалізації *SVM*, застосування різних гіперпараметрів, зокрема різних типів ядерних функцій під час керованого навчання, впливає на характер результатів.

Найкращі результати показали ядро радіальної базисної функції та поліноміальне, тоді як лінійне та сигмоїдальне дали однакові результати, загалом також достатньо високі, але нижчі за перші два. Разом із тим якщо ядро радіальної базисної функції забезпечило високу точність класифікації для всієї вибірки, то поліноміальне ядро показало високі результати точності для найбільш характерного класу датасету, тоді як інші класи мали значно нижчі збіги, що в підсумку призвело до низького середнього значення точності класифікації.

Отже, різні типи ядерних функцій можна застосовувати для виконання різних завдань класифікації.

У процесі виконання роботи було проаналізовано ефективність методу опорних векторів для розпізнавання образів (зображень). Було розроблено і протестовано з використанням різних значень гіперпараметрів програмний продукт і проаналізовано результати експериментів, що дало підстави зробити низку висновків щодо переваг і недоліків *SVM*.

Переваги. Як дуже складний і математично обґрунтований алгоритм, *SVM* — один із найточніших алгоритмів машинного навчання. Це динамічний алгоритм, який може розв'язувати низку проблем, у тому числі лінійні та нелінійні задачі, двійкові, біноміальні та мультикласові задачі класифікації, а також задачі регресії.

SVM використовує концепцію маржі (запасу) і намагається максимізувати диференціацію між двома класами; це зменшує ймовірність перенавчання моделі, роблячи її досить стабільною. Завдяки наявності ядер і самих основ, на яких побудовано *SVM*, він може легко працювати з даними великих розмірностей і є точним для них настільки, що може конкурувати з такими алгоритмами, як *Naive Bayes*, який спеціалізується на класифікації дуже великих розмірностей.

Таблиця 5.20

Розраховані значення параметрів для різних типів ядер методу *SVM*

Kernel	C	gamma	max matching	accuracy
linear	3935,9834	0,0310547	124	0,79
poly	3474,6873	1,01133E+16	145	*0,56*
rbf	1625,2796	0,0023368	136	0,85
sigmoid	1342,1071	0,0004505	124	0,79

SVM відомий своєю швидкістю обчислень і керуванням пам'яттю. Він використовує менше пам'яті, особливо порівняно з алгоритмами машинного та глибокого навчання, з якими *SVM* часто конкурує, а іноді навіть перевершує їх.

Недоліки. Порівняно з іншими лінійними алгоритмами, як-от лінійна регресія, *SVM* погано інтерпретується, особливо коли використовуються ядра, що роблять *SVM* нелінійним. Отже, непросто оцінити, як незалежні змінні впливають на цільову змінну.

З усіма своїми перевагами та недоліками *SVM* є широко використовуваним алгоритмом. Приклади застосування *SVM* охоплюють його реалізацію в розпізнаванні зображень, таких як розпізнавання рукописного тексту та класифікація зображень. Інші сфери впровадження — виявлення аномалій, виявлення викидів, класифікація тексту, аналіз часових рядів та галузі застосування, де використовують алгоритми глибокого навчання, такі як штучні нейронні мережі.

5.10. Метод класифікації *Random Forest*

Метод випадкового лісу (*random forest*) — алгоритм машинного навчання, запропонований Брейманом (Лео Брейман; Leo Breiman) [5.30] і Катлер (Адель Катлер; Adele Cutler), що полягає у використанні ансамблю вирішальних дерев. Алгоритм поєднує в собі дві основні ідеї: метод бегінга Бреймана і метод випадкових підпросторів, запропонований Тін Кам Хо (Tin Kam Ho). Алгоритм застосовують для завдань класифікації, регресії та кластеризації. Основна ідея полягає у використанні великого ансамблю вирішальних дерев, кожне з яких саме по собі дає дуже невисоку якість класифікації, але за рахунок їх великої кількості результат виходить задовільним.

Алгоритм навчання класифікатора

Нехай навчальна вибірка складається з N зразків, розмірність простору ознак дорівнює M і задано параметр m (у задачах класифікації зазвичай $m < M$) як неповну кількість ознак для навчання.

Найпоширеніший спосіб побудови дерев ансамблю — бегінг (*bagging*, скорочення від англ. *bootstrap aggregation*) — полягає у таких діях:

1. Генерують випадкову повторну підвибірку розміром N із вихідної навчальної вибірки, застосовуючи метод бутстрепа (*bootstrap sampling*). Це означає, що кожен зразок вибірки має однакову ймовірність бути обраним у підвибірку. Деякі зразки можуть потрапити до підвибірки кілька разів, тоді як інші залишаються невідібраними (*out-of-bag*). Очікувана частка невідібраних зразків становить приблизно $1/e$ (де e — основа натурального логарифма, приблизно 0,368 або 36,8% усіх зразків).

2. Будують вирішальне дерево, що класифікує зразки підвибірки, причому в ході створення чергового вузла дерева вибирають набір ознак, на основі яких виконують розбиття (не з усіх ознак *M*, а лише з *m* випадково обраних). Вибір найкращого з цих *m* ознак може здійснюватися у різний спосіб. В оригінальному методі Бреймана використовують критерій Джині, який застосовують також в алгоритмі побудови вирішальних дерев *CART*. У деяких реалізаціях алгоритму натомість використовується критерій приросту інформації [5.33].

3. Дерево будують до повного вичерпання підвибірки і не виконують процедуру відсікання гілок (*pruning*), на відміну від вирішальних дерев алгоритмів типу *CART* або C4.5 (*Data Mining*).

Класифікацію об'єктів здійснюють голосуванням: кожне дерево комітету відносить об'єкт, що класифікується, до одного з класів, а перемагає клас, за який проголосувала найбільша кількість дерев.

Оптимальну кількість дерев підбирають так, щоб мінімізувати похибку класифікатора на тестовій вибірці. У разі її відсутності мінімізується оцінка похибки на зразках, що не увійшли до набору.

Оцінка важливості змінних

Випадкові ліси, отримувані описаними вище методами, можна використати для оцінки важливості змінних у задачах регресії і класифікації. Спосіб такої оцінки було описано Брейманом (Лео Брейман — Leo Breiman).

Перший крок в оцінці важливості змінної у тренувальному наборі — тренування випадкового лісу на цьому наборі. У процесі побудови моделі для кожного елемента тренувального набору записують так звану помилку невідібраних елементів (*out-of-bag error*). Потім така похибка усереднюється по всьому випадковому лісу.

Для того щоб оцінити важливість параметра після тренування, значення параметра перемішуються для всіх записів тренувального набору і помилку невідібраних елементів обчислюють знову. Важливість параметра оцінюють за допомогою усереднення для всіх дерев різниці показників похибок невідібраних елементів до і після перемішування значень. У такому разі значення цих помилок нормалізуються як стандартне відхилення.

Параметри вибірки, які дають більші значення, вважаються важливішими для тренувального набору. Метод має потенційний недолік — для категоріальних змінних з великою кількістю значень метод схильний вважати такі змінні важливішими. Часткове перемішування значень у такому разі може знижувати вплив цього ефекту [5.32, 5.33]. З груп кореляційних параметрів, важливість яких виявляється однаковою, вибирають менші за чисельністю групи [5.34].

Переваги методу:

1. Здатність ефективно обробляти дані з великою кількістю ознак та класів.

2. Нечутливість до масштабування (і взагалі до будь-яких монотонних перетворень) значень ознак.

3. Однаково добре обробляються як неперервні, так і дискретні ознаки. Є методи побудови дерев за даними з пропущеними значеннями ознак.

4. Є методи оцінювання значущості окремих ознак моделі.

5. Внутрішня оцінка здатності моделі до узагальнення (тест за невідібраними зразками).

6. Висока паралелізованість і масштабованість.

Недоліки методу:

1. Великий розмір моделей.

2. Потрібні значні обсяги пам'яті для зберігання моделі.

3. Алгоритм використовується в наукових працях, наприклад, для оцінки якості статей Вікіпедії [5.34, 5.35].

Висновки

У розділі описано нейронно-нечітку мережу для класифікації *NEFClass*, її переваги та недоліки. Запропоновано модифіковану систему *NEFClass-M*, яка відрізняється рандомізацією вихідної вибірки, оптимізацією параметра «швидкість навчання» та використанням гауссових функцій належності (ФН) замість трикутних. Це дало змогу реалізувати градієнтний алгоритм навчання та його модифікації.

Порівняльні експерименти з використанням *NEFClass* і *NEFClass-M* показали переваги модифікованої системи за показником якості класифікації (відсоток правильно класифікованих зразків).

У завданні класифікації об'єктів електрооптичних зображень результати досліджень показали:

- якість навчання системи *NEFClass* значно залежить від параметра швидкості навчання. Збільшення швидкості навчання вагових коефіцієнтів сприяє швидшій збіжності алгоритму та зменшенню похибки, але надто велике значення швидкості призводить до явища «осциляції»;
- найефективнішим алгоритмом генерації правил виявився підхід «кращі для класу». Проте він не забезпечує суттєвого зменшення кількості неправильно класифікованих зразків. Встановлення зважених коефіцієнтів правил має слабкий вплив на результати;
- експерименти на реальних даних підтвердили адекватність застосування нейронно-нечіткого методу в завданнях класифікації об'єктів на електрооптичних зображеннях за умов високого рівня випадкових перешкод (наприклад, піна в прибережній зоні океану).

Розглянуто метод опорних векторів (*SVM*) як один із найпопулярніших методів класифікації. Викладено основи теорії методу, поняття опорних векторів,

математичну модель та алгоритм побудови лінійної межі, якщо вона є. Описано підхід для випадків, коли лінійної розділової межі немає, з використанням двоїстої задачі квадратичного програмування та алгоритму її розв'язання.

Для випадків нелінійної розділової межі запропоновано використання ядерних функцій. Наведено приклади ядерних функцій та сформульовано умови їх застосування.

Описано зв'язок ядерних функцій із нейронними мережами, які використовують сигмоїдальні та радіально-базисні функції. Розглянуто метод активного обмеження для побудови нелінійної розділяючої межі за допомогою методу опорних векторів.

Детально розглянуто застосування методу опорних векторів у завданні розпізнавання образів, зокрема облич відомих політичних діячів. Проведено аналіз якості розпізнавання з використанням різних ядерних функцій і визначено тип ядра, яке виявилося найефективнішим для цього завдання.

Описано метод класифікації *Random Forest*, його алгоритм і властивості. Наведено рекомендації щодо вибору найбільш значущих змінних (ознак) для побудови дерев рішень та класифікації.

СПИСОК ЛІТЕРАТУРИ

5.1. Engelbrecht A. P. Computational intelligence: An introduction (2nd ed.). John Wiley & Sons., 2007. 640 p. URL: https://archive.org/details/computationalint0002enge

5.2. Zgurovsky M., Zaychenko, Y. Fundamentals of computational intelligence: System approach. Springer, 2016. 395 p. URL: https://link.springer.com/book/10.1007/978-3-319-35162-9

5.3. Зайченко Ю. П. Основи проектування інтелектуальних систем. Київ: Видавничий дім «Слово», 2004. 352 с.

5.4. Nauck D., Kruse R. New learning strategies for NEFCLASS. *Proceedings of the Seventh International Fuzzy Systems Association World Congress (IFSA'97).* 1997. IV. P. 50—55.

5.5. Nauck D., Kruse R. What are neuro-fuzzy classifiers? *Proceedings of the Seventh International Fuzzy Systems Association World Congress (IFSA'97).* 1997. IV. P. 228—233.

5.6. Nauck D. Building neural fuzzy controllers with NEFCON-I. *Fuzzy Systems in Computer Science and Artificial Intelligence.* 1994. P. 141—151.

5.7. Nauck D., Klawonn F., Kruse R. Foundations of neuro-fuzzy systems. John Wiley & Sons. 1997. 305 p.

5.8. Зайченко Ю. П. Основи проектування інтелектуальних систем. Київ: Видавничий дім «Слово». 2004. 352 с.

5.9. Зайченко Ю. П. Нечіткі моделі та методи в інтелектуальних системах. Київ: Видавничий дім «Слово». 2008. 354 с.

5.10. Bartlett P., Shawe-Taylor J. Generalization performance of support vector machines and other pattern classifiers. Advances in Kernel Methods. MIT Press, 1998. URL: http://citeseer.ist.psu.edu/bartlett98generalization.html

5.11. Burges C. J. C. Geometry and invariance in kernel-based methods. In: B. Schölkopf, C. J. C. Burges, & A. J. Smola (Eds.). Advances in kernel methods: Support vector learning. MIT Press, 1999. P. 89—116. https://dl.acm.org/doi/10.5555/299094.299100

5.12. Fine S., Scheinberg, K. INCAS: An incremental active set method for SVM (Tech. Rep.). Haifa, 2002. URL: http://citeseer.ist.psu.edu/fine02incas.html

5.13. Mercer J. Functions of positive and negative type and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society of London. Series A*. 1909. 209. P. 415—446.

5.14. Osuna E., Freund R., Girosi F. An improved training algorithm for support vector machines. *Neural Networks for Signal Processing VII: Proceedings of the 1997 IEEE Workshop*. 1997. P. 276—285. URL: http://citeseer.ist.psu.edu/osuna97improved.html

5.15. Platt J. C. Fast training of support vector machines using sequential minimal optimization. In: B. Schölkopf, C. J. C. Burges, A. J. Smola (Eds.). Advances in kernel methods: Support vector learning. MIT Press, 1999. P. 185—208.

5.16. Scheinberg K. An efficient implementation of an active set method for SVMs. *Journal of Machine Learning Research*. 2006. 7. P. 2237—2257.

5.17. Shawe-Taylor J., Cristianini N. Robust bounds on generalization from the margin distribution (Tech. Rep. NC2-TR-1998-029). Royal Holloway, University of London, 1998. URL: http://citeseer.ist.psu.edu/shawe-taylor98robust.html

5.18. Smola A., Schölkopf B. A tutorial on support vector regression (Tech. Rep. NeuroCOLT2 NC2-TR-1998-030), 1998. URL: http://citeseer.ist.psu.edu/smola98tutorial.html

5.19. Vapnik V., Chapelle O. Bounds on error expectation for support vector machines. *Neural Computation*. 2000. 12(9). P. 2013—2036. URL: http://citeseer.ist.psu.edu/vapnik99bounds.html

5.20. Cortes C., Vapnik, V. Support-vector networks. *Machine Learning*. 1995. 20. P. 273—297. URL: https://link.springer.com/article/10.1007/BF00994018

5.21. Bennett K. P., Demiriz A. Semi-supervised support vector machines. *Advances in Neural Information Processing Systems*. 1999. 11. P. 368—374. URL: https://papers.nips.cc/paper/1998/file/5568f6dfc5a6ecf7b0fbd3c780e5af3b-Paper.pdf

5.22. Vapnik V. The nature of statistical learning theory. Springer, 1995. URL: https://link.springer.com/book/10.1007/978-1-4757-2440-0

5.23. Joachims T. Transductive inference for text classification using support vector machines. *Proceedings of the Sixteenth International Conference on Machine Learning (ICML'99)*. Morgan Kaufmann Publishers Inc, 1999. P. 200—209.

5.24. Hsin-Yuan H., Chih-Jen L. Linear and kernel classification: When to use which? *SIAM*. 2023. P. 216—224.

5.25. Platt J. C., Cristianini N., Shawe-Taylor J. Large margin DAGs for multiclass classification. *Advances in Neural Information Processing Systems*. 2000. P. 547—553.

5.26. Weston J., Collobert R., Sinz F., Bottou L., Vapnik V. Inference with the Universum. *Proceedings of the 23rd International Conference on Machine Learning (ICML)*. 2006. P. 1009—1016.

5.27. Chapelle O., Schölkopf B., Zien, A. Semi-supervised learning. MIT Press, 2006. 508 p.

5.28 Phillips P. J. Support vector machines applied to face recognition. *Advances in Neural Information Processing Systems*. 1999. 11. P. 803—809.

5.29 Chang C.-C., Lin C.-J. LIBSVM: A library for support vector machines. Last updated: August, 2001. 23. P. 2022. URL: https://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf

5.30 Breiman L. Random forests. *Machine Learning*. 2001. 45(1). P. 5—32. URL: https://link.springer.com/article/10.1023/A:1010933404324?utm_source=chatgpt.com

5.31 Hastie T., Tibshirani R., Friedman J. The elements of statistical learning: Data mining, inference, and prediction (2nd ed.). Springer, 2009. 745 p. URL: https://link.springer.com/book/10.1007/978-0-387-84858-7

5.32 Deng H., Runger G., Tuv E. Bias of importance measures for multi-valued attributes and solutions. *Proceedings of the 21st International Conference on Artificial Neural Networks (ICANN)*. 2011. P. 293—300. URL: https://link.springer.com/chapter/10.1007/978-3-642-21738-8_38

5.33 Altmann A., Tolosi L., Sander O., Lengauer T. Permutation importance: A corrected feature importance measure. Bioinformatics, 2010. 26(10). P. 1340—1347. URL: https://academic.oup.com/bioinformatics/article/26/10/1340/193348

5.34 Tolosi L., Lengauer T. Classification with correlated features: Unreliability of feature ranking and solutions. *Bioinformatics.* 2011. 27(14). P. 1986—1994. URL: https://academic.oup.com/bioinformatics/article/27/14/1986/194387

5.35 Węcel K., Lewoniewski W. Modelling the quality of attributes in Wikipedia infoboxes. *Lecture Notes in Business Information Processing.* 2015. 228. P. 308—320. URL: https://link.springer.com/chapter/10.1007/978-3-319-26762-3_27

РОЗДІЛ 6

КЛАСТЕРНИЙ АНАЛІЗ В ІНТЕЛЕКТУАЛЬНИХ СИСТЕМАХ

6.1. Кластер-аналіз. Постановка задачі.
Критерії якості та метрики кластер-аналізу

6.2. Класифікація алгоритмів кластерного аналізу

6.3. Нечіткий метод *k*-середніх

6.4. Визначення початкового розміщення центрів кластерів

6.5. Алгоритм нечіткого кластер-аналізу Густавсона — Кесселя

6.6. Застосування нечітких методів *k*-середніх і Густавсона — Кесселя
в задачах автоматичної класифікації

6.7. Адаптивні робастні алгоритми кластеризації

Висновки

Список літератури

Термін «кластерний аналіз» уперше ввів Роберт Трайон (Robert C. Tryon) у 1939 р. як набір різних алгоритмів класифікації без учителя [6.1]. Загальне питання дослідників у багатьох сферах полягає в тому, як організувати дані в наочні структури, тобто розгорнути таксономію. Наприклад, біологи ставлять за мету розподілити тварин за різними видами, щоб змістовно описати відмінності між ними. Відповідно до сучасної системи, прийнятої в біології, людина належить до приматів, ссавців, амніотів, хребетних, хордових і тварин. Зауважимо, що у цій класифікації чим вищий рівень агрегації, тим менше подібності між членами у відповідному класі. Людина має більше подібності до інших приматів (тобто мавп), ніж до «віддалених» членів класу ссавців (наприклад, собак) тощо.

Сфери застосування. Техніку кластеризації застосовують у найрізноманітніших галузях. Джон Хартіган (John A. Hartigan, 1975) навів чудовий огляд багатьох опублікованих досліджень з результатами, отриманими методами кластерного аналізу. Наприклад, у галузі медицини кластеризація захворювань, лікування захворювань або симптомів захворювань призводить до таксономій, що широко використовуються. У психіатрії правильна діагностика кластерів симптомів, як-от параноя, шизофренія тощо, є вирішальною для успішної терапії. В археології за допомогою кластерного аналізу дослідники намагаються встановити таксономії кам'яних знарядь, поховання об'єктів тощо. Відомі широкі застосування кластерного аналізу в маркетингових дослідженнях. Щоразу, коли необхідно класифікувати «гори» інформації до придатних для подальшого оброблення груп, кластерний аналіз виявляється дуже корисним і ефективним. В останні роки кластерний аналіз широко використовують в інтелектуальному аналізі даних та *Data Mining* як один із найважливіших методів [6.2].

Мета цього розділу — розгляд методів кластерного аналізу, як чітких (методу *k*-середніх, методу Уорда (Joe H. Ward Jr.), найближчого сусіда, найдальшого сусіда), так і нечітких (нечіткого методу *k*-середніх, методу Густавсона — Кесселя, а також методів визначення початкового положення центрів кластерів — пікового та різницевого групування. Крім того, розглянуто перспективні нечіткі методи можливісної кластеризації, які мають підвищену стійкість до різного роду шумів та викидів, що робить їх робастними. Описано різні варіанти реалізації цих методів, зокрема рекурентні алгоритми та методи кластерного аналізу потокових даних, проаналізовано їхні властивості і зроблено їх порівняльний аналіз із класичними нечіткими методами кластеризації.

6.1. Кластер-аналіз. Постановка задачі. Критерії якості та метрики кластер-аналізу

Розглянемо постановку задачі кластерного аналізу. Нехай задано множину спостережень

$$X = \{X_1, X_2, \ldots, X_n\},$$

де $X_i = \{x_{ij}\}$, $j = \overline{1, N}$.

Потрібно розбити вибірку X на непересічні підмножини — кластери $S_1, \ldots, S_k$, щоб забезпечити мінімум (екстремум) деякого критерію (функціоналу якості), тобто знайти такі $S = (S_1, S_2, \ldots, S_k)$, щоб $F(S) \to \min(\max)$.

Можливі різні види критеріїв (функціоналів) розбиття. Зауважимо, що це завдання тісно пов'язане з визначенням деякої метрики у просторі ознак.

Розглянемо найбільш використовувані **функціонали якості розбиття** [6.3].

1. Коефіцієнт розбиття F, який визначається як

$$F = \sum_{j=1}^{K} \sum_{i=1}^{n} \frac{w_{ij}^2}{n}, \tag{6.1}$$

де $w_{ij} \in [0; 1]$ — деякий ступінь належності i-го об'єкта j-му кластеру. Діапазон зміни функціонала становить

$$F \in \left[\frac{1}{k}; 1\right],$$

де n — число об'єктів; k — число кластерів.

2. Індекс чіткості (*NFI*):

$$NFI = \frac{KF - 1}{K - 1}, \; NFI \in [0; 1], \tag{6.2}$$

де K — число класів (кластерів); F — коефіцієнт розбиття.

3. Ентропія розбиття:

$$H = -\sum_{j=1}^{K} \sum_{i=1}^{n} \frac{w_{ij}^2 \ln(w_{ij})}{n}, \; H \in (0; \ln K). \tag{6.3}$$

4. Нормалізована ентропія розбиття:

$$H_1 = \frac{H}{1 - K/n}, \; H_1 \in \left(0; \frac{n^* \ln K}{n - K}\right), \tag{6.4}$$

де n — число об'єктів (точок).

5. Модифікована ентропія:

$$H_2 = \frac{H}{\ln K}, \; H_2 \in (0; 1). \tag{6.5}$$

6. Другий функціонал Рубенса:

$$F2 = \frac{1}{2}\left(\frac{1}{n}\sum_{i=1}^{n} \max_j W_{ij} + \min_i \max_j W_{ij}\right), \; F2 \in \left(\frac{1}{K}, 1\right). \tag{6.6}$$

7. Третій функціонал Рубенса (Heinrich Rubens) (другий індекс чіткості):

$$NF2I = \frac{K^*F2 - 1}{K - 1},\ NF2I \in (0; 1). \tag{6.7}$$

Оскільки вихідна інформація задається як матриця X, то виникає *проблема вибору метрики*. Вибір метрики — найважливіший чинник, що впливає на результати кластер-аналізу. Залежно від типу ознак застосовують різні критерії оцінки близькості (метрики).

Нехай є зразки X_i та X_k у n-вимірному просторі ознак. Розглянемо ***основні метрики кластеризації***.

1. Індекс Калінскі — Харабаша (Calinski — Harabasz), що визначається як

$$CH(C) = \frac{N - K}{K - 1} \cdot \frac{\sum_{k=1}^{K} |c_k| \, ||\overline{c_k} - X||}{\sum_{k=1}^{K} \sum_{i=1}^{c_k} x_i ||x_i - c_k||},$$

де N — кількість вхідних векторів, що використовуються для навчання мережі; K — кількість виділених кластерів; $|c_k|$ — кількість елементів у k-му кластері; x_i — i-й елемент k-го кластера; c_k — центроїда k-го кластера; X — «глобальна» центроїда вхідного набору даних.

Отже, цей індекс можна трактувати як відношення зваженої міжкластерної дисперсії до внутрішньокластерної дисперсії. Тобто компактність виділених кластерів розраховується на основі відстані від точок кластера до їх центрів, а роздільність — на основі відстані від центрів кластерів до «глобальної» центроїди. Зазначимо, що більші значення цієї метрики свідчать про краще значення кластеризації.

2. Індекс Девіса — Болдуїна (David L. Davies (Девід Л. Девіс), Donald W. Bouldin (Дональд В. Болдуїн)), що визначається як

$$DB(C) = \frac{1}{K} \sum_{c_k \in C} \max_{c_l \in C / c_k} \{ \frac{S(c_k) + S(c_l)}{||\overline{c_k} - c_l||} \},$$

де $S(c_k) = \frac{1}{|c_k|} \sum_{x_i \in c_k} ||x_i - \overline{c_k}||$.

Цей індекс обчислює компактність як відстань від об'єктів кластера до їх центрів (центроїд), а роздільність — як відстань між центроїдами. Зазначимо, що для цієї метрики менші значення відповідають кращій кластеризації. Індекс Девіса — Болдуїна дуже близький за змістом до критерію Хі — Бені (X.L. Xie (Ши Ляо Се, Xie Liao), G. Beni (Гуїдо Бені, Guido Beni)).

3. «Силует» кластера, що визначається за виразом

$$\mathrm{Sil}(c) = \frac{1}{N} \sum_{c_k \in C} \sum_{x_i \in c_k} \frac{b(x_i, c_k) - a(x_i, c_k)}{\{\max a(x_i, c_k), b(x_i, c_k)\}},$$

Таблиця 6.1

Метрики, що використовуються під час кластеризації

Найменування метрики	Тип ознак	Формула для оцінки міри близькості (метрики)
Евклідова відстань	Кількісні	$d_{ik} = (\Sigma^{N}_{j=1}(x_{ij} - x_{kj})^2)^{1/2}$
Міра подібності Хеммінга	Номінальні (якісні)	$\mu^{N}_{ij} = n_{ik} / N$, де n_{ik} — число спільних ознак у зразків X_i і X_k
Міра подібності Роджерса — Танімото	Номінальні шкали	$\mu^{R\text{-}T}_{ij} = n''_{ik} (n'_i + n'_k - n''_{ik})$, де n''_{ik} — число спільних одиничних ознак у зразків X_i і X_k; n'_i, n'_k — загальна кількість одиничних ознак у зразків X_i та X_k відповідно
Манхеттенська метрика	Кількісні	$d^{(1)}_{ik} = \Sigma^{N}_{j=1} \lvert x_{ij} - x_{kj} \rvert$
Відстань Мінковського	Кількісні	$d_{ik} = \Sigma^{N}_{j=1} (x_{ij} - x_{kj})^p)^{1/p}$, $p > 1$, p — ціле число
Відстань Махаланобіса	Кількісні	$d^{M}_{ik} = (x_{ij} - x_{kj})^T W^{-1} (x_{ij} - x_{kj})$, де W — коваріаційна матриця вибірки $X = \{X_1, X_2, \ldots, X_n\}$
Відстань Журавльова	Змішані	$d_{ik} = \Sigma^{N}_{j=1} I^{j}_{ik}$, де I^{j}_{ik} {1, якщо $\lvert x_{ij} - x_{kj} \rvert < \varepsilon$; 0, інакше

де $a(x_i, c_k)$ — середня відстань від $x_i \in c_k$ до інших об'єктів із кластера c_k, $b(x_i, c_k)$ — середня відстань від $x_i \in c_k$ до інших об'єктів із кластера c_l.

Розрахунки середніх відстаней обчислюються так:

$$a(x_i, c_k) = \frac{1}{|c_k|} \Sigma_{x_j \in c_k} ||x_i - x_j||;$$

$$b(x_i, c_k) = \frac{1}{|c_k|} \{\frac{1}{|c_l|} \Sigma_{x_j \in c_l} ||x_i - x_j||\}.$$

Значення метрики «силует» перебувають у діапазоні від –1 до 1. Чим ближче розраховане значення до 1, тим краще побудована кластеризація.

Інші метрики, що використовуються під час кластеризації, наведено в табл. 6.1.

Є велика кількість алгоритмів кластеризації, які використовують різні метрики та критерії розбиття. При цьому число класів (кластерів) визначається або апріорі, або в процесі роботи самого алгоритму.

6.2. Класифікація алгоритмів кластерного аналізу

Важливо, щоб у результаті виконання кластеризації було побудовано *K* кластерів. Припускається, що кластеризація має виявити природні локальні згущення об'єктів. Тому число кластерів є параметром, який часто істотно ускладнює вигляд алгоритму, якщо воно вважається невідомим, і значно впливає на якість результату, якщо воно відоме.

Проблема вибору числа кластерів дуже нетривіальна. Досить зазначити, що для отримання задовільного теоретичного рішення часто потрібно зробити досить суттєві припущення про властивості деякого заздалегідь заданого сімейства розподілів. Однак про які припущення може йтися, коли, особливо на початку дослідження, про дані практично нічого не відомо? Тому алгоритми кластеризації зазвичай будуються як певний спосіб перебирання числа кластерів і визначення його оптимального значення під час цього процесу.

Число методів розбиття множини на кластери досить велике. Усі їх можна розподілити на ієрархічні та неієрархічні. У неієрархічних алгоритмах характер їх роботи та умова зупинки потрібно заздалегідь регламентувати нерідко досить великою кількістю параметрів, що іноді важко, особливо на початковому етапі вивчення матеріалу. Проте в таких алгоритмах досягається велика гнучкість у варіюванні кластеризації і зазвичай визначається число кластерів. З іншого боку, коли об'єкти характеризуються великою кількістю ознак (параметрів), то важливого значення набуває завдання групування ознак. Вихідна інформація міститься у квадратній матриці зв'язків ознак, зокрема кореляційній матриці. Основою успішного вирішення завдання групування є неформальна гіпотеза про невелику кількість прихованих факторів, що визначають структуру взаємних зв'язків між ознаками. В ієрархічних алгоритмах фактично відмовляються від визначення числа кластерів, будуючи повне дерево вкладених кластерів (дендрограми). Число кластерів визначається з припущень, що в принципі не належать до роботи алгоритмів, наприклад, за динамікою зміни порога розбиття (злиття) кластерів. Проблеми таких алгоритмів добре вивчено: вибір мір близькості кластерів, проблема інверсій індексації в дендрограмах, негнучкість ієрархічних класифікацій, яка іноді дуже небажана. І все ж побудова кластеризації у вигляді дендрограми дає можливість отримати найповніше уявлення про структуру кластерів.

Ієрархічні алгоритми пов'язані з побудовою дендрограм і поділяються на:

а) *агломеративні*, що характеризуються послідовним об'єднанням вихідних елементів та відповідним зменшенням числа кластерів (побудова кластерів знизу вгору);

б) *дивізимні* (подільні), у яких число кластерів зростає, починаючи з одного, у результаті утворюється послідовність розщеплювальних груп (побудова кластерів згори донизу).

Ієрархічні алгоритми

Агломеративні алгоритми. На першому кроці всю множину I зображують як множину кластерів:

$$c_1 = \{i_1\}, c_2 = \{i_2\}, \ldots, c_m = \{i_m\}.$$

На наступному кроці вибирають два найближчих один до одного кластери (наприклад, c_p і c_q) і об'єднують в один загальний кластер. Нова множина, що складається вже з $(m - 1)$ кластерів, має вигляд

$$c_1 = \{i_1\}, c_2 = \{i_2\}, c_p = \{i_p, i_q\}, \ldots, c_m = \{i_m\}.$$

Повторюючи процес, отримуємо послідовні множини кластерів, що складаються з $(m - 2)$, $(m - 3)$, $(m - 4)$ тощо кластерів.

Наприкінці процедури отримуємо єдиний кластер, що складається з m об'єктів і збігається з початковою множиною I.

Визначати відстані між кластерами можна різними методами, залежно від яких отримують алгоритми з різними властивостями.

Є кілька методів перерахунку відстаней з використанням попередніх значень відстаней для кластерів, що об'єднуються, які відрізняються коефіцієнтами у формулі [6.1]:

$$d_{rs} = \alpha_p d_{ps} + \alpha_q d_{qs} + \beta d_{pq} + \gamma |d_{ps} - d_{qs}|.$$

Якщо кластери p і q об'єднуються в кластер r й потрібно розрахувати відстань від нового кластера до кластера s, застосування того чи іншого методу залежить від способу визначення відстані між кластерами. Ці методи різняться значеннями коефіцієнтів α_p, α_q, β та γ. Коефіцієнти перерахунку відстаней між кластерами α_p, α_q, β та γ, де k_p і k_q — число об'єктів у кластерах p і q, наведено в табл. 6.2.

Дивізимні алгоритми. Дивізимні кластерні алгоритми, на відміну від агломеративних, на першому кроці подають усі множини елементів I як єдиний кластер. На кожному кроці алгоритму один з наявних кластерів рекурсивно поділяється на два дочірні. Отже, ітераційно утворюються кластери згори вниз. Цей підхід не так докладно описано в літературі кластерного аналізу, як агломеративні алгоритми. Його застосовують, коли необхідно розподілити всю множину об'єктів I на відносно невелику кількість кластерів.

Один із перших дивізимних алгоритмів був запропонований Р. Макнотоном (Роберт МакНаутон; Robert McNaughton) у 1965 р. [6.1].

На першому кроці всі елементи поміщаються в один кластер $C_1 = I$. Потім вибирають елемент, у якого середня відстань від інших елементів у цьому кластері найбільша. Середнє значення можна обчислити, напри-

клад, за формулою

$$D_{C_1} = \frac{1}{\frac{(m(m-1)}{2}} \times \sum_{\substack{q=1 \\ q \neq p}}^{m} \sum_{p=1}^{m} d(i_p, i_q), \quad \forall i_p, \quad i_q \in C.$$

Вибраний елемент видаляється із кластера C_1 і формує перший член другого кластера C_2.

На кожному наступному кроці елемент кластера C_1, для якого різниця між середньою відстанню до елементів, що знаходяться в C_2, і середньою відстанню до елементів, що залишаються в C_1, найбільша, переноситься в C_2. Перенесення елементів C_1 в C_2 триває доти, доки відповідні різниці середніх не стануть негативними, тобто є елементи, розміщені до елементів кластера C_2 ближче ніж до елементів кластера C_1.

У результаті один кластер ділиться на два дочірні, один з яких розщеплюється на наступному рівні ієрархії. Кожний наступний рівень застосовує процедуру розподілу до одного з кластерів, отриманих на попередньому рівні. Вибір кластера, що розщеплюється, можна виконувати по-різному.

Таблиця 6.2

Перерахунок відстаней між кластерами

Назва методу	α_p	α_q	β	γ
Відстань між найближчими сусідами — найближчими об'єктами кластерів (*Nearest neighbor*)	1/2	1/2	0	−1/2
Відстань між найдальшими сусідами (*Furthest neighbor*)	1/2	1/2	0	1/2
Метод медіан (*Median clustering*) — той самий центроїдний метод, але центр об'єднаного кластера обчислюється як середнє всіх об'єктів	1/2	1/2	−1/4	0
Середня відстань між кластерами (*Between—groups linkage*)	1/2	1/2	0	0
Середня відстань між усіма об'єктами пари кластерів з урахуванням відстаней усередині кластерів	$\frac{k_p}{k_p + k_q}$	$\frac{k_q}{k_p + k_q}$	0	0
Відстань між центрами кластерів (*Centroid clustering*), або центроїдний метод. Недоліком цього методу є те, що центр об'єднаного кластеру обчислюється як середнє центрів кластерів, що об'єднуються, без урахування їх обсягу	$\frac{k_p}{k_p + k_q}$	$\frac{k_q}{k_p + k_q}$	$\frac{-k_p k_q}{k_p + k_q}$	0
Метод Уорда (*Ward's method*). Як відстань між кластерами береться приріст суми квадратів відстаней об'єктів до центрів кластерів, що отримується внаслідок їх об'єднання	$\frac{k_r + k_p}{k_r + k_q + k_p}$	$\frac{k_r + k_q}{k_r + k_q + k_p}$	$\frac{-k_r}{k_r + k_q + k_p}$	0

У 1990 р. Кауфман і Роузеув запропонували вибирати на кожному рівні кластер для розщеплення з найбільшим діаметром, який обчислюється за такою формулою:

$$D_C = \max d(i_p, i_q) \quad \forall i_p, \quad i_q \in C.$$

Рекурсивний поділ кластерів триває, доки всі кластери або стануть сиглетонами (тобто з одного об'єкта), або доки всі члени одного кластера не матимуть нульову відмінність один від одного.

Неієрархічні алгоритми

Великої популярності у вирішенні завдань кластеризації набули алгоритми, засновані на пошуку оптимального у певному сенсі розподілу множини даних на кластери (групи). У багатьох завданнях завдяки своїм перевагам використовуються саме алгоритми побудови розбиття. Зазначені алгоритми намагаються згрупувати дані (у кластери) так, щоб цільова функція алгоритму розбиття досягала екстремуму (мінімуму). Розглянемо основні алгоритми кластеризації, що ґрунтуються на методах розбиття. У цих алгоритмах використовуються такі базові поняття:

• навчальна множина (вхідна множина даних) *M*, на якій будується розбиття;

• метрика відстані

$$d_A^2(m_j, c^{(i)}) = ||(m_j - c^{(i)})^t A(m_j - c^{(i)})||,$$

де матриця визначає спосіб обчислення відстані. Наприклад, для одиничної матриці *A* використовуватимемо відстань за Евклідом, вектор центрів кластерів, матрицю розбиття за кластерами *U*, цільову функцію $J = J(M, d, C, U)$, набір обмежень.

Алгоритм *k*-середніх. Розглянемо докладніше алгоритм *k*-середніх (*Hard C-Means*) [6.2] на прикладі даних класифікації ірисів. Для більшої наочності обмежимося двома параметрами — довжиною та шириною чашолистка. Це дасть змогу подати дані у двовимірному просторі (рис. 6.1). Точки позначено номерами об'єктів.

Спочатку вибирається *k* довільних вихідних центрів — точок у просторі всіх об'єктів. Не дуже критично, які саме це будуть центри, процедура вибору вихідних точок позначиться головно на тривалості обчислень. Наприклад, це можуть бути перші *k* об'єктів множини *I*, наприклад точки 1, 2 і 3.

Далі ітераційно виконують операцію двох кроків. На першому етапі всі об'єкти розбивають на *k* груп, найближчих до одного з центрів. Близькість визначається відстанню, яка обчислюється одним із описаних раніше способів (наприклад, за допомогою евклідової відстані). Рис. 6.2. ілюструє розбиття ірисів на три кластери.

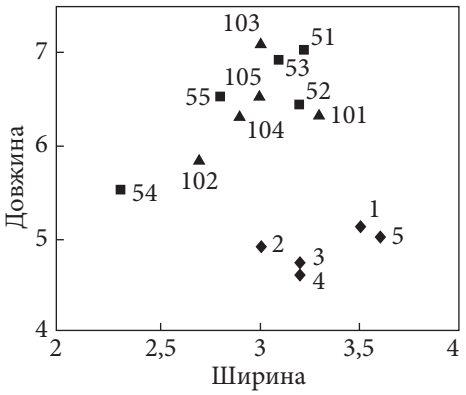


Рис. 6.1. Подання у двовимірному просторі

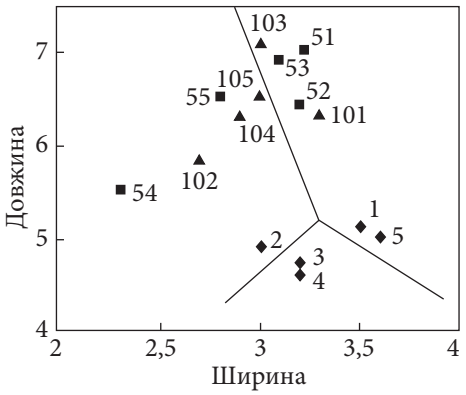


Рис. 6.2. Розбиття на три кластери

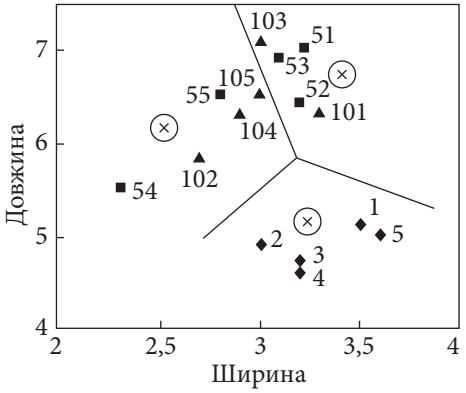


Рис. 6.3. Перерозподіл відповідно до нових центрів

На другому етапі обчислюють нові центри кластерів. Центри можна визначити як середні значення змінних об'єктів, віднесених до сформованих груп. Нові центри, звісно, можуть відрізнятись від попередніх. На рис. 6.3. позначено нові центри та новий поділ відповідно до них. Природно, що деякі точки, які раніше належали до одного кластера, за нового розбиття потрапляють до іншого кластера (у цьому прикладі такими точками є 1, 2, 5 та ін.). Нові центри позначені на рисунку символом «×», обведеним колом.

Розглянута операція повторюється рекурсивно до того часу, доки центри кластерів (відповідно, межі між ними) не перестануть змінюватися. У прикладі друге розбиття є останнім. Як випливає з результатів кластеризації, нижній кластер повністю збігається з класом *Iris setosa*. В інших кластерах є представники обох класів. У нашому випадку це сталося через кластеризацію тільки за двома параметрами, а не за всіма чотирма.

Після того, як об'єкти відображено в точці багатовимірного простору, процедура автоматичного розбиття на кластери стає дуже простою. Проблема в тому, що вихідні об'єкти, як завжди, можна подати у вигляді точок.

У геометрії всі змінні рівнозначні, у реальних даних зміна однієї з них на деяке значення за змістом може означати істотно більше, ніж така сама зміна іншої змінної. Дійсні змінні можна привести до приблизно рівнозначного масштабу, розділивши на їх характерний природний масштаб, або, якщо він невідомий, на середнє значення цієї змінної, діапазон її зміни (різниця між максимальним і мінімальним значеннями змінної) або на її стандартне відхилення. Тоді геометрична відстань між точками приблизно відповідатиме інтуїтивним уявленням про близькість об'єктів.

Насамкінець слід зазначити, що метод *k*-середніх добре працює, якщо дані за своєю природою поділяються на компактні, приблизно сферичні групи.

Цей алгоритм є прообразом практично всіх алгоритмів нечіткої кластеризації, і її розгляд допоможе кращому розумінню принципів, на яких засновано складніші алгоритми [6.2].

Опис алгоритму *k*-середніх. Базові визначення і поняття в рамках цього алгоритму мають вигляд:

- навчальна множина $M = \{m_j\}_{j=1}^{d}$, де d — кількість точок (векторів) даних;
- метрика відстані, яку розраховують за формулою;
- вектор центрів кластерів

$$C = \{c^{(i)}\}_{i=1}^{n}, \tag{6.8}$$

де

$$c^{(i)} = \frac{\sum_{j=1}^{d} u_{ij} m_j}{\sum_{j=1}^{d} u_{ij}};\ 1 \le i \le n; \tag{6.9}$$

- матриця розбиття

$$U = \{u_{ij}\},$$

де

$$u_{ij}^{(l)} = \begin{cases} 1 \text{ при } (d(m_j, c^{(i)}) = \min d(m_j, c^{(l)})), \\ 0, \text{ інакше;} \end{cases} \tag{6.10}$$

- цільова функція

$$J(M, U, C) = \sum_{i=1}^{n} \sum_{j=1}^{d} u_{ij} d_{j=1}^{2}(m_j, c^{(i)}); \tag{6.11}$$

- набір обмежень

$$u_{ij} \in \{0, 1\};\ \ \sum_{j=1}^{n} u_{ij} = 1, \tag{6.12}$$

який визначає, що кожен вектор даних може належати тільки одному кластера і не належати іншим. У кожному кластері міститься щонайменше одна точка, але з більш загальної кількості точок.

Конструктивно алгоритм є ітераційною процедурою з чотирьох кроків [6.5].

1. Проініціалізувати початкове розбиття (наприклад, випадково), вибрати точність ε (використовується в умовах завершення алгоритму), проініціалізувати номер ітерації $l = 0$.

2. Визначити центри кластерів за формулою

$$c^{(l)} = \frac{\sum_{j=1}^{d} u_{ij}^{(l-1)} m_j}{\sum_{j=1}^{d} u_{ij}^{(l-1)}}; \; 1 \le i \le n. \tag{6.13}$$

3. Оновити матрицю розбиття для того, щоб мінімізувати квадрати похибок, використовуючи формулу

$$u_{ij}^{(l)} = \begin{cases} 1 \text{ при } (d(m_j, c_k^{(i)}) = \min\limits_{l} d(m_j, c_k^{(l)})), \\ 0, \text{ інакше.} \end{cases} \tag{6.14}$$

4. Перевірити умову $||U^{(l)} - U^{(l-1)}|| < \delta$. Якщо умова виконується, завершити процес, якщо ні — перейти до кроку 2 із номером ітерації $l = l + 1$.

Основний недолік, властивий цьому алгоритму через дискретний характер елементів матриці розбиття, — великий розмір простору розбиття. Одним зі способів подолання цього недоліку є задання елементів матриці розбиття числами з одиничного інтервалу. Тобто належність елемента даних кластера має визначатися функцією належності — елемент даних може належати кільком кластерам з різним ступенем належності. У такому разі ми приходимо до задачі нечіткої кластеризації. Цей підхід втілився в алгоритмі нечіткої кластеризації — нечіткому методі *k*-середніх (*fuzzy C-means*) [6.5].

6.3. Нечіткий метод *k*-середніх
Нейронна мережа із самоорганізацією

Розглянемо нейронну мережу із самоорганізацією, де навчання проводиться без учителя. Алгоритм самоорганізації відносить вектор x до відповідного кластера даних, які зображаються центром c_i, за допомогою змагального навчання.

Базова форма алгоритму самоорганізації дає можливість точно знайти положення центрів c_i відповідних груп даних (кластерів), на які розбивається вихідний багатовимірний простір. Ці центри можуть використовуватися в гібридному алгоритмі навчання НМ як початкові значення, що істотно прискорює процес навчання і гарантує збіжність до глобального мінімуму.

Алгоритм нечіткої самоорганізації *k*-середніх

Припустимо, що в мережі є k нечітких нейронів з центрами в точках c_i ($i = 1, 2, \ldots, k$). Початкові значення цих центрів можуть бути обрані випадково з ділянок допустимих значень відповідних компонентів векторів x_j ($j = 1, 2, \ldots, p$), використаних для навчання. Нехай функція фазифікації задана у вигляді узагальненої функції Гаусса, вираженої формулою (3.42) (див. розд. 3).

Вектор x_j, що подається на вхід мережі, належатиме до різних груп, які становлять центри c_i, зі ступенем u_{ij}, причому $0 < u_{ij} < 1$, а сумарний ступінь належності до всіх груп, очевидно, дорівнює 1. Тому

$$\sum_{i=1}^{K} u_{ij} = 1 \tag{6.15}$$

для $j = 1, 2, \ldots, p$. Функцію похибки, що відповідає такому уявленню, можна визначити як суму похибок належності до центрів c_i з урахуванням ступеня належності u_{ij} [6.4]:

$$E = \sum_{i=1}^{K} \sum_{j=1}^{p} u_{ij}^{m} \| c_i - x_j \|^2, \tag{6.16}$$

де m — ваговий коефіцієнт фазифікації, який набуває значення з інтервалу $(1, \infty)$. Мета навчання із самоорганізацією полягає в такому підборі центрів c_i, щоб для заданої множини навчальних векторів x_i забезпечити досягнення мінімуму функції (6.16) за одночасного дотримання умов обмеження (6.15). Отже, виникає задача мінімізації нелінійної функції (6.16) з обмеженнями типу (6.15). Розв'язання цієї задачі можна звести до мінімізації функції Лагранжа, визначеної у вигляді [6.4]:

$$LE = \sum_{i=1}^{K} \sum_{j=1}^{N} u_{ij}^{m} \| c_i - x_j \|^2 + \sum_{j=1}^{p} \lambda_j \left(\sum_{i=1}^{K} u_{ij} - 1 \right), \tag{6.17}$$

де λ_j ($j = 1, 2, \ldots, p$) — множники Лагранжа.

У [6.4] доведено, що розв'язання задачі (6.17) можна подати у вигляді

$$c_i = \frac{\sum_{j=1}^{p} u_{ij}^{m} x_j}{\sum_{j=1}^{p} u_{ij}^{m}}; \tag{6.18}$$

$$u_{ij} = \frac{1}{\sum_{k=1}^{K} \left(\frac{d_{ij}^2}{d_{kj}^2} \right)^{\frac{1}{m-1}}}, \tag{6.19}$$

де d_{ij} — евклідова відстань між центром c_i та вектором x_j, $d_{ij} = \| c_i - x_j \|$. Оскільки точні значення центрів c_i на початку процесу невідомі, алгоритм навчання має бути ітераційним. Його можна сформулювати у такому вигляді.

1. Виконати випадкову ініціалізацію коефіцієнтів u_{ij}, вибираючи їх значення з інтервалу [0, 1] так, щоб дотримувалася умова (6.15).

2. Визначити k центрів c_i відповідно до (6.18).

3. Розрахувати значення функції похибки відповідно до виразу (6.16). Якщо її значення виявиться нижчим від встановленого порога або якщо зменшення цієї похибки щодо попередньої ітерації дуже мале, то завершити обчислення. Останні значення центрів і є потрібним рішенням. В іншому випадку перейти до п. 4.

4. Обчислити нові значення u_{ij} за формулою (6.19) і перейти до п. 2.

Таку процедуру нечіткої самоорганізації називатимемо нечітким алгоритмом *k*-середніх (*fuzzy C-means*).

Багаторазове повторення ітераційної процедури забезпечить досягнення мінімуму функції *E*, який необов'язково буде глобальним мінімумом. Якість центрів, що оцінюється значенням функції похибки *E*, істотно залежить від попереднього підбору як значень u_{ij}, так і центрів c_i. Найкращим може бути визнано таке розміщення центрів, за якого вони розташовуються в ділянках, що містять найбільшу кількість поданих векторів x_j. За такого підбору центрів вони становитимуть вектори даних x_j з найменшою сумарною похибкою. Тому початку ітераційної процедури розрахунку оптимальних значень центрів має передувати процедура їх ініціалізації. До найвідоміших алгоритмів ініціалізації належать алгоритми пікового групування та різницевого групування даних.

6.4. Визначення початкового розміщення центрів кластерів

Алгоритм пікового групування

Алгоритм пікового групування був запропонований Г.-П. Крігелем (Ганс-Пітер Кріґель; Hans-Peter Kriegel) та П. Фільовим (Павло Фільов; Pavel Filyov) [6.3, 6.7]. З використанням *N* вхідних векторів будується сітка, яка рівномірно покриває простір цих векторів. Вузли цієї сітки розглядають як потенційні вектори υ, для кожного з яких розраховують пікову функцію:

$$m(\upsilon) = \sum_{j=1}^{p} \exp\left\{\frac{-\|\upsilon - x_j\|^{2b}}{2\sigma^2}\right\}, \tag{6.20}$$

де σ — деяка константа, яку підбирають окремо для кожної конкретної задачі.

Величина $m(\upsilon)$ сприймається як оцінка висоти пікової функції. Вона пропорційна кількості векторів x_j, що потрапляють в околицю потенційного центру *v*. Велике значення $m(\upsilon)$ свідчить про те, що центр *v* розміщується в ділянці, в якій зосереджено найбільшу кількість векторів $\{x_j\}$.

Коефіцієнт σ впливає на кінцеві пропорції між $m(\upsilon)$ і *v* незначно.

Після розрахунку значень $m(\upsilon)$ для всіх потенційних центрів відбирають перший центр c_1, який має найбільше значення $m(\upsilon)$. Для вибору наступних центрів необхідно вилучити c_1 та вузли, розміщені в безпосередній близькості до c_1. Це можна зробити перевизначенням пікової функції за допомогою відокремлення від неї функції Гаусса з центром у точці c_1. Позначивши цю нову функцію через $m_{new}(\upsilon)$, отримаємо

$$m_{new}(\upsilon) = m(\upsilon) - m(c_1)\exp\left\{\frac{-\|\upsilon - c_1\|^{2b}}{2\sigma^2}\right\}. \tag{6.21}$$

Зазначимо, що ця функція має «нуль» у точці c_1.

Далі ця сама процедура повторюється з наступним центром c_2 тощо.

Процес знаходження наступних центрів c_2, c_2, … реалізується послідовно на модифікованих значеннях $m_{new}(v)$, які отримують за винятком найближчих сусідів, центри яких було знайдено на попередніх етапах. Він закінчується у момент локалізації всіх центрів.

Метод пікового групування ефективний за невеликої розмірності вектора X. Інакше кількість потенційних центрів зростає лавиноподібно.

Алгоритм різницевого групування

Алгоритм різницевого групування — це модифікація попереднього алгоритму, в якому самі вектори x_j розглядаються як потенційні центри v. Пікову функцію $D(x_i)$ у цьому випадку задають у вигляді

$$D(x_i) = \Sigma_{j=1}^{N} \exp\left(-\frac{||x_i - x_j||^2}{(r_a/2)^2}\right), \qquad (6.22)$$

де коефіцієнт r_a визначає сферу сусідства. На величину $D(x_i)$ значною мірою впливають лише ті x_j, що перебувають у межах цієї ділянки.

За високої щільності точок в околі x_i значення функції $D(x_i)$ велике. Після розрахунку значень пікової функції для кожної точки x_i відбирається вектор x, для якого міра щільності $D(x)$ виявиться найбільшою. Саме ця точка стає першим центром c_1.

Вибір наступного центру c_2 можливий після вилучення попереднього центру та всіх точок, що лежать у його околі.

Так само, як і в попередньому випадку, пікову функцію перевизначають так:

$$D_{new}(x_i) = D(x_i) - D(c_1)\exp\left(-\frac{||x_i - c_1||^{2b}}{(r_b/2)^2}\right).$$

Під час нового визначення функції D коефіцієнти r_b позначають нові значення константи, яка визначає сферу сусідства наступного центру. Зазвичай дотримуються умов $r_b \geq r_a$.

Після модифікації значення пікової функції відбувається пошук нової точки x, на якій $D_{new}(x) \to \max$. Вона стає новим центром.

Процес знаходження наступного центру відновлюється після вилучення всіх компонентів, які відповідають вже відібраним точкам. Ініціалізація закінчується в момент фіксації всіх центрів, передбачених початковими умовами.

6.5. Алгоритм нечіткого кластер-аналізу Густавсона — Кесселя

У класичному алгоритмі *k*-середніх елементи вибирають за допомогою звичайної евклідової відстані між вектором *x* та центром кластера:

$$d(x, c) = ||x - c|| = \sqrt{(x - c)^T (x - c)}.$$

За такого задання відстані між двома векторами множина точок, рівновіддалених від центра, набуває вигляду сфери з однаковим масштабом по всіх осях. Але якщо дані утворюють групи, форма яких відрізняється від сферичної або якщо шкали окремих координат вектора дуже різняться, то така метрика стає неадекватною. І тут якість кластеризації можна значно підвищити за допомогою поліпшеної версії алгоритму самоорганізації, що називається алгоритмом Густавсона — Кесселя [6.3, 6.5].

Основні зміни щодо базового алгоритму *k*-середніх складаються у введенні у формулу розрахунку метрики матриці масштабування *A*. За такого масштабування відстань між центром *c* та векторами *x* визначають за формулою

$$d(x, c) = ||x - c|| = \sqrt{(x - c)^T A(x - c)}. \quad (6.23)$$

Як матрицю масштабування зазвичай використовують додатно визначену матрицю, тобто матрицю, всі власні числа якої дійсні і додатні.

Аналогічно базовому алгоритму *k*-середніх мета навчання з використанням алгоритму Густавсона — Кесселя полягає в такому розміщенні центрів, за якого мінімізується критерій:

$$E = \sum_i \sum_j u_{ij}^m d^2(x_j, c_i). \quad (6.24)$$

Опис алгоритму Густавсона — Кесселя [6.5].

1. Провести початкове розміщення центрів у просторі даних. Створити елементарну форму матриці масштабування *A*.

2. Сформувати матрицю коефіцієнтів належності всіх векторів *x* до центрів за формулою

$$u_{ij} = \frac{1}{\sum_{k=1}^{K} \left(\frac{d^2(x_j, c_i)}{d^2(x_j, c_k)} \right)^{\frac{1}{m-1}}}. \quad (6.25)$$

3. Розрахувати нове розміщення центрів відповідно до формули

$$c_i = \frac{\sum_{j=1}^{N} u_{ij}^m x_j}{\sum_{j=1}^{N} u_{ij}^m}. \quad (6.26)$$

4. Згенерувати для кожного вектора матрицю коваріацій:

$$S_i = \Sigma\Sigma_{j=1}^{N} u_{ij}^{m} (x_j - c_i)\,(x_j - c_i)^T. \qquad (6.27)$$

5. Розрахувати нову матрицю масштабування для кожного k-го центру за формулою

$$A_i = \sqrt[n]{\det(S_i)}^{*}\, S_i^{-1}. \qquad (6.28)$$

6. Якщо останні зміни положень центрів та матриці коваріації досить малі відносно попередніх значень і не перевищують заданої величини, то завершити ітераційний процес.

6.6. Застосування нечітких методів *k*-середніх і Густавсона — Кесселя в задачах автоматичної класифікації

Розглянемо застосування нечітких методів кластерного аналізу *k*-середніх та Густавсона — Кесселя в тестових і практичних задачах 1-ї й 122-ї автоматичних класифікацій [6.5].

Вхідні дані: вибірка точок, кількість кластерів.

Вихідні дані: матриця належності U, центри кластерів.

Приклад 6.1. Вихідну вибірку точок наведено на рис. 6.4. Центри кластерів знаходимо за допомогою методів:

а) пікового групування;

б) різницевого групування.

Далі, для знаходження кластерів використовуємо метод Густавсона — Кесселя.

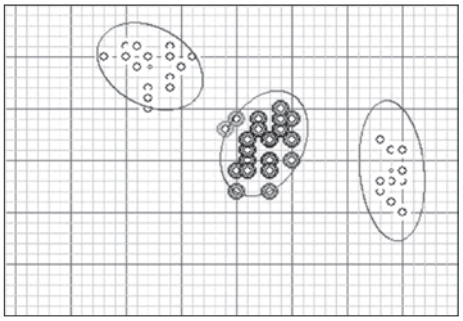


Рис. 6.4. Розміщення кластерів (приклад 6.1)

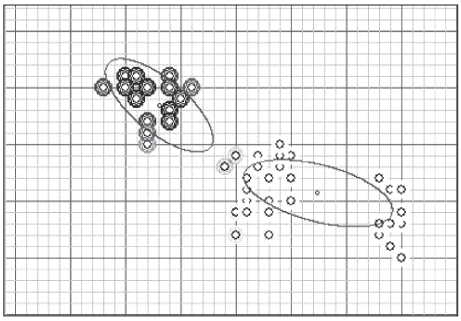


Рис. 6.5. Результат кластеризації для двох кластерів (приклад 6.1)

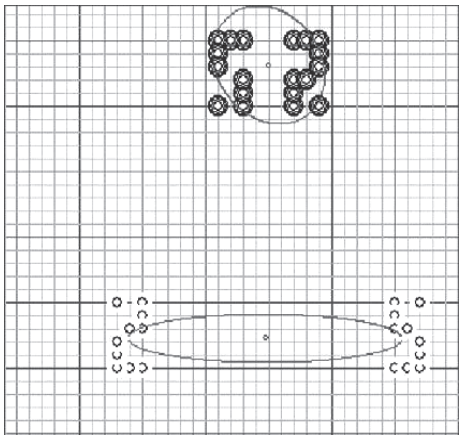


Рис. 6.6. Результат кластеризації для двох кластерів (приклад 6.2)

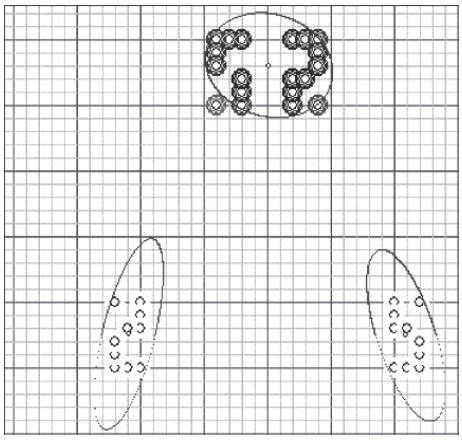


Рис. 6.7. Результат кластеризації для трьох кластерів (приклад 6.2)

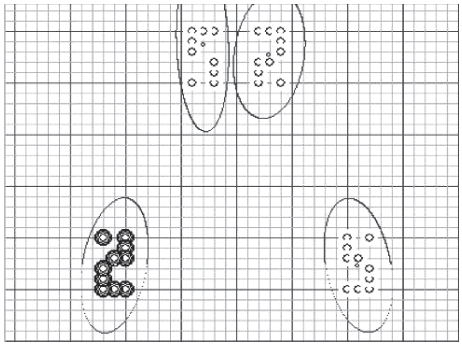


Рис. 6.8. Результат кластеризації для чотирьох кластерів (приклад 6.2)

Розміщення кластерів для цього прикладу наведено на рис. 6.4. Для випадку двох кластерів за попередньої вибірки даних результати кластеризації наведено на рис. 6.5.

Приклад 6.2. Нехай вихідна множина точок задана на рис. 6.6. Розглянемо спочатку випадок двох кластерів. Відповідні результати також наведено на рис. 6.6 [6.5]. Для трьох кластерів результати кластеризації наведено на рис. 6.7.

Для чотирьох кластерів (k = 4) відповідне рішення показано на рис. 6.8.

Приклад 6.3. Класифікація країн ООН [6.5].

Вихідні дані *Millennium Indicators* ООН наведено у табл. 6.3, центри кластерів — у табл. 6.4.

Таблиця 6.3

Вихідні дані *Millennium Indicators* ООН

Країна	Кількість населення за межею бідності, % X	Кількість дітей віком до 5 років із недостатньою вагою, % Y	Кількість грамотного населення, % W	Гендерна рівність. Частка жінок серед працівників несільськогосподарської сфери, % Z
Азербайджан	49,6	6,8	99,9	48,5
Албанія	25,4	14,3	99,4	40,3
Алжир	12,2	6	89,9	15,5
Ангола	70	30,5	71,4	26,4
Аргентина	15	5,4	98,6	47,6
Афганістан	70	48	50	17,8
Бангладеш	49,8	47,7	49,7	24,2
Бахрейн	15	8,7	97	13,4
Беліз	40	6,2	84,2	44,4
Бенін	33	22,9	55,5	46
Білорусь	41,9	2	99,8	55,9
Болгарія	12,8	2	99,7	52,2
Болівія	62,7	7,5	97,3	36,5
Боснія і Герцеговина	19,5	4,1	99,6	35,8
Ботсвана	70	12,5	89,1	47
Бразилія	17,4	5,7	96,3	46,7
Буркіна-Фасо	45,3	34,3	19,4	15,2
Бурунді	70	45,1	72,3	13,3
Бутан	70	18,7	80	12
В'єтнам	50,9	33,1	94,1	51,8
Вірменія	53,7	2,6	99,8	47
Габон	40	11,9	59	37,7
Гамбія	64	17	42,2	20,9
Гаїті	45	17,3	66,2	39,5
Гана	50	24,9	81,8	56,5
Гаяна	35	13,6	80	37,4
Гватемала	56,2	22,7	80,1	38,7
Гвінея	40	23,2	50	30,3
Гвінея-Бісау	48,7	25	44,1	10,8
Гондурас	53	16,6	88,9	50,5
Грузія	11,1	3,1	99,8	45,2
Демократична Республіка Конго	70	31,1	68,7	25,9
Джибуті	45,1	18,2	73,2	25
Домініканська Республіка	28,6	5,3	94	34,9
Еквадор	35	11,6	96,4	41,1
Екваторіальна Гвінея	50	18,6	92,7	10,5

Продовження табл. 6.3

Країна	Кількість населення за межею бідності, % X	Кількість дітей віком до 5 років із недостатньою вагою, % Y	Кількість грамотного населення, % W	Гендерна рівність. Частка жінок серед працівників несільськогосподарської сфери, % Z
Еритрея	53	39,6	60,9	35
Есватіні (Свазіленд)	40	10,3	88,1	31,3
Естонія	8,9	1	99,8	51,5
Ефіопія	44,2	47,2	57,4	39,9
Єгипет	16,7	8,6	73,2	21,6
Ємен	41,8	45,6	67,9	6,1
Замбія	72,9	28,1	81,2	29,4
Зімбабве	34,9	13	97,6	21,8
Індія	28,6	47	64,3	17,5
Індонезія	27,1	26,1	98	30,8
Ірак	35	15,9	41	11,9
Іран	30	10,9	86,3	17,2
Йорданія	11,7	4,4	99,4	24,9
Кабо-Верде	40	13,5	89,1	39,1
Казахстан	34,6	4,2	99,8	48,7
Камбоджа	36,1	45,2	80,3	52,6
Камерун	40,2	21	81,1	20,7
Кенія	52	20,2	95,8	38,5
Киргизстан	64,1	11	99,7	44,1
Китай	4,6	10	98,9	39,5
Колумбія	64	6,7	97,2	48,8
Конго	50	13,9	97,8	26,1
Коста-Рика	22	5,1	98,4	39,5
Кот-д'Івуар	59	21,2	59,8	20,2
КНДР	60	20,8	99,8	40,7
Куба	60	4,1	99,8	37,7
Кувейт	20	9,8	93,1	24,1
Лаос	38,6	40	78,5	42,1
Лесото	50	17,9	87,2	24,7
Ліберія	60	26,4	70,8	23,6
Ліван	20	3	92,1	25,9
Лівія	30	4,7	97	15
Мавританія	46,3	31,8	49,6	37
Мадагаскар	71,3	33,1	70,1	24,2
Малаві	65,3	21,9	63,2	12,5
Малайзія	40	12,4	97,2	38
Малі	63,8	33,2	24,2	35,9

Продовження табл. 6.3

Країна	Кількість населення за межею бідності, % X	Кількість дітей віком до 5 років із недостатньою вагою, % Y	Кількість грамотного населення, % W	Гендерна рівність. Частка жінок серед працівників несільськогосподарської сфери, % Z
Мальдіви	70	30,4	99,2	36,1
Марокко	19	8,9	69,5	26,2
Мексика	30	7,5	96,6	37,4
Мозамбік	69,4	23,7	62,8	11,4
Молдова	23,3	3,2	98,7	54,6
Монголія	36,3	12,7	97,7	49,4
Непал	42	48,3	70,1	11,8
Нігер	63	39,6	25,6	8,6
Нігерія	34,1	28,7	88,6	34
Нікарагуа	47,9	9,6	86,2	41,1
ОАЕ	20	14,4	91,4	14,4
Пакистан	32,6	38	53,9	8,7
Панама	37,3	6,8	96,1	44
Папуа Нова Гвінея	37,5	7	68,6	35,4
Парагвай	21,8	4,6	96,3	42
Перу	49	7,1	96,6	37,2
Польща	23,8	3	99,8	47,7
Руанда	51,2	27,2	76,5	14,6
Румунія	21,5	5,7	97,8	45,3
РФ	30,9	3	99,8	50,1
Сальвадор	48,3	10,3	88,9	31,1
Сенегал	33,4	22,7	51,5	25,7
Сербія та Чорногорія	30	1,9	99,8	44,9
США	5	1,4	99,1	48,8
Таїланд	13,1	18,6	98	46,9
Танзанія	35,7	29,4	91,6	28,5
Тринідад і Тобаго	21	5	99,8	41,3
Туніс	7,6	4	94,3	25,3
Туреччина	25	8,3	95,5	20,6
Туркменістан	30	12	99,8	20
Уганда	44	22,8	80,2	35,6
Угорщина	17,3	3	99,5	47,1
Узбекистан	27,5	7,9	99,7	41,5
Україна	31,7	3	99,9	53,6
Фіджі	40	7,9	99,3	35,9
Філіппіни	36,8	30,6	95,1	41,1
Хорватія	20	0,6	99,6	46,3

Закінчення табл. 6.3

Країна	Кількість населення за межею бідності, % X	Кількість дітей віком до 5 років із недостатньою вагою, % Y	Кількість грамотного населення, % W	Гендерна рівність. Частка жінок серед працівників несільськогосподарської сфери, % Z
Центральноафриканська Республіка	70	24,3	58,5	30,4
Чад	64	28,1	37,3	5,5
Чехія	10	1	99,8	45,8
Чилі	17	0,7	99	37,3
Шрі-Ланка	25	29,4	97	43,2
Ямайка	18,7	3,6	94,5	48

Таблиця 6.4

Центри кластерів

Кластер	Розподіл за %			
Перший	28,25	5,96	97,86	43,77
Другий	29,85	12,51	82,14	23,67
Третій	59,72	26,00	65,40	24,48
Четвертий	40,39	34,25	74,51	31,35

Як видно з даних наведених таблиць, до першого кластера належать країни з досить високими показниками (порівняно з іншими країнами вибірки). Це країни ЄС, Східної Європи, Балкан, Північної та Латинської Америки.

До другого кластера відносять країни, показники яких нижчі. Це країни Північної Африки та Близького Сходу. У другому кластері найнижчий показник рівня гендерної рівності.

До третього кластера належать найбідніші країни з найнижчим рівнем грамотності, а також із низьким рівнем гендерної рівності. Здебільшого це африканські країни.

До четвертого кластера віднесено бідні країни з найнесприятливішими умовами для життя дітей.

Приклад 6.4. Класифікація країн ООН за показниками сталого розвитку.

Проведемо дослідження методу кластеризації нечіткого методу *k*-середніх за показниками сталого розвитку країн ООН. Для цього використаємо дані Світового центру даних України в КПІ *WDC*. Як показники сталого розвитку візьмемо такі:

- *Gini* — індекс Джині;
- *Ihd* — індекс стану здоров'я;
- *Iql* — індекс рівня життя;
- *Isd* — індекс сталого розвитку.

Алгоритмом попереднього розміщення центрів слугує алгоритм різницевого групування.

Кластеризацію проведемо для різної кількості кластерів $K = \overline{2,5}$. Крім значення критерію, що оптимізується, якість розбиття критерію будемо оцінювати також за показником Хі — Бені:

$$\chi = d_{ср} / D_{ср},$$

де $d_{ср}$ — середня внутрішньокластерна відстань; $D_{ср}$ — середня міжкластерна відстань. Цей показник необхідно мінімізувати.

Експеримент 1. $K = 3$. Показники сталого розвитку та ступені належності країн до кластерів наведено у табл. 6.5.

Центри кластерів наведено у табл. 6.6.

Таблиця 6.5

Показники сталого розвитку країн ООН

Країна	Показник				Номер кластера	Ступінь належності до		
	Gini	*Ihd*	*Iql*	*Isd*		кластера 1	кластера 2	кластера 3
Австралія	0,569	0,819	0,782	0,866	1	0,900	0,028	0,070
Австрія	0,687	0,729	0,800	0,815	1	0,954	0,013	0,032
Азербайджан	0,600	0,539	0,359	0,405	3	0,130	0,247	0,621
Албанія	0,583	0,548	0,573	0,573	3	0,401	0,106	0,491
Алжир	0,567	0,484	0,265	0,350	3	0,088	0,438	0,472
Аргентина	0,352	0,630	0,763	0,494	3	0,388	0,117	0,494
Бангладеш	0,652	0,210	0,206	0,175	2	0,047	0,812	0,140
Бельгія	0,614	0,748	0,817	0,676	1	0,952	0,012	0,034
Бенін	0,498	0,178	0,334	0,208	2	0,017	0,912	0,069
Болгарія	0,361	0,583	0,699	0,518	3	0,298	0,106	0,595
Болівія	0,172	0,433	0,492	0,298	3	0,086	0,272	0,641
Боснія і Герцеговина	0,548	0,534	0,492	0,354	3	0,114	0,168	0,716
Ботсвана	0,132	0,418	0,464	0,347	3	0,090	0,273	0,636
Бразилія	0,216	0,518	0,626	0,606	3	0,214	0,131	0,654
Велика Британія	0,553	0,727	0,800	0,780	1	0,946	0,014	0,039
Венесуела	0,397	0,513	0,265	0,306	3	0,063	0,388	0,547
В'єтнам	0,520	0,332	0,265	0,317	2	0,028	0,811	0,160
Вірменія	0,655	0,512	0,334	0,474	3	0,177	0,274	0,547
Гамбія	0,324	0,141	0,265	0,235	2	0,023	0,877	0,099
Гватемала	0,219	0,316	0,437	0,325	3	0,067	0,405	0,527
Гондурас	0,167	0,376	0,359	0,269	3	0,071	0,453	0,475
Греція	0,588	0,734	0,743	0,517	1	0,740	0,062	0,196
Грузія	0,441	0,516	0,411	0,476	3	0,023	0,038	0,937
Данія	0,761	0,747	0,743	0,816	1	0,925	0,023	0,051

Продовження табл. 6.5

Країна	Показник				Номер кластера	Ступінь належності до		
	Gini	*Ihd*	*Iql*	*Isd*		кластера 1	кластера 2	кластера 3
Домініканська Республіка	0,303	0,463	0,464	0,412	3	0,023	0,058	0,917
Еквадор	0,294	0,512	0,600	0,403	3	0,088	0,104	0,807
Естонія	0,553	0,680	0,699	0,689	1	0,919	0,018	0,062
Ефіопія	0,676	0,101	0,143	0,201	2	0,063	0,772	0,163
Єгипет	0,631	0,399	0,265	0,419	2	0,103	0,486	0,410
Замбія	0,264	0,145	0,310	0,254	2	0,039	0,782	0,178
Зімбабве	0,264	0,035	0,107	0,107	2	0,061	0,754	0,184
Ізраїль	0,486	0,754	0,651	0,644	1	0,758	0,049	0,191
Індія	0,536	0,265	0,225	0,253	2	0,012	0,933	0,053
Індонезія	0,537	0,371	0,359	0,268	2	0,050	0,647	0,301
Ірландія	0,588	0,778	0,743	0,752	1	0,975	0,006	0,018
Ісландія	0,962	0,750	0,743	0,878	1	0,766	0,084	0,148
Іспанія	0,580	0,743	0,743	0,724	1	0,982	0,004	0,013
Італія	0,552	0,733	0,782	0,668	1	0,930	0,017	0,052
Йорданія	0,517	0,490	0,359	0,445	3	0,070	0,171	0,758
Казахстан	0,655	0,540	0,265	0,434	3	0,163	0,331	0,504
Камбоджа	0,380	0,236	0,157	0,204	2	0,015	0,920	0,064
Камерун	0,376	0,201	0,143	0,218	2	0,017	0,911	0,071
Канада	0,622	0,771	0,722	0,841	1	0,947	0,014	0,037
Кенія	0,317	0,211	0,287	0,234	2	0,020	0,877	0,102
Киргизстан	0,605	0,368	0,310	0,317	2	0,066	0,627	0,306
Китай	0,437	0,463	0,384	0,332	3	0,044	0,225	0,730
Кіпр	0,962	0,678	0,600	0,681	1	0,681	0,115	0,202
Колумбія	0,157	0,502	0,546	0,618	3	0,177	0,152	0,669
Коста-Рика	0,271	0,557	0,600	0,724	3	0,334	0,124	0,541
Латвія	0,558	0,621	0,699	0,655	1	0,826	0,037	0,136
Литва	0,520	0,641	0,676	0,650	1	0,767	0,046	0,186
Люксембург	0,962	0,730	0,546	0,802	1	0,690	0,112	0,196
Мадагаскар	0,325	0,178	0,206	0,191	2	0,021	0,890	0,088
Малаві	0,490	0,138	0,287	0,237	2	0,014	0,929	0,055
Малайзія	0,344	0,585	0,437	0,527	3	0,058	0,056	0,884
Марокко	0,451	0,325	0,464	0,427	3	0,062	0,263	0,674
Мексика	0,248	0,594	0,651	0,552	3	0,237	0,117	0,645
Мозамбік	0,356	0,080	0,206	0,241	2	0,029	0,863	0,106
Молдова	0,510	0,404	0,437	0,336	3	0,064	0,316	0,618
Монголія	0,542	0,402	0,334	0,305	2	0,057	0,564	0,377
Намібія	0,047	0,379	0,464	0,449	3	0,121	0,283	0,594

Продовження табл. 6.5

Країна	Показник				Номер кластера	Ступінь належності до		
	Gini	*Ihd*	*Iql*	*Isd*		кластера 1	кластера 2	кластера 3
Непал	0,324	0,172	0,287	0,303	2	0,025	0,849	0,125
Нігер	0,409	0,167	0,157	0,181	2	0,017	0,916	0,065
Нідерланди	0,655	0,773	0,782	0,775	1	0,974	0,007	0,017
Нікарагуа	0,239	0,323	0,334	0,315	2	0,051	0,547	0,400
Німеччина	0,702	0,768	0,800	0,806	1	0,949	0,015	0,035
Нова Зеландія	0,549	0,791	0,833	0,854	1	0,893	0,030	0,075
Норвегія	0,744	0,820	0,763	0,851	1	0,903	0,030	0,065
ОАЕ	0,962	0,684	0,225	0,516	1	0,363	0,281	0,355
Пакистан	0,619	0,232	0,130	0,172	2	0,044	0,821	0,133
Панама	0,239	0,601	0,573	0,600	3	0,211	0,113	0,675
ПАР	0,166	0,366	0,464	0,299	3	0,081	0,348	0,570
Парагвай	0,245	0,429	0,546	0,583	3	0,121	0,130	0,748
Перу	0,312	0,554	0,310	0,522	3	0,076	0,138	0,784
Польща	0,589	0,657	0,722	0,625	1	0,874	0,028	0,096
Португалія	0,501	0,657	0,782	0,667	1	0,837	0,037	0,125
Республіка Корея	0,641	0,759	0,743	0,610	1	0,915	0,022	0,062
Румунія	0,650	0,618	0,600	0,547	1	0,578	0,094	0,327
РФ	0,422	0,548	0,310	0,515	3	0,076	0,135	0,788
Сальвадор	0,331	0,457	0,437	0,504	3	0,021	0,041	0,937
Сенегал	0,486	0,158	0,287	0,195	2	0,013	0,936	0,050
Словаччина	0,744	0,688	0,651	0,687	1	0,890	0,031	0,078
Словенія	0,650	0,701	0,763	0,673	1	0,972	0,007	0,019
США	0,452	0,786	0,937	0,805	1	0,793	0,060	0,146
Таджикистан	0,602	0,343	0,189	0,253	2	0,045	0,783	0,170
Таїланд	0,220	0,450	0,411	0,400	3	0,055	0,177	0,766
Танзанія	0,520	0,147	0,225	0,255	2	0,015	0,928	0,056
Трінідад і Тобаго	0,463	0,573	0,464	0,385	3	0,059	0,083	0,856
Туніс	0,452	0,493	0,464	0,492	3	0,026	0,033	0,939
Туреччина	0,474	0,487	0,492	0,455	3	0,039	0,051	0,908
Уганда	0,381	0,167	0,244	0,225	2	0,010	0,945	0,044
Угорщина	0,649	0,671	0,763	0,663	1	0,952	0,012	0,035
Узбекистан	0,538	0,395	0,225	0,177	2	0,043	0,770	0,185
Україна	0,716	0,534	0,464	0,361	3	0,233	0,271	0,495
Уругвай	0,419	0,615	0,743	0,687	1	0,666	0,069	0,264
Філіппіни	0,386	0,426	0,310	0,340	3	0,043	0,411	0,544
Фінляндія	0,726	0,752	0,699	0,858	1	0,915	0,025	0,058
Франція	0,619	0,754	0,817	0,794	1	0,959	0,011	0,029
Хорватія	0,601	0,618	0,676	0,603	1	0,757	0,052	0,189

Закінчення табл. 6.5

Країна	Показник				Номер кластера	Ступінь належності до		
	Gini	*Ihd*	*Iql*	*Isd*		кластера 1	кластера 2	кластера 3
Чехія	0,744	0,717	0,722	0,702	1	0,943	0,016	0,040
Чилі	0,243	0,641	0,546	0,731	3	0,342	0,127	0,530
Швейцарія	0,600	0,756	0,722	0,851	1	0,939	0,017	0,043
Шрі Ланка	0,464	0,456	0,359	0,450	3	0,043	0,139	0,816
Ямайка	0,357	0,501	0,546	0,405	3	0,037	0,053	0,909
Японія	0,759	0,767	0,800	0,793	1	0,925	0,023	0,051

Таблиця 6.6

Центри кластерів

Номер кластера	Показник				Критерій Хі — Бені
	Gini	*Ihd*	*Iql*	*Isd*	
1	0,63499	0,724	0,73749	0,73457	—
2	0,44656	0,236	0,25218	0,24782	—
3	0,38459	0,493	0,45717	0,45412	0,438

Проаналізуємо отримані результати. До *першого* кластера потрапляють країни з найбільшими значеннями всіх показників. Це країни Західної Європи, а також низка інших. А саме: Австралія, Австрія, Бельгія, Велика Британія, Данія, Ізраїль, Ірландія, Ісландія, Іспанія, Італія, Канада, Кіпр, Латвія, Литва, Люксембург, Нідерланди, Нова Зеландія, Норвегія, Польща, Португалія, Словаччина, Словенія, США, Угорщина, Уругвай, Хорватія, Чехія, Швеція, Швейцарія.

До *другого* кластера потрапляють країни із середнім значенням показника *Gini* та мінімальними значеннями всіх інших показників. Це країни Африки та Південно-Східної Азії. До них належать: Бангладеш, В'єтнам, Єгипет, Замбія, Зімбабве, Індія, Індонезія, Камбоджа, Камерун, Киргизстан, Нігер, Нікарагуа, Пакистан, Сенегал, Таджикистан, Танзанія, Уганда та ін.

До *третього* кластера відносять країни із середніми значеннями всіх показників та малим значенням індексу *Gini*. До нього входять країни СНД, Латинської Америки та частина найрозвиненіших країн Азії і Африки: Азербайджан, Албанія, Алжир, Аргентина, Болгарія, Болівія, Боснія та Герцеговина, Бразилія, Венесуела, Вірменія, Гватемала, Гондурас, Грузія, Йорданія, Казахстан, Китай, Колумбія, Коста-Рика, Мексика, Молдова, ПАР, Парагвай, Перу, Росія, Тринідад і Тобаго, Туніс, Туреччина, Україна, Чилі, Ямайка.

Експеримент 2. $K = 4$. Ступені належності країн до кластерів наведено у табл. 6.7.

Таблиця 6.7

Ступені належності країн до кластерів

Країна	Номер кластера	Ступінь належності до кластера			
		$K = 1$	$K = 2$	$K = 3$	$K = 4$
Австралія	1	0,86029	0,070031	0,022093	0,047588
Австрія	1	0,94636	0,0251	0,008865	0,019672
Азербайджан	4	0,033017	0,073697	0,051201	0,84209
Албанія	4	0,28424	0,3116	0,072582	0,33157
Алжир	4	0,027483	0,071321	0,10306	0,79814
Аргентина	2	0,22755	0,53851	0,067084	0,16686
Бангладеш	3	0,042206	0,084422	0,63857	0,2348
Бельгія	1	0,92169	0,039019	0,011551	0,027739
Бенін	3	0,017031	0,046297	0,82568	0,111
Болгарія	2	0,14689	0,65894	0,051013	0,14315
Болівія	2	0,061197	0,50498	0,18867	0,24515
Боснія і Герцеговина	4	0,057183	0,1661	0,071758	0,70496
Ботсвана	2	0,060793	0,52973	0,18418	0,22529
Бразилія	2	0,068824	0,80042	0,041411	0,089343
Велика Британія	1	0,91502	0,044743	0,012211	0,028022
Венесуела	4	0,039499	0,16547	0,19342	0,6016
В'єтнам	4	0,027619	0,082803	0,4324	0,45718
Вірменія	4	0,070525	0,11933	0,091308	0,71884
Гамбія	3	0,011694	0,037745	0,88941	0,061153
Гватемала	2	0,052387	0,36066	0,31644	0,27051
Гондурас	3	0,054833	0,31568	0,35575	0,27374
Греція	1	0,60917	0,19229	0,051299	0,14724
Грузія	4	0,038094	0,26718	0,053444	0,64128
Данія	1	0,90361	0,042127	0,016682	0,037581
Домініканська Республіка	2	0,032441	0,67082	0,072181	0,22456
Еквадор	2	0,040697	0,79243	0,045605	0,12127
Естонія	1	0,8311	0,091463	0,020293	0,057147
Ефіопія	3	0,050831	0,094557	0,63814	0,21647
Єгипет	4	0,04537	0,097315	0,16448	0,69284
Замбія	3	0,026225	0,093996	0,75376	0,12602
Зімбабве	3	0,040152	0,099806	0,71609	0,14395
Ізраїль	1	0,61194	0,22343	0,041591	0,12304
Індія	3	0,019765	0,051379	0,75142	0,17743
Індонезія	4	0,030765	0,093396	0,26095	0,61489
Ірландія	1	0,96099	0,019763	0,005589	0,013659
Ісландія	1	0,68922	0,12112	0,063855	0,12581
Іспанія	1	0,96064	0,020311	0,005394	0,013656

Продовження табл. 6.7

Країна	Номер кластера	Ступінь належності до кластера			
		K = 1	*K* = 2	*K* = 3	*K* = 4
Італія	1	0,86916	0,06992	0,017282	0,043638
Йорданія	4	0,009514	0,034653	0,018891	0,93694
Казахстан	4	0,069722	0,12096	0,11663	0,69268
Камбоджа	3	0,009822	0,028878	0,90035	0,060951
Камерун	3	0,009908	0,028354	0,90497	0,056772
Канада	1	0,92961	0,034214	0,010931	0,025244
Кенія	3	0,013179	0,047963	0,85807	0,080792
Китай	4	0,019842	0,10456	0,078945	0,79666
Киргизстан	4	0,03518	0,085876	0,22919	0,64975
Кіпр	1	0,57037	0,14719	0,085695	0,19675
Колумбія	2	0,066882	0,7671	0,056491	0,10953
Коста-Рика	2	0,16698	0,6328	0,060572	0,13965
Латвія	1	0,69365	0,16621	0,035339	0,1048
Литва	1	0,61283	0,2234	0,040208	0,12357
Люксембург	1	0,58655	0,1471	0,084004	0,18234
Мадагаскар	3	0,010267	0,031539	0,90351	10,054684
Малаві	3	0,011209	0,029699	0,89214	0,066953
Малайзія	2	0,049112	0,71574	0,043663	0,19148
Марокко	4	0,048106	0,2518	0,17008	0,53001
Мексика	2	0,079655	0,78924	0,03832	0,092786
Мозамбік	3	0,014971	0,041838	0,87376	0,069428
Молдова	4	0,024628	0,095535	0,092677	0,78716
Монголія	4	0,022514	0,068667	0,14794	0,76088
Намібія	2	0,075055	0,54377	0,17888	0,20229
Непал	3	0,017185	0,062621	0,81948	0,10071
Нігер	3	0,008523	0,022791	0,92278	0,045911
Нідерланди	1	0,9755	0,011725	0,003857	0,008913
Нікарагуа	3	0,042242	0,23973	0,4443	0,27372
Німеччина	1	0,94112	0,027184	0,009862	0,021834
Нова Зеландія	1	0,84821	0,07761	0,023867	0,050316
Норвегія	1	0,87596	0,055127	0,021828	0,047085
ОАЕ	4	0,25172	0,18882	0,17458	0,38488
Пакистан	3	0,039064	0,080179	0,66428	0,21648
Панама	2	0,067097	0,81058	0,034832	0,087487
ПАР	2	0,060062	0,42837	0,25723	0,25433
Парагвай	2	0,039361	0,83131	0,041076	0,088257
Перу	2	0,065334	0,46554	0,10782	0,3613

Закінчення табл. 6.7

Країна	Номер кластера	Ступінь належності до кластера			
		K = 1	*K* = 2	*K* = 3	*K* = 4
Польща	1	0,76646	0,12003	0,028969	0,084542
Португалія	1	0,71572	0,16448	0,033888	0,085911
Республіка Корея	1	0,85087	0,070721	0,021213	0,057195
Румунія	1	0,42991	0,22585	0,068687	0,27555
РФ	4	0,058523	0,27694	0,090317	0,57422
Сальвадор	2	0,03225	0,70707	0,054953	0,20573
Сенегал	3	0,00987	0,026109	0,90456	0,059459
Словаччина	1	0,82228	0,073597	0,027661	0,076458
Словенія	1	0,94376	0,027013	0,008162	0,02106
США	1	0,69788	0,16177	0,047239	0,093105
Таджикистан	3	0,040625	0,091683	0,48922	0,37848
Таїланд	2	0,045151	0,57189	0,13572	0,24724
Танзанія	3	0,013224	0,032817	0,87388	0,080083
Тринідад і Тобаго	4	0,052559	0,26387	0,06346	0,62011
Туніс	4	0,049968	0,38186	0,056285	0,51189
Туреччина	4	0,050238	0,3103	0,058481	0,58098
Уганда	3	0,002341	0,00716	0,97664	0,013863
Угорщина	1	0,90577	0,04526	0,013551	0,035419
Узбекистан	3	0,038792	0,098392	0,4853	0,37751
Україна	4	0,11937	0,15783	0,11911	0,60369
Уругвай	1	0,48376	0,33651	0,051159	0,12857
Філіппіни	4	0,029716	0,15589	0,21217	0,60223
Фінляндія	1	0,88749	0,050062	0,019109	0,043343
Франція	1	0,94945	0,024991	0,007902	0,017653
Хорватія	1	0,61489	0,18965	0,045698	0,14976
Чехія	1	0,91329	0,037491	0,013955	0,03526
Чилі	2	0,18624	0,58983	0,067466	0,15646
Шрі-Ланка	4	0,015491	0,078825	0,040186	0,8655
Швейцарія	1	0,91403	0,042608	0,013135	0,03023
Швеція	1	0,85291	0,064036	0,025878	0,057175
Ямайка	2	0,038649	0,71492	0,050335	0,19609
Японія	1	0,90616	0,041501	0,016386	0,035957

Центри кластерів наведено у табл. 6.8.

Цікаво проаналізувати динаміку зміни кластерів при переході від *K* = 3 до *K* = 4. До першого кластера потрапляють країни з найбільшими значеннями всіх показників. Склад цього кластера мало змінився. У

другому кластері виявляються країни з мінімальним значенням індексу Джині та середніми значеннями решти всіх показників. До нього потрапили країни Латинської Америки: Аргентина, Бразилія, Панама, Парагвай, Перу, Уругвай та ін.

До третього кластера потрапляють країни з мінімальними значеннями всіх показників, крім індексу Джині. До нього належать країни з другого кластера попередньої кластеризації за $K = 3$: Бангладеш, Бенін, Замбія, Зімбабве, Індія, Камбоджа, Камерун, Кенія, Мозамбік, Непал, Пакистан, Сенегал, Таджикистан, Танзанія, Узбекистан. До четвертого кластера потрапили країни із середніми значеннями всіх показників. Сюди виокремилися країни з третього кластера попередньої кластеризації: Азербайджан, Венесуела, В'єтнам, Грузія, Індонезія, Йорданія, Киргизстан, Російська Федерація, Україна, Шрі-Ланка. Значення показника Хі — Бені зменшилося з 0,438 до 0,39492.

Експеримент 3. $K = 5$. Ступені належності країн до кластерів наведено у табл. 6.9. Центри кластерів наведено у табл. 6.10.

Таблиця 6.8

Центри кластерів

Номер кластера	Показник				Критерій Хі — Бені
	Gini	*Ihd*	*Iql*	*Isd*	
1	0,64429	0,73332	0,74677	0,74678	—
2	0,29515	0,51657	0,53844	0,51317	—
3	0,419	0,19409	0,23834	0,22998	—
4	0,51726	0,46924	0,3707	0,39482	0,39492

Таблиця 6.9

Ступені приналежності країн до кластерів

Країна	Номер кластера	Ступінь належності до кластера				
		$K = 1$	$K = 2$	$K = 3$	$K = 4$	$K = 5$
Австралія	4	0,028978	0,2119	0,014149	0,70964	0,035335
Австрія	4	0,006161	0,053511	0,002912	0,93078	0,006639
Азербайджан	1	0,78956	0,051485	0,049416	0,028095	0,081446
Албанія	2	0,18835	0,47345	0,04663	0,13521	0,15636
Алжир	1	0,82338	0,028074	0,069284	0,016971	0,062292
Аргентина	2	0,12139	0,41437	0,053003	0,13406	0,27718
Бангладеш	3	0,24678	0,049573	0,57541	0,034882	0,093353
Бельгія	2	0,027003	0,49013	0,011914	0,43962	0,031328

Продовження табл. 6.9

Країна	Номер кластера	Ступінь належності до кластера				
		K = 1	*K* = 2	*K* = 3	*K* = 4	*K* = 5
Бенін	3	0,1243	0,023193	0,77719	0,015018	0,060299
Болгарія	2	0,12724	0,36517	0,050096	0,10752	0,34997
Болівія	5	0,15768	0,063974	0,12383	0,034958	0,61956
Боснія і Герцеговина	1	0,59253	0,10515	0,07106	0,047525	0,18373
Ботсвана	5	0,1424	0,061352	0,11911	0,034341	0,6428
Бразилія	5	0,11748	0,17707	0,059091	0,080593	0,56577
Велика Британія	4	0,026047	0,38307	0,012041	0,54546	0,033388
Венесуела	1	0,55544	0,050154	0,15722	0,029262	0,20793
В'єтнам	1	0,51889	0,033172	0,33294	0,020621	0,094378
Вірменія	1	0,65482	0,093737	0,080793	0,055977	0,11467
Гамбія	3	0,059145	0,014193	0,87059	0,009355	0,046712
Гватемала	5	0,19463	0,057787	0,22428	0,032922	0,49038
Гондурас	5	0,21239	0,061992	0,26879	0,036927	0,4199
Греція	2	0,061036	0,67659	0,022818	0,17262	0,066937
Грузія	1	0,49314	0,074801	0,056708	0,034363	0,34099
Данія	4	0,011673	0,073903	0,005405	0,89772	0,011302
Домініканська Республіка	5	0,047578	0,013529	0,01688	0,006503	0,91551
Еквадор	5	0,10619	0,075517	0,043915	0,032134	0,74225
Естонія	2	0,007059	0,93195	0,002709	0,049494	0,008785
Ефіопія	3	0,21701	0,056798	0,58356	0,041937	0,1007
Єгипет	1	0,70189	0,050202	0,12513	0,031987	0,09079
Замбія	3	0,12101	0,032736	0,70388	0,021214	0,12116
Зімбабве	3	0,13452	0,044755	0,67795	0,03214	0,11063
Ізраїль	2	0,045296	0,72284	0,016661	0,15281	0,0624
Індія	3	0,21322	0,02654	0,67643	0,017483	0,066324
Індонезія	1	0,65009	0,036307	0,19294	0,021251	0,099406
Ірландія	4	0,01947	0,31353	0,008457	0,63558	0,022959
Ісландія	4	0,07351	0,22394	0,038218	0,60069	0,063649
Іспанія	2	0,01948	0,53592	0,008205	0,41307	0,023321
Італія	2	0,021394	0,73864	0,009059	0,20366	0,027251
Йорданія	1	0,86615	0,024928	0,029187	0,012934	0,066803
Казахстан	1	0,65081	0,083645	0,097773	0,053421	0,11435
Камбоджа	3	0,064649	0,012421	0,87745	0,008345	0,03713
Камерун	3	0,056502	0,011737	0,88974	0,007985	0,034038
Канада	4	0,012706	0,11169	0,005799	0,85549	0,014317
Кенія	3	0,085486	0,017889	0,81697	0,011408	0,068244

Продовження табл. 6.9

Країна	Номер кластера	Ступінь належності до кластера				
		K = 1	*K* = 2	*K* = 3	*K* = 4	*K* = 5
Китай	1	0,72329	0,030965	0,072053	0,016174	0,15752
Киргизстан	1	0,6945	0,037965	0,16324	0,023386	0,08091
Кіпр	4	0,13101	0,29413	0,058053	0,42667	0,090131
Колумбія	5	0,11949	0,12809	0,066169	0,066603	0,61964
Коста-Рика	5	0,13153	0,29453	0,061883	0,14011	0,37195
Латвія	2	0,010631	0,93104	0,003899	0,041344	0,013081
Литва	2	0,015094	0,91235	0,005382	0,046843	0,02033
Люксембург	4	0,11814	0,26437	0,055413	0,47491	0,087171
Мадагаскар	3	0,052205	0,012037	0,88991	0,008051	0,037801
Малаві	3	0,072576	0,014571	0,86586	0,009689	0,037302
Малайзія	5	0,18422	0,10273	0,048675	0,045563	0,61882
Марокко	1	0,44044	0,069601	0,14402	0,035961	0,30999
Мексика	5	0,12	0,22376	0,054203	0,089499	0,51253
Мозамбік	3	0,063756	0,016691	0,86151	0,011458	0,046589
Молдова	1	0,74294	0,037131	0,081522	0,019287	0,11912
Монголія	1	0,81266	0,022268	0,091717	0,012958	0,060397
Намібія	5	0,14794	0,083903	0,13474	0,049651	0,58377
Непал	3	0,10326	0,022698	0,7737	0,014567	0,085779
Нігер	3	0,042468	0,009308	0,91693	0,006392	0,024898
Нідерланди	4	0,005495	0,060511	0,002504	0,92544	0,00605
Нікарагуа	3	0,22752	0,050302	0,34747	0,029938	0,34477
Німеччина	4	0,005211	0,041855	0,002465	0,94495	0,005515
Нова Зеландія	4	0,033254	0,25806	0,016601	0,64987	0,042208
Норвегія	4	0,015959	0,090139	0,007705	0,87006	0,016139
ОАЕ	1	0,31116	0,20901	0,13628	0,19543	0,14811
Пакистан	3	0,22783	0,04559	0,60453	0,032471	0,089574
Панама	5	0,12259	0,19245	0,053521	0,083782	0,54766
ПАР	5	0,17557	0,064401	0,17815	0,036535	0,54535
Парагвай	5	0,1021	0,086319	0,051571	0,041603	0,7184
Перу	5	0,2766	0,086255	0,08901	0,04745	0,50069
Польща	2	0,006716	0,9508	0,002487	0,032341	0,007654
Португалія	2	0,024919	0,8037	0,010584	0,12464	0,036159
Республіка Корея	2	0,034625	0,6304	0,013663	0,28535	0,035967
Румунія	2	0,13736	0,55592	0,037472	0,1681	0,10116
РФ	1	0,47226	0,088606	0,083782	0,047758	0,3076
Сальвадор	5	0,10343	0,033001	0,031414	0,015763	0,81639
Сенегал	3	0,064919	0,012897	0,88056	0,008556	0,033071

Закінчення табл. 6.9

Країна	Номер кластера	Ступінь належності до кластера				
		K = 1	*K* = 2	*K* = 3	*K* = 4	*K* = 5
Словаччина	4	0,05326	0,38994	0,0203	0,4918	0,044693
Словенія	2	0,022477	0,58069	0,009256	0,36362	0,02395
США	4	0,061269	0,38077	0,032707	0,43818	0,087079
Таджикистан	1	0,41368	0,047503	0,40667	0,031929	0,10022
Таїланд	5	0,10757	0,033414	0,061378	0,017869	0,77977
Танзанія	3	0,086925	0,016687	0,845	0,011364	0,040024
Тринідад і Тобаго	1	0,48872	0,10126	0,063708	0,043981	0,30233
Туніс	5	0,39726	0,1015	0,057769	0,042609	0,40086
Туреччина	1	0,45056	0,10492	0,060609	0,043032	0,34088
Уганда	3	0,011571	0,002392	0,97706	0,00158	0,007398
Угорщина	2	0,024219	0,66403	0,009861	0,27619	0,025696
Узбекистан	1	0,40725	0,046811	0,40326	0,030358	0,11233
Україна	1	0,51689	0,15521	0,10029	0,087653	0,13996
Уругвай	2	0,058678	0,64784	0,02532	0,16176	0,1064
Філіппіни	1	0,55129	0,038277	0,16791	0,021665	0,22087
Фінляндія	4	0,014606	0,08816	0,00673	0,87602	0,014486
Франція	4	0,014257	0,15667	0,006725	0,80564	0,016709
Хорватія	2	0,032738	0,83558	0,010896	0,086855	0,033932
Чехія	4	0,02894	0,26282	0,012032	0,66966	0,026553
Чилі	5	0,13941	0,28582	0,064938	0,15059	0,35924
Шрі-Ланка	1	0,76949	0,031705	0,048229	0,016455	0,13412
Швейцарія	4	0,018516	0,16575	0,008493	0,78572	0,021522
Швеція	4	0,022142	0,11413	0,010434	0,83181	0,021489
Ямайка	5	0,13726	0,061349	0,040382	0,025552	0,73546
Японія	4	0,013436	0,089068	0,006384	0,87773	0,013385

Таблиця 6.10

Центри кластерів

Номер кластера	Показник				Критерій Хі — Бені
	Gini	*Ihd*	*Iql*	*Isd*	
1	0,52645	0,45648	0,35255	0,37995	—
2	0,56042	0,67078	0,70818	0,64717	—
3	0,41643	0,18501	0,23361	0,22605	—
4	0,68519	0,75586	0,75276	0,80007	—
5	0,27997	0,47653	0,48824	0,45777	0,38816

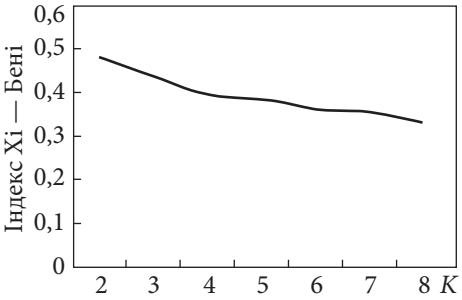


Рис. 6.9. Залежність індексу Хі — Бені від числа кластерів K

На рис. 6.9 показано залежність індексу Хі — Бені (X.L. Xie (Ши Ляо Се, Xie Liao), G. Beni (Гуїдо Бені, Guido Beni)) від числа кластерів K. Як видно з наведеного графіка, значення показника Хі — Бені помітно зменшується за $K = 2, 4$, а потім його значення змінюється слабо. Тому оптимальну кількість кластерів слід прийняти $K = 4$.

6.7. Адаптивні робастні алгоритми кластеризації

Алгоритм можливісної кластеризації

Основні недоліки імовірнісного підходу (алгоритм нечітких середніх) пов'язані з обмеженнями (6.15). У найпростішому випадку двох кластерів ($m = 2$) легко побачити, що спостереження x_k, яке однаково належить обом кластерам, і спостереження x_p, яке не належить жодному з них, можуть мати однакові рівні належності:

$$w_{(k,1)} = w_{(k,2)} = w_{(p,1)} = w_{(p,2)} = 0{,}5.$$

Природно, що ця обставина, яка знижує точність класифікації, призвела до появи ***можливісного (possibilistic) підходу*** до нечіткої класифікації [6.8]. В алгоритмі можливісної кластеризації цільова функція має вигляд

$$E(w_{k,j}, c_j) = \sum_{j=1}^{m}\sum_{k=1}^{N} w_{k,j}^{\beta}\, d^2(x_k, c_j) + \sum_{j=1}^{m} \mu_j \left(\sum_{k=1}^{N}(1 - w_{k,j})^{\beta}\right), \qquad (6.29)$$

де скалярний параметр $\mu_j > 0$ визначає відстань, на якій рівень членства набуває значення 0,5, тобто якщо $d^2(x_k, c_j) = \mu_j$, то $w_{k,j} = 0{,}5$.

Мінімізація (6.29) за допомогою $w_{k,j}$, c_j, μ_j дає очевидний розв'язок:

$$w_{k,j} = \left(1 + \left(\frac{d^2(x_k, c_j)}{\mu_j}\right)^{\frac{1}{\beta - 1}}\right)^{-1}; \qquad (6.30)$$

$$c_j = \frac{\sum_{k=1}^{N} w_{k,j}^{\beta} x_k}{\sum_{k=1}^{N} w_{k,j}^{\beta}}; \qquad (6.31)$$

$$\mu_j = \frac{\sum_{k=1}^{N} w_{k,j}^{\beta} d^2(x_k, c_j)}{\sum_{k=1}^{N} w_{k,j}^{\beta}}. \qquad (6.32)$$

Видно, що можливісний та ймовірнісний алгоритми дуже подібні і переходять один в одного за допомогою заміни виразу (6.29) на формулу (6.17) і навпаки.

Спільним недоліком розглянутих алгоритмів є їх обчислювальна складність і неможливість роботи в реальному часі. Алгоритм (6.17)—(6.19) починається з початкового завдання (нормальної випадкової) матриці розбиття W^0. На основі її значень обчислюється початковий набір прототипів c_j^0, який потім використовується для обчислення нової матриці W^1. Далі ця процедура триває і отримується послідовність розв'язків $c_j^1 W^2$, …, W^t, $c_j^t W^{t+1}$ тощо, доки різниця $||W^{t+1} - W^t||$ не стане меншою за заданий поріг ε. Тому всю наявну вибірку даних обробляють повторно.

Розв'язок, отриманий за допомогою можливісного алгоритму, рекомендується використовувати як початкові умови для імовірнісного алгоритму (6.30)—(6.32). Параметр відстані μ_j ініціалізується відповідно до (6.32) за результатами імовірнісного алгоритму.

Рекурентні алгоритми нечіткої кластеризації

Аналіз виразу (6.17) показує, що для розрахунку рівнів членства $w_{k,j}$ замість лагранжіана (6.17) можна використати його в локальній модифікації:

$$L(w_{k,j}, c_j, \lambda_k) = \sum_{j=1}^{m} w_{k,j}^{\beta} d^2(x_k, c_j) + \lambda_k(\sum_{j=1}^{m} w_{k,j} - 1). \qquad (6.33)$$

Оптимізація виразу (6.33) за процедурою Ерроу — Гурвіча — Узави (Кеннет Джозеф Ерроу; Kenneth Joseph Arrow), (Леонід Гурвіч; Leonid Hurwicz), (Хірофумі Узава; Hirofumi Uzawa) приводить до алгоритму:

$$w_{k,j} = \frac{d^2(x_k, c_{k,j})^{\frac{1}{\beta - 1}}}{\sum_{l=1}^{m} (d^2(x_k, c_{k,l}))^{\frac{1}{\beta - 1}}}; \qquad (6.34)$$

$$c_{k+1,j} = c_{k,j} - \eta_k \nabla_{cj} L_k (w_{k,j}, c_{k,j}, \lambda_k) = c_{k,j} - \eta_k w_{k,j}^{\beta} d(x_{k+1}, c_{k,j}) \nabla c_j. \qquad (6.35)$$

Процедура (6.34), (6.35) близька до алгоритму навчання Чанга — Лі (Ши-Фу Чанг; Shih-Fu Chang), (Цун-Хан Лі; Tsung-Han Lee), а для $\beta = 2$ збігається з градієнтною процедурою кластеризації Парка — Деггера (Сан-Вун Парк; Sang-Woon Park), (Рудольф Деггер; Rudolf Degger) [6.9]:

$$w_{k,j} = \frac{||x_k, c_{k,j}||^{-2}}{\sum_{l=1}^{m} ||x_k, c_{k,l}||^{-2}}; \qquad (6.36)$$

$$c_{k+1,j} = c_{k,j} - \eta_k w_{k,j}^2 (x_{k+1}, c_{k,j}). \qquad (6.37)$$

У рамках можливісного підходу локальний критерій набуває вигляду

$$E_k(w_{k,j}, c_j) = \sum_{j=1}^{m} w_{k,j}^{\beta} d^2(x_k, c_j) + \sum_{j=1}^{m}(1 - w_{k,j})^{\beta}, \tag{6.38}$$

а результат його оптимізації має вигляд

$$w_{k,j} = \left(1 + \left(\frac{d^2(x_k, c_{k,j})}{\mu_j}\right)^{\frac{1}{\beta-1}}\right)^{-1}; \tag{6.39}$$

$$c_{k+1,j} = c_{k,j} - \eta_k w_{k,j}^{\beta} d(x_{k+1}, c_{k,j}) \nabla c_j, \tag{6.40}$$

де параметр відстані μ_j ініціалізований згідно з (6.32).

У цьому випадку N у рівнянні (6.32) — це обсяг набору даних, який використовується для ініціалізації. У квадратичному випадку алгоритм (6.39), (6.40) перетворюється на досить просту процедуру і результат оптимізації має вигляд

$$w_{k,j} = \frac{\mu_j}{\mu_j + ||x_k, c_{k,j}||^2}, \tag{6.41}$$

де μ_j— параметр відстані, який ініціалізується результатами імовірнісної кластеризації (наприклад, за допомогою алгоритму нечітких k-середніх (6.17)—(6.19) згідно з рівнянням):

$$\mu_j = \frac{\sum_{k=1}^{N} w_{k,j}^2 d^2 ||x_k, c_j||^2}{\sum_{k=1}^{N} w_{k,j}^2}. \tag{6.42}$$

Робастні адаптивні алгоритми ймовірнісної нечіткої кластеризації

Розглянуті методи кластеризації можуть ефективно розв'язувати задачу класифікації зі значним перетином кластерів, однак вони припускають, що дані всередині кожного кластера розміщені досить компактно, без різких (аномальних) викидів.

Однак слід зазначити, що фактичні дані зазвичай спотворені викидами, частка яких, за деякими оцінками [6.9], становить до 20 %, тому говорити про компактне розміщення даних не завжди коректно. У зв'язку з цим останнім часом багато уваги приділяється проблемам нечіткого кластерного аналізу даних, розподіл щільності яких відрізняється від нормального наявністю «важких хвостів» [6.10, 6.11].

Робастний рекурсивний алгоритм для ймовірнісної нечіткої кластеризації. Після стандартизації компонент векторів ознак у такий спосіб, щоб усі вихідні вектори належали одиничному гіперкубу $[0, 1]^n$, будується цільова функція

$$E(w_{k,j}, c_j = \sum_{j=1}^{m} \sum_{k=1}^{N} w_{k,j}^{\beta} D(x_k, c_j) \tag{6.43}$$

в умовах обмежень

$$\sum_{j=1}^{m} w_{k,j} = 1, \quad k = 1, \ldots, N; \tag{6.44}$$

$$0 < \sum_{k=1}^{N} w_{k,j} \le N, \quad j = 1, \ldots, m. \tag{6.45}$$

Тут $D(x_k, c_j)$ — відстань між x_k та c_j у прийнятій метриці. Результатом кластеризації вважається $N \times m$ матриця $W = \{w_{k,j}\}$, яку називають «матрицею нечіткої декомпозиції». Зазвичай як функцію відстані $D(x_k, c_j)$ застосовують метрику Мінковського L^p:

$$D(x_k, c_j) = \left(\sum_{i=1}^{n} |x_{k,j} - c_{j,i}|^p\right)^{1/p}, \ p \ge 1, \tag{6.46}$$

де $x_{k,i}$, $c_{j,i}$ — i-ті компоненти $(n \times 1)$-векторів x_k, c_j відповідно.

Оцінки, пов'язані з квадратичними цільовими функціями, є оптимальними, коли дані належать до класу розподілів зі скінченною дисперсією, найвідомішим членом якого є гауссів.

Варіювання параметром p дає можливість покращити властивості робастності процедур кластеризації, однак якість оцінки визначається типом розподілу даних. Так, оцінки з $p = 1$ є оптимальними для лапласового розподілу даних, але їх побудова пов'язана з великими обчислювальними витратами. Досить реалістичним є клас наближених нормальних розподілів [6.11].

Наближено нормальний розподіл є сумішшю гауссової щільності та розподілу деякої довільної щільності, яка спотворює нормальний розподіл викидами. Оптимальна цільова функція в цьому випадку є квадратично-лінійною і наближається до лінійної зі збільшенням відстані від мінімуму.

Найяскравішим представником наближеної функції щільності нормального розподілу є

$$p(x_i, c_i) = Se(c_i, s_i) = \frac{1}{2s_i} \operatorname{sech}^2 \frac{x_i - c_i}{s_i}, \tag{6.47}$$

де c_i, s_i — параметри, що визначають центр і ширину розподілу.

Ця функція нагадує гауссову в околі центру, проте має більш важкі «хвости». З розподілом (6.47) пов'язана цільова функція

$$f_i(x_i, c_i) = \beta_i \ln \cosh \frac{x_i - c_i}{\beta_i}, \tag{6.48}$$

де параметр β_i визначає крутизну цієї функції, причому в околі мінімуму ця функція дуже близька до квадратичної, прагнучи зі зростанням X до лінійної.

Також цікавим є той факт, що похідна цієї функції

$$f_i'(x_i) = \varphi(x_i) = \tanh \frac{x_i}{\beta_i} \tag{6.49}$$

є стандартною функцією активації штучних нейронних мереж (*Back propagation*, див. розд. 1). Використовуючи як метрику структуру

$$D^R(x_k, c_j) = \sum_{i=1}^{n} f_i(x_{k,j}, c_{j,i}) = \sum_{i=1}^{n} \beta_i \ln \cosh \frac{x_{k,i} - c_{j,i}}{\beta_i}, \tag{6.50}$$

можна ввести цільову функцію робастної класифікації [6.11]

$$E^R(x_{k,j}, c_j) = \sum_{k=1}^{N} \sum_{j=1}^{m} w_{k,j}^{\beta} D^2(x_k, c_j) = \sum_{k=1}^{N} \sum_{j=1}^{m} w_{k,j}^{\beta} \sum_{i=1}^{n} \beta_i \ln \cosh \frac{x_{k,i} - c_{j,i}}{\beta_i} \tag{6.51}$$

і відповідний лагранжіан

$$L = \sum_{k=1}^{N} \sum_{j=1}^{m} w_{k,j}^{\beta} \sum_{i=1}^{n} \beta_i \ln \cosh \frac{x_{k,i} - c_{j,i}}{\beta_i} + \sum_{k=1}^{N} \lambda_k (\sum_{j=1}^{m} w_{k,j} - 1), \tag{6.52}$$

де λ_k — невизначений множник Лагранжа, що забезпечує виконання обмежень (6.44), (6.45). Сідлову точку лагранжіана (6.52) можна знайти, розв'язавши систему рівнянь Куна — Такера (Гарольд В. Кун; Harold W. Kuhn), (Альберт В. Таккер; Albert W. Tucker):

$$\begin{cases} \dfrac{\partial L(w_{k,j}, c_j, \lambda_k)}{\partial w_{k,j}} = 0, \\ \dfrac{\partial L(w_{k,j}, c_j, \lambda_k)}{\partial \lambda_k} = 0, \\ \nabla c_j L(w_{k,j}, c_j, \lambda_k) = 0. \end{cases} \tag{6.53}$$

Розв'язання першого і другого рівнянь приводять до добре відомих результатів:

$$\begin{cases} w_{k,j} = \dfrac{(D^R(x_k, c_j))^{\frac{1}{1-\beta}}}{\sum_{l=1}^{m} (D^R(x_k, c_j))^{\frac{1}{1-\beta}}}, \\ \lambda_k = \left(\sum_{l=1}^{m} (\beta D^R(x_k, c_l))^{\frac{1}{1-\beta}} \right)^{1-\beta}. \end{cases} \tag{6.54}$$

Однак третє рівняння

$$\nabla c_j L(w_{k,j}, c_j, \lambda_k) = \sum_{k=1}^{N} w_{k,j}^{\beta} \nabla c_j D^R(x_k, c_j) = 0, \tag{6.55}$$

очевидно, не має аналітичного розв'язку. Розв'язок рівняння (6.55) можна отримати за допомогою локальної модифікації лагранжіана та рекурентного алгоритму нечіткої кластеризації [6.13].

Пошук локальної сідлової точки лагранжіана

$$L_k(w_{k,j}, c_j, \lambda_k) = \sum_{j=1}^{m} w_{k,j}^{\beta} D^R(x_k, c_j) + \lambda_k \sum_{j=1}^{m} w_{k,j} - 1) \tag{6.56}$$

з використанням процедур Ерроу — Гурвіца — Удзави (Кеннет Джозеф Ерроу; Kenneth Joseph Arrow), (Леонід Гурвіц; Leonid Hurwicz), (Хірофумі Узава; Hirofumi Uzawa) приводить до алгоритму

$$\begin{cases} w_{k,j}^{pr} = \dfrac{(D^R(x_k, c_j))^{\frac{1}{1-\beta}}}{\sum_{l=1}^{m}(D^R(x_k, c_j))^{\frac{1}{1-\beta}}}, \\ c_{k+1,j,i} = c_{k,j} - \eta_k \dfrac{\partial L(w_{k,j}, c_j, \lambda_k)}{\partial c_{j,i}} = c_{k,j,i} - \eta_k w_{k,j}^{\beta} \tanh \dfrac{x_{k,i} - c_{k,j,i}}{\beta_i}. \end{cases} \tag{6.57}$$

де η_k — параметр швидкості навчання; $c_{k,j,i}$ — i-й компонент j-го прототипу, обчислений на k-му кроці.

Однак, незважаючи на невелику обчислювальну складність, алгоритм (6.57) має недолік, притаманний усім імовірнісним алгоритмам кластеризації.

Робастний рекурсивний алгоритм можливісної нечіткої кластеризації для великих даних

У разі великої вибірки даних (BD) і послідовного надходження даних у систему (наприклад, часових рядів) можна використовувати рекурсивні алгоритми можливісної нечіткої кластеризації.

Для можливісних алгоритмів нечіткої кластеризації критерієм є вираз

$$E^R(w_{k,j}, c_j, \mu_k) = \sum_{k=1}^{N}\sum_{j=1}^{m} w_{k,j}^{\beta} D^2(x_k, c_j) + \sum_{j=1}^{m}\mu_i \sum_{k=1}^{N}(1 - w_{k,j})^{\beta}. \tag{6.58}$$

Мінімізація (6.58) за параметрами, $w_{k,j}$, c_j та μ_i приводить до системи рівнянь

$$\begin{cases} \dfrac{\partial E^R(w_{k,j}, c_j, \mu_j)}{\partial w_{k,j}} = 0, \\ \dfrac{\partial E^R(w_{k,j}, c_j, \mu_j)}{\partial \lambda_k} = 0, \\ \nabla c_j E^R(w_{k,j}, c_j, \lambda_k) = 0. \end{cases} \tag{6.59}$$

Розв'язання перших двох рівнянь (6.59) дає відомий результат:

$$\begin{cases} w_{k,j}^{\text{pos}} = \left(1 + \left(\dfrac{D^R(x_k, c_j)}{\mu_j}\right)^{\frac{1}{\beta-1}}\right)^{-1}, \\ \mu_j = \dfrac{\sum_{k=1}^{N} w_{k,j}^{\beta} D^R(x_k, c_j)}{\sum_{k=1}^{N} w_{k,j}^{\beta}}. \end{cases} \tag{6.60}$$

а третій

$$\nabla c_j E^R(w_{k,j}, c_j, \mu_j) = \sum_{k=1}^{N} w_{k,j}^{\beta} \nabla c_j D^R(x_k, c_j) = 0 \tag{6.61}$$

повністю відповідає (6.55).

Вводимо локальну модифікацію (6.58)

$$E_k^R = \sum_{j=1}^{m} w_{k,j}^{\beta} D^R(x_k, c_j) + \sum_{j=1}^{m} \mu_i (1 - w_{k,j})^{\beta} =$$

$$= \sum_{j=1}^{m} w_{k,j}^{\beta} \sum_{i=1}^{n} \beta_i \ln \cosh \frac{x_{k,i} - c_{j,i}}{\beta_i} + \sum_{j=1}^{m} \mu_i (1 - w_{k,j})^{\beta} \tag{6.62}$$

і, оптимізуючи її, отримуємо:

$$\begin{cases} w_{k,j}^{\text{pos}} = \left(1 + \left(\dfrac{D^R(x_k, c_j)}{\mu_j}\right)^{\frac{1}{\beta - 1}}\right), \\ c_{k+1,j,i} = c_{k,j,i} - \eta_k \dfrac{\partial E_k^R(w_{k,j}, c_j, \mu_j)}{\partial c_{j,i}} = c_{k,j,i} + \eta_k w_{k,j}^{\beta} \tanh \dfrac{x_k - c_{k,j,i}}{\beta_i}, \end{cases} \tag{6.63}$$

де параметр відстані μ_{kj} можна визначити за другим рівнянням системи (6.60) для k спостережень, а не для всього обсягу вибірки N.

Слід зазначити, що останні рівняння систем (6.53) та (6.59) є ідентичними і визначаються лише вибором метрики. Це дає можливість використовувати будь-яку метрику, придатну для конкретного випадку, що визначатиме лише процедуру налаштування прототипів, якщо рівняння для обчислення ваг залишиться незмінним.

Розглянуті робастні рекурсивні методи можна використовувати як у пакетному режимі, так і в режимі онлайн. В останньому випадку кількість спостережень k становить дискретний час.

Експериментальні дослідження. Експерименти зі сховищем даних, спотворених аномальними викидами, показали високу ефективність запропонованих алгоритмів у разі оброблення інформації, поданої у вигляді таблиць «об'єкт — властивість» [6.12, 6.13] та у вигляді часових рядів [6.12].

Зокрема, розглянуто задачу класифікації даних спеціально штучно сформованої вибірки, що містить тривимірний кластер даних, спостереження якого позначено символами «о», «×» та «+» [6.13] (див. рис. 6.10). Точки в кожній кластерній вибірці розподілені відповідно до щільності розподілу Лапласа з «важкими хвостами»:

$$p(x_i) = \sigma(1 + (x_i - c)^2)^{-1}, \tag{6.64}$$

де σ і c — ширина і центр відповідно.

Вибірка містить 9 000 спостережень (по 3 000 у кожному кластері) і розділена на навчальну (7 200 випадків) і тестову (1 800 випадків) підвибірки [6.12, 6.13].

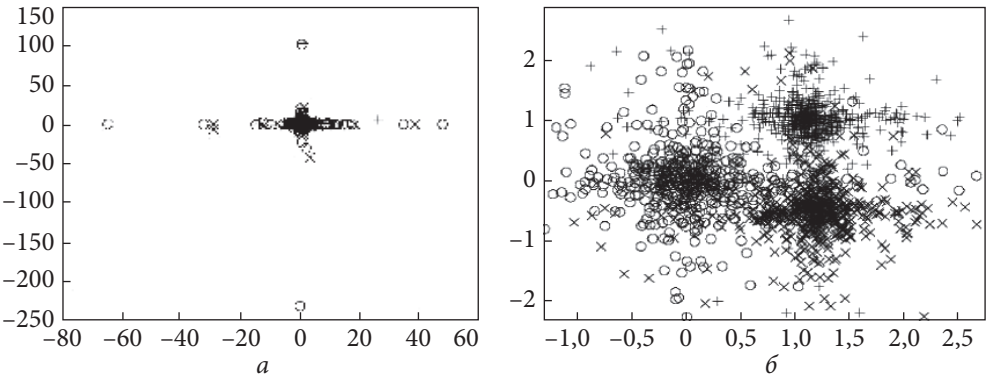


Рис. 6.10. Повний зразок (*а*) та його центральна частина (*б*)

Слід зазначити, що деякі спостереження перебувають дуже далеко від центрів кластерів (рис. 6.10, *а*). Прототипи кластерів розміщені в центральній ділянці даних, як показано на рис. 6.10, *б*. Для того щоб знайти правильні прототипи, алгоритм кластеризації має бути нечутливим до викидів.

Для всіх алгоритмів, що брали участь у порівнянні, процедура експерименту мала такий вигляд. На початку навчання вибірку було кластеризовано відповідними алгоритмами і знайдено прототипи кластерів. Потім навчальну та тестову вибірки було класифіковано відповідно до результатів кластеризації. Спостереження, що належать до кожного кластера, в процесі класифікації обчислюються відповідно до рівнянь (6.19), (6.57) або (6.63) залежно від типу алгоритму кластеризації. Кластер, до якого належить спостереження з максимальним ступенем належності, визначає клас цього спостереження. Класифікація і навчання виконуються в режимі онлайн-надходження спостережень, де $\beta = 2$, $\beta_1 = \beta_2 = \beta_3 = 1$, $\eta(k) = 0{,}01$. Результати наведено в табл. 6.11.

Як можна побачити на рис. 6.11, центри кластерів (прототипів), отриманих за алгоритмом нечітких *k*-середніх Бездека, зміщені від візуальних центрів кластерів через наявність «важких хвостів» щільності розподілу даних, на відміну від робастних методів з цільовою функцією (6.57) та (6.63), в яких прототипи знаходяться більш точно, що підтверджується меншою похибкою класифікації (див. табл. 6.11).

Таблиця 6.11

Результати навчання і похибка класифікації

Алгоритм	Навчальна вибірка	Зразок для тестування
Нечіткі *k*-середні	17,1 % (1229 спостережень)	16,6 % (299 спостережень)
Робастний імовірнісний	15,6 % (1127 спостережень)	15,6 % (281 спостереження)
Робастний можливісний	15,2 % (1099 спостережень)	14,6 % (263 спостереження)

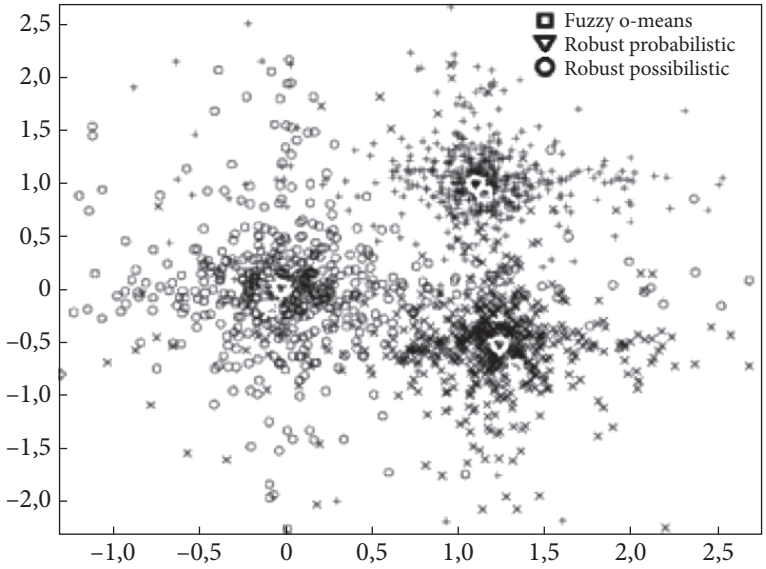


Рис. 6.11. Прототипи кластерів

Постійне зростання успішного застосування технологій обчислювального інтелекту в галузях аналізу даних підтверджує універсальність цього підходу. Водночас реальні проблеми, що виникають під час оброблення дуже великих баз даних (*Big Data*), ускладнюють використання сучасних алгоритмів та інструментів і потребують їх удосконалення для вирішення завдань інтелектуального аналізу даних у режимі реального часу з використанням парадигм ОІ та м'яких обчислень.

Висновки

Кластерний аналіз охоплює набір різних алгоритмів класифікації. Щоразу, коли необхідно класифікувати великий обсяг інформації на придатні для подальшого оброблення групи, кластерний аналіз виявляється дуже корисним і ефективним. Кластерний аналіз необхідний для класифікації інформації, за його допомогою можна у певний спосіб структурувати змінні і дізнатися, які змінні об'єднуються насамперед, а які слід розглядати окремо.

Важлива перевага кластерного аналізу полягає в тому, що він дає можливість виконувати розподіл об'єктів не за одним параметром, а за цілим набором ознак. Крім того, кластерний аналіз, на відміну від більшості математико-статистичних методів, не накладає жодних обмежень на вигляд об'єктів, що розглядаються, і дає змогу досліджувати множину вихідних даних практично довільної природи. Це має значення, наприклад, для прогнозування кон'юнктури, коли оперують показниками різних видів, що утруднює застосування традиційних економетричних підходів.

Як і будь-який інший метод, кластерний аналіз має певні недоліки та обмеження. Зокрема, склад і кількість кластерів залежить від обраних критеріїв розбиття. Під час зведення вихідного масиву даних до компактнішого вигляду можуть виникати певні спотворення, а також втрачатися індивідуальні ознаки окремих об'єктів унаслідок заміни їх характеристиками узагальнених значень параметрів кластера. У процесі класифікації об'єктів часто ігнорується можливість відсутності в аналізованої сукупності будь-яких значень кластерів.

Основним недоліком розглянутих методів нечіткої кластеризації *k*-середніх і Густавсона — Кесселя є те, що вони можуть бути використані лише у разі, якщо відома кількість кластерів. Проте зазвичай кількість кластерів невідома, а візуальні спостереження у багатовимірному випадку просто не призведуть до успіху. Для автоматичного визначення кількості кластерів можна використати показник Хі — Бені.

Як випливає з наведених результатів досліджень, метод Густавсона — Кесселя дає можливість шукати кластери як різноспрямовані еліпси. Іноді вибір методу попереднього розміщення центрів кластерів, а отже, і розташування попередніх центрів впливає на результати кластеризації і сприяє суттєвому їх поліпшенню.

Розглянуто також новий клас нечітких алгоритмів кластеризації: можливісні алгоритми (*possibilistic*) нечіткої кластеризації. В них немає обмеження щодо суми ступенів належності кожної точки даних до всіх кластерів. Показано зв'язки між імовірнісними та можливісними алгоритмами нечіткої кластеризації. Розглянуто рекурентні можливісні алгоритми кластеризації, проаналізовано їхні властивості та зв'язки з імовірнісними алгоритмами. Доведено що рекурентні можливісні алгоритми мають важливу властивість, вони є робастними і малочутливими до різних шумів та екстремальних викидів у вибірках даних. Проведено експериментальні дослідження робастних рекурентних алгоритмів можливісної кластеризації та їх порівняльний аналіз з робастними алгоритмами ймовірнісної кластеризації.

СПИСОК ЛІТЕРАТУРИ

6.1. Everitt B. S., Landau S., Leese M., Stahl D. Cluster Analysis: A Survey. Springer, 2011. 346 p.

6.2. Witten I. H., Frank E., Hall M. A. Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann. 2011.3rd ed. 664 p.

6.3. Russell S., Norvig P. Artificial Intelligence: A Modern Approach. 4th ed. Pearson, 2020. 1136 p.

6.4. Klir G. J., Yuan B. Fuzzy Sets and Fuzzy Logic: Theory and Applications. Prentice Hall, 1995. 574 p.

6.5. Zgurovsky M., Zaychenko Yu. Fundamentals of computational intelligence. System approach. Springer, 2016. 275 p.

6.6. Mendel J. M. Uncertain Rule-Based Fuzzy Systems: Introduction and New Directions. Springer, 2017. 2nd ed. 684 p.

6.7. Yager R.R., Filev D.P. Approximate clustering via the mountain method. *IEEE Trans. on Syst., Man and Cybern*. 1994. Vol. 24. P. 1279—1284.

6.8. Krishnapuram R., Keller J. Fuzzy and possibilistic clustering methods for computer vision. *IEEE Trans. on Fuzzy Systems*. 1993. Vol. 1. P. 98—110.

6.9. Park D.C., Dagher I. Gradient based fuzzy C-means (GBFCM) algorithm. *Proc. IEEE Int. Conf. On Neural Networks*. 1984. P. 1626—1631.

6.10. Bodyanskiy Ye., Gorshkov Ye., Kokshenev I., Kolodyazhniy V. Robust recursive fuzzy clustering algorithms. *Proc. East West Fuzzy Colloquim 2005.* Zittau — Goerlitz: HS, 2005. P. 301—308.

6.11. Bodyanskiy Ye., Gorshkov Ye., Kokshenev I., Kolodyazhniy V. Outlier resistant recursive fuzzy clustering algorithm. Computational Intelligence: Theory and Applications. Ed. by B. Reusch. *Advances in Soft Computing.* Vol. 38. Berlin — Heidelberg: Springer-Verlag, 2006. P. 647—652.

6.12. Bodyanskiy Ye. Computational intelligence techniques for data analysis. *Lecture Notes in Informatics*. Vol. P-72. Bonn: GI, 2005. P. 15—36.

6.13. Bodyanskiy Ye., Gorshkov Ye., Kokshenev I., Kolodyazhniy V., Shilo O. Robust recursive fuzzy clustering-based segmentation of biomedical time series. *Proc. 2006 Int. Symp. on Evolving Fuzzy Systems*. Lancaster (UK), 2006. P. 101—105.

РОЗДІЛ 7

ГЕНЕТИЧНІ АЛГОРИТМИ ТА ЕВОЛЮЦІЙНЕ МОДЕЛЮВАННЯ

7.1. Генетичні алгоритми

7.2. Застосування генетичного алгоритму для задачі структурного синтезу мереж

7.3. Еволюційне програмування

7.4. Диференціальна еволюція

7.5. Самонавчання ДЕ

7.6. Знаходження найкоротшого шляху на графі на основі ГА

Висновки

Список літератури

7.1. Генетичні алгоритми

Генетичні алгоритми (ГА) є, мабуть, першими алгоритмічними моделями, розробленими для моделювання генетичних систем. Вперше запропоновані А. Фрезером (Alexander S. Fraser) [7.9] і пізніше А. Бремерманом (Alan T. Bremermann) та Дж. Рівом (George Reeve), вони стали популярними завдяки вичерпній роботі Джона Холланда (John H. Holland) [7.3], якого вважають батьком ГА. Генетичні алгоритми моделюють еволюцію, у якій характеристики індивідів виражені у вигляді генів. Основні рушійні оператори ГА — це рекомбінація (моделює репродукцію), селекція (модель виживання найкращих) та мутація. Розглянемо загальний опис генетичного алгоритму.

На попередньому етапі потрібно створити та ініціалізувати початкову популяцію $C(0)$, вибрати умову зупинки, покласти лічильник поколінь $t = 0$. На t-й ітерації виконати такі кроки.

1. Оцінити фітнес-індекс $f(X_i(t))$ кожного індивіда $X_i(t)$.
2. Виконати репродукцію, щоб створити нащадка.
3. Вибрати нову популяцію $C(t + 1)$.
4. Перевірити умови зупинки. Якщо виконується, то зупинитися, інакше $t = t + 1$, і перейти на наступну ітерацію.

Канонічний генетичний алгоритм. Канонічний ГА, запропонований Дж. Холландом [7.3], відповідає загальному генетичному алгоритму з такими специфічними особливостями:

- використовується зображення індивідів у вигляді рядка;
- застосовується пропорційна селекція для вибору батьків для кросовера;
- одноточковий кросовер використовується як первинний метод для генерування нащадків;
- поступово розподілені мутації застосовуються як фоновий оператор малої важливості.

Слід зазначити, що спочатку мутації не вважали важливим оператором для застосувань ГА. І тільки в наступних реалізаціях було з'ясовано і використано дослідницьку силу мутації як потужного засобу забезпечення необхідної різноманітності; мутації було використано для поліпшення пошукових можливостей ГА.

За останні роки розроблено чимало варіацій генетичних алгоритмів, що різняться схемою зображення, операторами селекції та кросовера, а також оператором мутацій.

Розглянемо основні підходи до реалізації ГА, що ілюструють їхні переваги.

Кросовер. Оператори кросовера можна розподілити на три основні категорії:

• асексуальні, в яких нащадок генерується від одного з батьків;

• сексуальні, або бісексуальні, в яких використовують двох батьків для генерування одного та двох нащадків;

• мультисексуальні, в яких більш ніж двох батьків використовують для генерування одного або двох нащадків.

Операторів кросовера далі можна розподілити за схемою подання, що використовується. Наприклад, бінарні оператори, розроблені для подання у вигляді бінарних рядків, та оператори, які використовують зображення з рухомою комою (плаваючою точкою).

Батьків обирають за однією схемою селекції. При цьому немає жодної гарантії, що обрані батьки будуть відповідними, тобто задовольняти певний критерій. Під час селекції батьків слід враховувати такі обставини: внаслідок ймовірнісної селекції батьків може статися те, що одні й ті самі індивіди будуть обрані для обох батьків у разі кросовера, і тоді згенерований нащадок буде точною копією батька. Процес вибору (селекції) батьків має містити тест, який перешкоджає таким небажаним діям.

Можливо також, що той самий індивід братиме участь у більш ніж одному застосуванні операції кросовера. Тоді виникає проблема у разі використання схеми пропорційної селекції.

Якщо використовується один нащадок, він може замінити «найгіршого» батька. Така заміна може бути заснована на припущенні, що нащадок має бути кращим, ніж «найгірший» батько. Оператори кросовера можуть бути реалізовані тоді, коли нащадок замінить найгіршого індивіда з усієї популяції. У разі двох нащадків можна використовувати аналогічну стратегію.

З моменту появи канонічного ГА було розроблено кілька варіацій генетичних алгоритмів, що відрізняються схемою зображення, операторами селекції, кросовера та мутації. Деякі нові застосування використовують інші концепції з природи, а саме — масові острови популяції. Наведемо опис найважливіших з них, які демонструють багатство ідей генетичних алгоритмів.

Бінарні подання

Більшість операторів кросовера є сексуальними, які застосовують до двох батьків. Якщо $x_1(t)$ і $x_2(t)$ позначають двох обраних батьків, процес рекомбінації викладено в алгоритмі 7.1. У цьому алгоритмі $m(t)$ становить

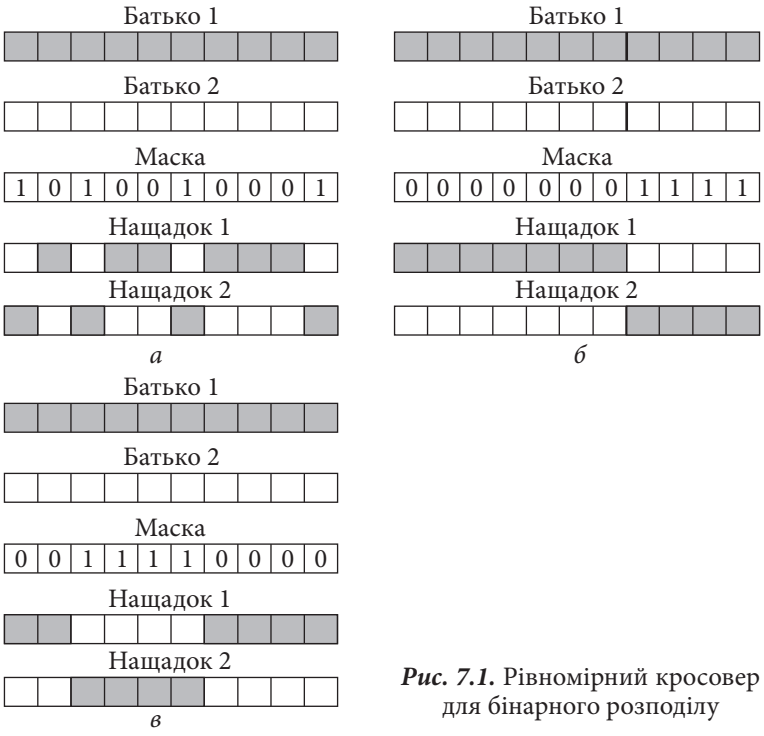


Рис. 7.1. Рівномірний кросовер для бінарного розподілу

маску, що визначає, які біти у батьків мають зазнати свопінгу (тобто обміну, щоб згенерувати двох нащадків). Було розроблено кілька операторів кросовера для обчислення маски [7.1].

1. Одноточковий кросовер (рис. 7.1, *б*). Дж. Холланд запропонував, щоб за кросовера свопінг (обмін генами) охоплював сегменти генів, а не окремі гени.

2. Двоточковий кросовер, за якого випадково вибирають позиції двох бітів і бітові рядки між цими точками обмінюються, як показано на рис. 7.1, *в* (див. алгоритм 7.3). Маски підраховуються згідно з алгоритмом 7.2. Цей оператор може бути узагальнено на *n*-точковий кросовер.

3. Рівномірний кросовер. Випадково створюється маска розмірності n_x, як викладено в алгоритмі 7.3. Тут p_x — ймовірність біт-свопінгу. Якщо $p_x = 0{,}5$, то кожен біт має однакові шанси зазнати свопінгу. Рівномірний кросовер для бінарного розподілу показано на рис. 7.1, *а*.

Алгоритм 7.1. Загальний алгоритм кросовера бітових рядків [7.1].

1. Покласти $\tilde{x}_1(t) = x_1(t)$; $\tilde{x}_2(t) = x_2(t)$.

2. Якщо $U(0, 1) \le p_c$ (де $U(0, 1)$ — рівномірний розподіл на інтервалі $(0, 1)$, то обчислити бінарну маску $m(t)$).

3. $j = 1$. Якщо $m_j = 1$, то обмін бітами (свопінг); $\tilde{x}_{1j}(t) = x_{2j}(t)$ та $\tilde{x}_{2j}(t) = x_{1j}(t)$.

4. $j = 1 + 1$. Якщо $j < n_x$, то крок 2. Інакше — крок 4.

5. Зупинка.

Алгоритм 7.2. Формування маски одноточкового кросовера.

1. Вибрати точку кросовера $k \in U(1, n_x - 1)$, де $U(1, n_x - 1)$ — рівномірний розподіл на ряді цілих чисел, $\{1, n_x - 1\}$.
2. Ініціалізувати маску: $m_j(t) = 0$ для всіх $j = 1, 2, \ldots, n_x$.
3. Покласти $m_j(t) = 1$ для всіх $j = k + 1, \ldots, n_x$.
4. Кінець.

Алгоритм 7.3. Двоточковий кросовер.

1. Вибрати дві точки кросовера $K_1, K_2 \in U(1, n_x)$.
2. Ініціалізувати маску: $m_j(t) = 0$ для всіх $j = 1, 2, \ldots, n_x$.
3. Для всіх $j \in \{K_1 + 1, K_2\}$ покласти $m_j(t) = \begin{cases} 1, K_1 + 1 \le j \le K_2; \\ 0 \text{ — в інших випадках.} \end{cases}$
4. Кінець.

А. Бремерман запропонував перші варіанти мультибатьківського кросовера для бінарного подання. За заданих n_μ батьківських векторах $x_1(t), \ldots, x_{n_\mu}(t)$ за збігом здебільшого генерується один нащадок згідно з формулою

$$\tilde{x}_{l_j}(t) = \begin{cases} 0, \text{ якщо } n'_\mu > n_\mu / 2, \quad l = 1, ..., n_x; \\ 1 \text{ — інакше,} \end{cases} \tag{7.1}$$

де n'_μ — кількість батьків, у яких $x_{l_j}(t) = 0$.

Мультибатьківська версія n-точкового кросовера була запропонована також Бремерманом, в якій $n_\mu - 1$ ідентичних точок кросовера вибираються у батьків. Генерується один нащадок за допомогою вибору по одному сегменту кожного з батьків.

Подання з плаваючою точкою

Оператори кросовера, розглянуті вище, можна застосовувати й для зображення з плаваючою точкою (в іншій термінології — рухомою комою. — Авт.) як стратегії дискретної рекомбінації. На противагу цим дискретним операторам, у яких відбувається свопінг інформації (генів) між батьками, проміжні оператори рекомбінації, розроблені спеціально для подання з плаваючою точкою, формують компоненти через обраних батьків.

Один із перших операторів кросовера з плаваючою точкою — лінійний оператор, запропонований А. Райтом (Alberto Wright). З батьків $x_1(t)$ і $x_2(t)$ генеруються три кандидати в нащадки: $x_1(t) + x_2(t)$; $(1{,}5x_1(t) - 0{,}5x_2(t))$; $(-0{,}5x_1(t) + 1{,}5x_2(t))$. Два найкращі рішення відбираються як нащадки. Райт запропонував також спрямований евристичний оператор кросовера, в якому один нащадок створюється від двох батьків відповідно до співвідношення

$$\tilde{x}_{ij}(t) = U(0, 1)(x_{2j}(t) - x_{1j}(t)) + x_{2j}(t) \tag{7.2}$$

за умови (обмеження), що батько $x_2(t)$ не може бути гіршим ніж $x_1(t)$.

З. Михалевич (Zbigniew Michalewicz) [7.24] запропонував арифметичний кросовер, що становить мультибатьківську стратегію рекомбінації, в якій обчислюються зважене середнє для двох або більше батьків. Генерується один нащадок згідно з виразом

$$\tilde{x}_{ij}(t) = \sum_{l=1}^{n_\mu} \gamma_{lj}\, x_{lj}(t), \tag{7.3}$$

де $\sum_{l=1}^{n_\mu} \gamma_{lj} = 1$.

Окремим випадком є оператор кросовера для $n_\mu = 2$, для якого

$$\tilde{x}_{ij}(t) = (1 - \gamma)x_{1j}(t) + \gamma x_{2j}(t), \tag{7.4}$$

де $\gamma \in (0; 1)$.

Дж. Ешельман (Lawrence J. Eshelman) і Дж. Д. Шаффер (James D. Schaffer) [7.30] розробили варіант зваженого середнього, що задається рівнянням (7.4), названий бленд-кросовером (*BLX*-α), в якому

$$\tilde{x}_{ij}(t) = (1 - \gamma)x_{1j}(t) + \gamma_j x_{2j}(t), \tag{7.5}$$

де $\gamma_j = (1 + 2\alpha)U(0; 1) - \alpha$.

Оператор *BLX*-α випадково добирає для кожної компоненти випадкове значення в інтервалі

$$[x_{1j}(t) - \alpha(x_{2j}(t) - x_{1j}(t)),\ x_{2j}(t) + \alpha(x_{2j}(t) - x_{1j}(t))]. \tag{7.6}$$

При цьому передбачається, що $x_{1j}(t) < x_{2j}(t)$.

Встановлено також, що випадок $\alpha = 0{,}5$ добре працює. З. Михалевич зі співавторами [7.29] розробили геометричний кросовер для двох батьків:

$$\tilde{x}_{1j}(t) = (x_{1j}(t)\ x_{2j}(t))^{0,5}. \tag{7.7}$$

Геометричний кросовер може бути узагальнений для випадку багатьох батьків відповідно до виразу

$$\tilde{x}_{ij}(t) = (x_{ij}^{\alpha_1})\, x_{2j}^{\alpha_2}, ..., x_{m_{ij}}^{\alpha_n}),$$

де $\sum_{l=1}^{n_\mu} \alpha_l = 1$, $\alpha_l > 0$.

К. Деб (Kalyanmoy Deb) і Р. Агравал (Rohit B. Agrawal) [7.31] розробили бінарний кросовер для моделювання поведінки одноточкового оператора кросовера для бінарного подання. Двоє батьків $x_1(t)$ і $x_2(t)$ використовуються для генерації двох нащадків відповідно до рівнянь

$$\tilde{x}_{1j}(t) = 0{,}5[(1 + \gamma_j)x_{1j}(t) + (1 - \gamma_j)x_{2j}(t)],$$
$$\tilde{x}_{2j}(t) = 0{,}5[(1 - \gamma_j)x_{1j}(t) + (1 + \gamma_j)x_{2j}(t)],$$

де $\gamma_j = \begin{cases} (2r_j)^{\frac{1}{\eta + 1}}, \text{ якщо } r_j \le 0{,}5; \\ \left(\dfrac{1}{2(1 - r_j)}\right)^{\frac{1}{\eta + 1}} \text{ — інакше,} \end{cases}$

при цьому $r_j \in U(0, 1)$, а $\eta > 0$ — індекс розподілу. К. Деб та Р. Агравал запропонували взяти $\eta = 1$.

Розглянемо тепер оператори мультибатьківського кросовера. Головна мета використання таких операторів — покращити пошукові можливості операторів мультибатьківського кросовера порівняно з операторами сексуального кросовера (з двома батьками).

М. Оно (Munenori Ono) та Ш. Кабаяші (Shigeru Kobayashi) [7.15] розробили унімодально розподілений оператор *UNDX*, у якому два або більше нащадків генеруються з використанням трьох батьків. Нащадки створюються на основі еліпсоїдного розподілу ймовірностей, де одна з осей формується вздовж лінії, що з'єднує двох батьків. Екстент (розширення) ортогонального спрямування визначається на перпендикулярі, проведеному від третього батька на цю вісь. *UNDX*-оператор може бути узагальнений для будь-якої кількості батьків при $3 \le n_\mu \le n_s$.

Для узагальнення $n_\mu - 1$ батьків вибираються випадково і підраховується їхній центр мас $\bar{x}(t)$, де

$$x_{ij}(t) = \sum_{l=1}^{n_\mu - 1} x_{lj}(t). \tag{7.10}$$

За середнім значенням обчислюється $l = 1, \ldots, n_\mu - 1$ векторів напрямку $d_l(t) = x_l(t) - \bar{x}(t)$. Використовуючи вектори напрямку, обчислюють напрямні косинуси як $e_l(t) = \dfrac{d_l(t)}{||d_l(t)||}$, (де $||d_l(t)||$ — довжина вектора $d_l(t)$). Вибирають випадкового батька з індексом n_μ.

Нехай $x_{n_\mu} - \bar{x}(t)$ — вектор, ортогональний до всіх $e_l(t)$ і $\delta = ||x_{n_\mu} - \bar{x}(t)||$. Нехай $e_l(t)$, $l = n_\mu, \ldots, n_s$ — ортогональний базис підпростору, який ортогональний до підпростору, породженого напрямними косинусами $e_l(t)$, $l = \overline{1, n_\mu - 1}$. Далі генеруються нащадки за виразом

$$\tilde{x}_i(t) = \tilde{x}(t) + \sum_{l=1}^{n_\mu - 1} N(0, \sigma_1^2)||d_l||e_l + \sum_{l=n_\mu}^{n_s} N(0, \sigma_2^2)\, \delta e_l(t), \tag{7.11}$$

де $\sigma_1 = \dfrac{1}{\sqrt{n_\mu - 2}}$, а $\sigma_2 = \dfrac{0{,}35}{\sqrt{n_s - n_\mu - 2}}$.

Використовуючи рівняння (7.11), можна згенерувати будь-яке число нащадків, що розміщуються навколо центра мас обраних батьків. Більша ймовірність приписується створенню нащадка біля центра мас, а не біля батьків.

Дія операторів кросовера на основі центра мас для кількох батьків (точки зображують потенційних нащадків) показано на рис. 7.2.

С. Цуцуї (Satoshi Tsutsui) та Д. Гольдберг (David E. Goldberg) [7.25] запропонували оператор симплекс-кросовера (*SPX*) як інший підхід з урахуванням центра мас. Вибирається $n_\mu = n_x + 1$ батьків незалежно один від одного для n_x-розмірного простору пошуку. Ці n_μ батьків формують симплекс. Симплекс розширюється в кожному з n_μ напрямків, як показано на

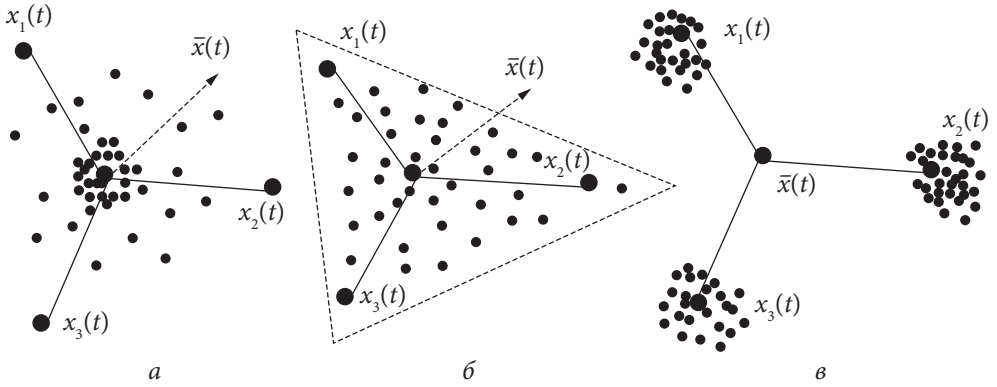


Рис. 7.2. Дія операторів кросовера для операторів з плаваючою точкою: *а* — оператор *UNDX* для $n_\mu = 4$; *б* — оператор *SPX*; *в* — оператор *DCX*

рис. 7.2, *б*. Для $n_x = 2$, $n_\mu = 3$ і $\tilde{x}(t) = \sum_{l=1}^{n_\mu} x_l(t)$ розширений симплекс визначається точками

$$(1 + \gamma)(x_l(t) - \tilde{x}(t)). \tag{7.12}$$

Для $l = \overline{1, n_\mu} = 3$ і $\gamma \geq 0$ нащадки генеруються поступово в розширеному симплексі.

К. Деб зі співавторами [7.31] запропонували варіацію оператора *UNDX*, який вони назвали «батькоцентрований кросовер» *DCX*. Замість генерації нащадків навколо центра мас вибраних батьків нащадки генеруються навколо вибраних батьків. Оператор *DCX* вибирає n_μ батьків і обчислює їхній центр мас $\tilde{x}(t)$. Кожен нащадок має бути згенерований одним із батьків, які рівноймовірно вибираються з n_μ батьків. Для кожного нащадка обчислюється вектор

$$d_i(t) = x_i(t) - \tilde{x}(t), \tag{7.13}$$

де $x_i(t)$ — рівноймовірно обраний батько. Від інших батьків визначаються відстані як перпендикуляри на вектор , що позначаються δ_l. Підраховується середнє з цих напрямків:

$$\bar{\delta} = \sum_{\substack{l=1 \\ l \neq i}}^{n_\mu} \frac{\delta_l}{n_\mu - 1}. \tag{7.14}$$

Далі нащадки генеруються згідно з виразом

$$\tilde{x}_i(t) = \bar{x}_i(t) + N(0, \sigma_1^2)\left\|d_i(t)\right\| + \sum_{\substack{l=1 \\ l \neq i}}^{n_\mu} N(0, \sigma_2^2)\bar{\delta}e_l(t), \tag{7.15}$$

де $x_i(t)$ — випадково обраний батько для нащадка $\tilde{x}_i(t)$; $\{e_l(t)\}$ — ортонормований базис розмірності $n_\mu - 1$, натягнутий на підпростір, перпендикулярний до $d_i(t)$. Дію оператора *DCX* показано на рис. 7.2, *в*.

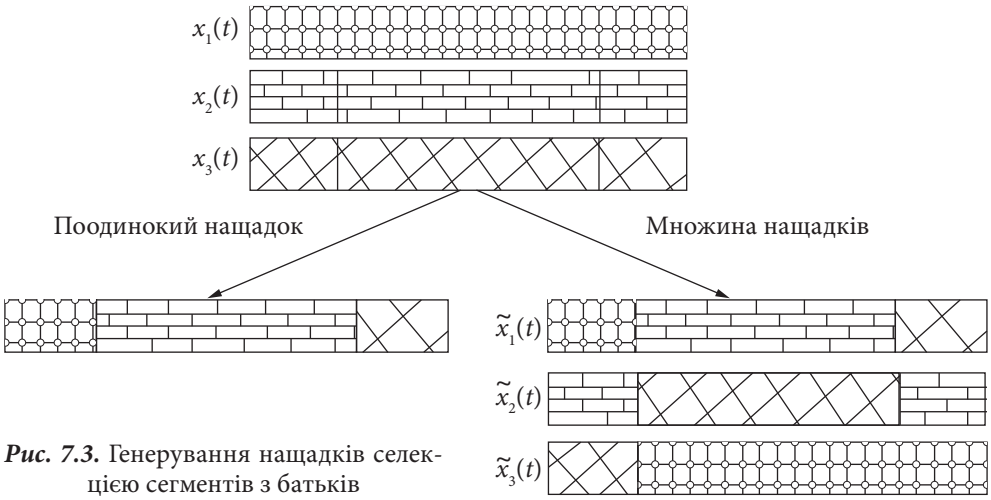


Рис. 7.3. Генерування нащадків селекцією сегментів з батьків

Діагональний кросовер. Діагональний кросовер є узагальненням n-точкового кросовера для більш ніж двох батьків: $n \geq 1$ точки кросовера вибираються і застосовуються до всіх $n_\mu = n + 1$ батьків. Один чи $n + 1$ нащадків можуть бути згенеровані селекцією сегментів із батьків, як показано на рис. 7.3, для $n = 2$; $n_\mu = 3$.

Мутації

Мета мутацій — ввести у створений індивід новий генетичний матеріал, тобто додати розмаїття генетичних характеристик популяції. Мутації використовують для підтримки кросовера, щоб повний діапазон алелів був доступним для кожного гена.

Мутація застосовується з певною ймовірністю P_m до кожного гена нащадка $\tilde{x}_i(t)$, щоб створити мутованого нащадка $x'(t)$. Імовірність мутації, яку також називають інтенсивністю мутацій, зазвичай вважають такою, що дорівнює малій величині $P_m \in [0; 1]$ так, щоб задовільні рішення були б не надто спотворені.

За умови заданої мутації кожного гена P_m імовірність того, що індивід буде мутований, задається за виразом [7.1, 7.2]

$$P_M\{\text{мутації } \tilde{x}_i(t)\} = 1 - (1 - P_m)^{n_x}, \tag{7.16}$$

де n_x — число генів індивіда; $P_\text{М}$ — імовірність мутації.

У припущенні бінарного подання, що $H(\tilde{x}_i(t); x_i'(t))$ — відстань (відповідно до результатів Дж. Холланда) між нащадком і його мутованої версією, імовірність того, що мутована версія зберігає подібність до оригіналу, має вигляд [7.3]

$$P\{x_i'(t) \approx \tilde{x}_i(t)\} = P_m^{H(\tilde{x}_i(t);x_i'(t))}(1 - P_m)^{nx - H(\tilde{x}_i(t);x_i'(t))}. \tag{7.17}$$

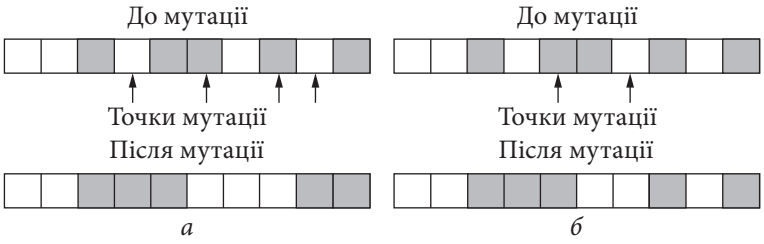


Рис. 7.4. Дія оператора мутацій

Для бінарного подання було розроблено такі оператори мутації: рівномірна мутація, де позиції бітів вибираються випадково, а відповідні біти замінюються на заперечення, як показано на рис. 7.4, *а*; упорядкована мутація (*inorder mutation*), де дві точки мутації вибираються випадково і лише біти між цими точками зазнають випадкової мутації (рис. 7.4, *б*).

Гауссова мутація. Для бінарного подання змінних з плаваючою точкою (рухомою комою) запропоновано, що бітовий рядок, який є змінною рішення, конвертується на значення з плаваючою точкою і мутується з гауссовим шумом. Для кожної хромосоми генеруються випадкові числа за розподілом Пуассона (Сімеон-Дені Пуассон) для визначення генів, які мають бути мутовані. Бітові рядки, що зображують ці гени, потім конвертуються на своє заперечення. До кожного значення з плаваючою точкою додається крок розміром $N(0; \sigma_j)$ (де $\sigma_j = 0{,}1$ від інтервалу j змінної рішення). Далі мутоване значення з рухомою комою конвертується назад у бітовий рядок. Р. Хінтердинг (Robert Hinterding) показав, що гауссова мутація з поданням з рухомою комою забезпечує кращі результати, ніж біт-фліпінг. Для бітових рядків високої розмірності випадкова мутація (див. рис. 7.4, *а*) може потребувати значних обчислювальних витрат. Для того щоб зменшити обчислювальну складність, було запропоновано [7.38] розбити бітовий рядок кожного індивіда на множину бункерів, і якщо бункер має бути мутований, то випадково вибирають один з його бітів, який далі мутується.

Як стверджують Р. Хінтердинг (Ralph Hinterding), З. Михалевич (Zbigniew Michalewicz) [7.7], для подання з рухомою комою функціонування відбувається краще, коли змінні рішення становлять безпосередньо значення з плаваючою точкою і для таких подань застосовуються відповідні оператори мутацій замість того, щоб конвертувати їх у бінарне подання.

Оператор макромутацій. Т. Джонс (Terry Jones) [7.33] запропонував оператор макромутацій, який було названо оператором «безголових курчат». Цей оператор створює нащадка рекомбінацією батьківського і випадково згенерованого індивідів, використовуючи будь-який із раніше розглянутих операторів кросовера. Такий оператор можна трактувати як мутацію завдяки введенню нового, випадково згенерованого генетичного матеріалу.

Параметри керування

Крім розміру популяції на показники якості генетичного алгоритму впливають також інтенсивності мутацій P_m та кросовера P_c. У перших роботах з ГА застосовували дуже низькі значення для P_m і відносно високі для P_c. Зазвичай величини P_m і P_c вибирають статистичні. Однак широко відомо, що ці параметри істотно впливають на характеристики ГА і вибір оптимальних значень для P_m і P_c може значно покращити характеристики (показники якості). Отримання цих оптимальних характеристик через емпіричний параметр — процес, що потребує тривалого часу.

Вирішенням проблеми знаходження оптимальних значень керівних параметрів є використання параметрів, що динамічно налаштовуються.

Динамічні генетичні алгоритми стали продовженням розвитку ідеї поколінських алгоритмів, які у процесі роботи змінюють розмір популяції. На відміну від генетичних алгоритмів з фіксованими параметрами динамічні генетичні алгоритми здатні змінювати і підлаштовувати свої параметри, такі як розмір популяції, ймовірність їх застосування і навіть генотип алгоритму, в процесі роботи. Безумовно, зручний вибір параметрів роботи генетичного алгоритму забезпечить його високу продуктивність. У найпростішому випадку вибір певного параметра його зміни може здійснюватися з множини зумовлених варіантів.

Виділимо такі основні стратегічні параметри, які можуть змінюватися у процесі роботи генетичного алгоритму:

- оператор мутації та ймовірність мутації;
- оператор кросовера та ймовірність кросовера;
- механізм селекції та її тиск (*selective pressure*);
- обсяг популяції рішень.

Дослідник має можливість визначити оптимальні значення параметрів алгоритму в ході попередніх досліджень, але оскільки процес генетичного пошуку — це, по суті, динамічний і адаптивний процес, то є сенс припустити, що оптимальні значення параметрів алгоритму можуть змінюватися в процесі роботи. Отже, відповідальність за вибір оптимальних параметрів функціонування можна перекласти на сам алгоритм, дозволивши йому динамічно підтримувати рівновагу параметрів під час роботи. Часто для підвищення ефективності застосування генетичного алгоритму використовують динамічні мутації та кросовер, і навіть змінюють розмір популяції рішень.

Є два основних підходи до визначення параметрів генетичного алгоритму (рис. 7.5): регулювання параметрів (*parameter tuning*) та керування параметрами (*parameter control*) [7.1, 7.2].

Регулювання параметрів полягає у запуску множини тестів з метою визначити, за яких значень параметрів генетичний алгоритм діє якнайкраще.

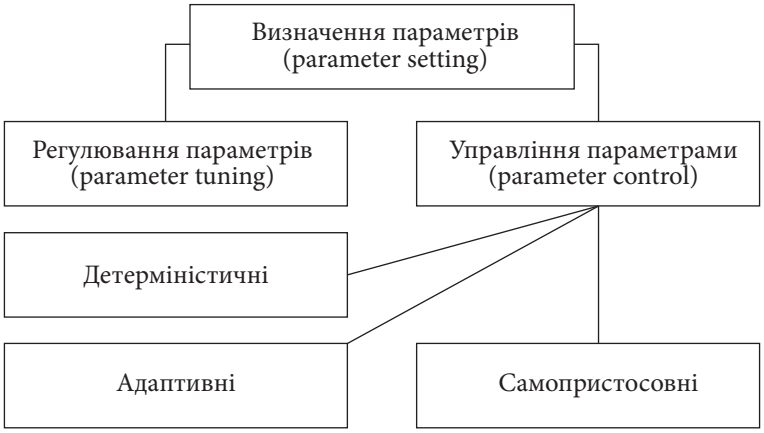


Рис. 7.5. Основні підходи до визначення параметрів генетичного алгоритму

Так, експериментально з'ясовують оптимальні значення ймовірностей мутації та кросовера, розмір популяції та інші параметри.

Де Йонг (De Jong) [7.34] здійснив масштабні дослідження (для низки тестових функцій) з метою виявлення оптимальних параметрів роботи генетичного алгоритму в традиційній постановці і рекомендував використовувати значення розміру популяції 50, ймовірності одноточкового кросовера 0,6 та ймовірності мутації 0,001. Однак, безумовно, ці параметри не є універсальними. Серед недоліків цього підходу також варто зазначити людський фактор у виборі значень параметрів, що може спричинити неоптимальну продуктивність. Традиційна процедура тестування та порівняння отриманих результатів перед запуском алгоритму потребує багато часу, крім того, неможливо врахувати аспект взаємодії параметрів алгоритму в процесі роботи. І, нарешті, під управлінням параметрами мається на увазі старт алгоритму з деякими початковими значеннями параметрів, можливо визначеними за допомогою першого методу. Далі під час роботи алгоритму ці параметри зазнають модифікацій різними способами.

Залежно від способу модифікації параметрів динамічні генетичні алгоритми поділяють на кілька категорій [7.22]:

1) детерміністичні (*deterministic*);

2) адаптивні (*adaptive*);

3) самопристосовні (*self-adaptive*).

У детерміністичних генетичних алгоритмах параметри змінюються під час роботи алгоритму за деяким заздалегідь заданим законом, який найчастіше залежить від часу (від кількості поколінь чи кількості обчислень цільової функції). Основною проблемою тут є знаходження вищезазначеного закону для максимальної продуктивності генетичного алгоритму в межах поставленого завдання.

Розглянемо, як саме концепцію детерміністичних генетичних алгоритмів можна застосувати на практиці. Наприклад, у генетичному алгоритмі з нефіксованим розміром популяції Я. Арабас (Jarosław Arabas), З. Міхалевич (Zbigniew Michalewicz), В. Мулавка (Witold Mulawka) [7.29] кожній особині приписують максимальний вік, тобто число поколінь, після яких особина гине. Використання алгоритму нового параметра — віку — дає можливість ввести оператор відбору в нову популяцію. Вік кожної особини індивідуальний і залежить від її пристосованості. У кожному поколінні t на етапі відтворення зазвичай створюється додаткова популяція з нащадків. Розмір додаткової популяції пропорційний розміру основної популяції A і дорівнює

$$\text{size}B(t) = \text{size}A(t) \cdot p,$$

де p — ймовірність відтворення.

Для відтворення особин вибирають з основної популяції з однаковою ймовірністю незалежно від їх пристосованості. Після застосування мутації та кросовера нащадкам приписують вік відповідно до значення їх пристосованості. Потім з основної популяції видаляють тих особин, термін життя яких минув, і додають нащадків з проміжної популяції. Отже, розмір популяції після однієї ітерації алгоритму обчислюється за такою формулою: $\text{size}A(t + 1) = \text{size}A(t) + \text{size}B(t) - D(t)$, де $D(t)$ — число особин, що помирають у поколінні t.

Для організації детерміністичної динамічної мутації Т. Бек (Thomas Bäck) та Ґ. Шутц (Günter Schutz) [7.2] використовували спадну функцію $p(t) = \left(2 + \frac{l-2}{T-1}t\right)^{-1}$ (де T — кількість ітерацій генетичного алгоритму, l — довжина хромосоми. Імовірність мутації, отримана за цією формулою, обмежена в інтервалі (0, 1/2)). Один із перших динамічно змінюваних параметрів інтенсивності мутацій увів Т. Фогарті (Timothy C. Fogarty) [7.2], який запропонував схему, де інтенсивність мутацій експоненційно зменшується з номером покоління t:

$$P_m(t) = \frac{1}{240} + \frac{0{,}11375}{2^t}. \tag{7.18}$$

Як альтернативу Фогарті запропонував також для бінарного подання ввести інтенсивність мутацій на 1 біт:

$$P_m(j) = \frac{0{,}3528}{2^{j-1}},\ j = 1, 2, ..., n_b, \tag{7.19}$$

де n_b вказує на найменший значущий біт.

Ці дві схеми було скомбіновано і отримано таку залежність для P_m:

$$P_m(j,t) = \frac{0{,}4026}{2^{t+j-1}} + \frac{28}{1905 \cdot 2^{j-1}}. \tag{7.20}$$

Різні схеми можна використати для зменшення інтенсивності мутацій. Розглянута схема є експоненціальною. Альтернативою може виступати лінійна схема, в якій забезпечуються повільніші зміни P_m, що дають можливість виконати більше число експериментів. Проте повільне зменшення інтенсивності мутацій може бути руйнівним для вже знайдених «хороших» рішень.

Практично всі подібні формули містять умову $1/t$, де t — номер покоління. У цьому разі початкова ймовірність мутації є досить високою і зменшується в міру роботи алгоритму. Отже, на ранніх етапах роботи генетичного алгоритму мутації відіграють ключову роль у визначенні найперспективніших галузей пошуку, проте згодом для їх неквапливого дослідження використовується кросовер. Такі формули не лише усувають необхідність задавати параметри, а й, за оцінками дослідників, значно підвищують ефективність генетичного алгоритму.

Зауважимо, що можливість мутації в роботі [7.38] змінюється детерміновано в часі і залежить від поточних результатів пошуку. Питання оптимальності такого підходу є спірним. Як альтернативу можна використовувати поточні результати пошуку адаптації параметрів генетичного алгоритму.

В адаптивних генетичних алгоритмах інформація про роботу алгоритму у вигляді зворотного зв'язку впливає на значення його параметрів. Однак на відміну від алгоритмів, що самопристосовуються, значення цих параметрів однакові для всіх особин популяції. І оскільки суть механізму зворотного зв'язку визначається дослідником, то питання, як зробити його оптимальним, залишається відкритим.

Такий спосіб управління алгоритмом використав І. Рехенберг (Ingo Rechenberg) [7.2], який постулював, що з п'яти мутацій одна має вести до появи кращого індивіда. Далі він пов'язав можливість мутації з частотою непоганих мутацій. Тож, якщо частота хороших мутацій вища за 1/5, то ймовірність мутації збільшується, і навпаки.

У формалізованому вигляді правило Рехенберга має вигляд:

$$\begin{aligned} &\text{якщо } \phi(k) < 1/5 \rightarrow (p_m/\lambda);\\ &\text{якщо } \phi(k) = 1/5 \rightarrow (p_m);\\ &\text{якщо } \phi(k) > 1/5 \rightarrow (\lambda p_m), \end{aligned} \qquad (7.21)$$

де $\phi(k)$ — відсоток успішних мутацій за k ітерацій, λ — навчальний коефіцієнт (зазвичай $\lambda = 1{,}1$).

Для підвищення ефективності застосування генетичного алгоритму Теренс запропонував використовувати дві адаптивні схеми зміни мутації.

1. *Схема підсилення* (*Constant Gain scheme*), яка має такий вигляд [7.2]:

• індивід (x, p_m) мутує трьома різними способами — $M(x, p_m/\omega) \rightarrow (x_1)$, $M(x, p_m) \rightarrow (x_2)$, та $M(x, \omega p_m) \rightarrow (x_3)$ (де ω — коефіцієнт дослідження);

• вибір найбільш пристосованого індивіда та нової ймовірності мутації — $MAX\{(x, p_m), (x_1, p_m/\lambda), (x_2, p_m), (x_3, \lambda p_m)\}$.

У наведених виразах λ — навчальний коефіцієнт, ω — коефіцієнт дослідження. Зазвичай λ і ω мають різні значення (1 < λ < ω). Щоб уникнути коливань у процесі навчання, λ обмежується як ω > λ. Типовими значеннями коефіцієнтів є λ = 1,1 та ω = 1,5;

2. *Схема ослаблення* (*Declining scheme*) сприяє використанню агресивнішого розміру кроку, водночас пригнічуючи випадкові коливання, які можуть з'явитися за високого коефіцієнта навчання λ, і має такий вигляд:

• індивід (x, p_m) мутує трьома різними способами — $M(x, p_m/\omega) \to (x_1)$, $M(x, p_m) \to (x_2)$, $M(x, \omega p_m) \to (x_3)$;

• зменшення ймовірності мутації батька — $(x, p_m) \to (x, \gamma p_m)$;

• або вибір найбільш пристосованого індивіда та нової ймовірності мутації — $MAX\{(x, \gamma p_m), (x_1, \lambda p_m), (x_2, \gamma p_m), (x_3, \lambda p_m)\}$.

Відмінність між наведеними схемами полягає в тому, що друга схема не має здатності збільшувати ймовірність мутації, тому поточна ймовірність мутації зменшуватиметься доти, доки не буде отримано чергового успішного нащадка. Використані коефіцієнти перебувають у таких межах: λ > 1, ω > 1 і 0,9 < γ < 1. Типовими значеннями коефіцієнтів є λ = 2,0, ω = 2,0 і γ = 0,95.

Для адаптації ймовірностей кросовера та мутації Юнші Ші (Yunshi Shi) застосував набір нечітких логічних правил, що стосуються ймовірності мутації [7.2]:

Набір нечітких адаптивних правил ШІ

1. Якщо BF — низьке або середнє, то MP — низька.
2. Якщо BF — середнє і UN — низька, то MP — низька.
3. Якщо BF — середнє і UN — середня, то MP — середня.
4. Якщо BF — високе і UN — низька, то MP — низька.
5. Якщо BF — високе і UN — середня, то MP — середня.
6. Якщо UN — висока, то MP вибирається випадково (висока, середня, низька).

У цих виразах BF позначає найкраще значення пристосованості, UN — кількість ітерацій з моменту останньої зміни BF, MP — імовірність мутації. Цей набір правил, безумовно, передбачає задання виду функцій нечіткої належності до різних метрик і ймовірностей мутації. Наприклад, може бути використано нечіткий логічний висновок М. Сугено. З метою організації динамічної зміни оператора кросовера чи мутації різні можливості, задані константами чи лінійними функціями, вибирають кожному за нечіткого логічного висновку.

В оригінальній постановці для набору правил ШІ як метрику використовували дисперсію пристосованості (*Variance of Fitness*) з високим, середнім та низьким ступенями нечіткої належності.

Для оптимізаційних задач з обмеженнями часто використовують штрафні функції. Цю ідею можна застосувати для організації адаптивного кросо-

вера та мутації. Так, для вирішення завдань фарбування графів А. Ейбен (Albert E. Eiben) і Й.К. ван дер Хаув (Joann K. van der Hauw) запропонували використовувати оцінну функцію з покроковою адаптацією ваг (stepwise adaptive weights) — у разі, якщо певне обмеження порушено, розмір штрафу для відповідної змінної збільшується: eval(x) = $f(x)$ + Wpenalty(x), де $f(x)$ — цільова функція, penalty(x) — штрафна функція, а W — ваговий коефіцієнт, який визначає, наскільки вагомий штраф буде призначено за порушення обмеження. Якщо всі обмеження задоволені, то penalty(x) = 0, інакше penalty(x) > 0. У загальному вигляді penalty(x) = $\Sigma_i \begin{cases} 1, \\ 0 \end{cases}$ (1 — у разі, якщо i-те обмеження порушено, 0 — якщо задоволено).

У самопристосовних генетичних алгоритмах завдяки зворотному зв'язку використовується інформація про роботу алгоритму, але кожна особина популяції має власні значення параметрів. Ті чи інші суттєві параметри кодуються в генному наборі особин.

Дж. Сміт (James E. Smith) та Т. Фогарті (Terence C. Fogarty) [7.35] висунули ідею організації багатобатьківського відтворення, в якому адаптивний механізм використовується для розрахунку кількості батьків. У різних підпопуляціях відбувається пошук найкращих рішень з використанням різних стратегій. Механізм спроєктовано так, щоб забезпечувалися збільшення кількості добрих рішень та обмін високоякісним генетичним матеріалом (частинами) з особинами інших підпопуляцій. У такій постановці використовується N-батьківський діагональний кросовер.

Для організації механізму самоадаптації розміру популяції Дж. Грефенстет (John J. Grefenstette) застосував метапідхід, заснований на використанні іншого генетичного алгоритму. У такий спосіб кожна ітерація метагенетичного алгоритму ініціювала запуск цільового генетичного алгоритму з адаптованими параметрами. Результатом роботи метаалгоритму стало визначення оптимального розміру популяції для цільового алгоритму.

Вперше ідею організації оператора мутації, що самопристосовується, реалізував Г.-П. Швефель (Hans-Paul Schwefel) в еволюційних стратегіях, де він намагався керувати кроком мутацій. Т. Бек (Thomas Bäck) поширив його ідеї на генетичні алгоритми. Він додав до індивіда додаткові біти, де кодувалися ймовірності мутації. Кожну закодовану ймовірність мутації застосовували як до самої себе, так і до відповідного гена. Експериментальні дослідження показали, що в разі застосування такого підходу досягається суттєве підвищення продуктивності генетичного алгоритму, можна також комбінувати схеми адаптації параметрів генетичного алгоритму в рамках одного завдання. Наприклад, організація механізму адаптації одночасно і кросовера, і мутації може істотно збільшити продуктивність генетичного алгоритму.

Отже, управління параметрами дає можливість використовувати різні, найбільш придатні значення параметрів на різних етапах генетичного

пошуку, щоб забезпечити максимальну продуктивність алгоритму. Адаптивні та самопристосовувані генетичні алгоритми ускладнюють власне алгоритм, делегуючи йому завдання визначення значень стратегічних параметрів, але разом з тим вони позбавляють дослідника необхідності регулювати параметри вручну.

Наступним етапом розвитку динамічних генетичних алгоритмів стало створення багаторівневих динамічних генетичних алгоритмів (*nGA*) [7.2]. Нижній рівень такого алгоритму безпосередньо виконує завдання покращення популяції рішень. Верхні рівні є генетичними алгоритмами, які вирішують оптимізаційне завдання щодо поліпшення параметрів алгоритму нижнього рівня. При цьому як цільову функцію зазвичай використовують швидкість роботи алгоритму нижнього рівня та швидкість поліпшення ним популяції від покоління до покоління.

Ефективна стратегія — пов'язати інтенсивність мутацій з фітнес-індексом індивіда: чим вищий фітнес-індекс індивіда, тим меншою має бути ймовірність мутацій його генів, і навпаки, що нижчий фітнес-індекс індивіда, то вища ймовірність мутацій його генів.

Для подань з рухомою комою на характеристики ГА впливає розмір кроку мутацій. Ідеальна стратегія — стартувати зі значними розмірами мутацій, щоб забезпечити великі випадкові стрибки у пошуковому просторі. Розмір кроку мутацій з часом зменшують так, щоб отримати дуже малі його зміни до закінчення процесу пошуку. Розмір кроку може бути також обернено пропорційним фітнес-індексу індивіда, для того щоб менш придатні індивіди мали більший розмір кроку, ніж більш придатні.

Інтенсивність кросовера P_c також значно впливає на характеристики ГА. Оскільки оптимальне значення P_c залежить від виконуваного завдання, для динамічного налаштування P_c можна використати ті самі альтернативні стратегії, що і для P_m.

На додаток до P_m та P_c вибір найкращого оператора селекції є також проблемно орієнтованим. Хоча найкращу комбінацію операторів кросовера мутації та селекції разом із кращими значеннями керівних параметрів можна отримати емпіричним шляхом, є велика кількість алгоритмів адаптації. Ці методи адаптивно переходять на різні оператори на основі поточних результатів пошуку. Загалом визначення найкращих наборів операторів і параметрів управління є проблемою багатокритеріальної оптимізації.

Стратегії селекції. Описані вище ГА значно відрізняються від біологічної еволюції тим, що розмір популяції фіксований. Це дає змогу описувати процес селекції, що складається з двох кроків:

1) селекція батьків;

2) стратегія заміщення, що вирішує, чи замінить нащадок батьків та яких саме батьків належить замінити.

Можна виділити два основні класи ГА на основі стратегії заміщення батьків: поколінські ГА і фіксовані ГА стійкого (фіксованого) стану [7.38]. У поколінських ГА стратегія заміщення замінює всіх батьків їхніми нащадками після того, як нащадки були згенеровані і зазнали мутацій. У результаті немає накладання (перетину) між поточною та новою популяціями (у припущенні, що елітизм не використовується). У ГА стійкого стану після генерації та мутації нащадка слід негайно вирішити: зберегти батька чи нащадка для наступного покоління. Отже, є перетин між поточним і наступним поколіннями.

Величина перетину між поточним і новим поколіннями називається поколінським зазором. Поколінські ГА мають нульовий зазор, тоді як ГА стійкого стану характеризуються великими проміжками між поколіннями.

Для ГА фіксованого стану розроблено численні стратегії [7.1, 7.2].

1. Замінити найгіршого, коли нащадок замінює найгіршого індивіда з поточного покоління.

2. Випадкова заміна, коли нащадок замінює випадково вибраного індивіда з поточного покоління.

3. Заміна найстарішого, стратегія *FIFO* (*first-in — first-out*), у якій відбувається заміна найстарішого індивіда з поточної популяції. Така стратегія має високу ймовірність заміни одного з найкращих індивідів.

4. Турнірна стратегія, коли випадково вибирається група індивідів і найгірший індивід із цієї групи замінюється нащадком. Альтернативно використовується розмір турніру, що дорівнює двом, і найгірший індивід замінюється нащадком з імовірністю $0{,}5 \le p_r \le 1$.

5. Консервативна стратегія — комбінує стратегію *FIFO* (замінити найстарішого) з модернізованою детермінованою турнірною стратегією. Розмір турніру складається з двох індивідів, один з яких є найстарішим у поточній популяції. Найгірший із двох замінюється нащадком. За такого підходу найстарішого індивіда не буде втрачено, якщо він має найкращий фітнес-індекс.

5. Стратегія «елітизму», за якого найкращі батьки вилучаються із селекції і автоматично передаються в наступне покоління.

6. Змагання «батько — нащадок», у якому використовується стратегія селекції, щоб вирішити, чи замінить нащадок одного зі своїх батьків.

Острівні генетичні алгоритми. ГА добре пристосовані для паралельної реалізації. Розрізняють три основні категорії паралельних ГА.

1. Однопопуляційні ГА, типу «керівник — підлеглий» (*master — slave*), у яких оцінка фітнес-індексу розподілена по декількох процесорах.

2. Однопопуляційні клітинні ГА, де кожний індивід приписаний до одного процесора і кожний процесор приписаний (призначений) лише одному індивіду. Для кожного індивіда визначається мала сфера сусідства, а селекція та репродукція обмежені лише сферою сусідства.

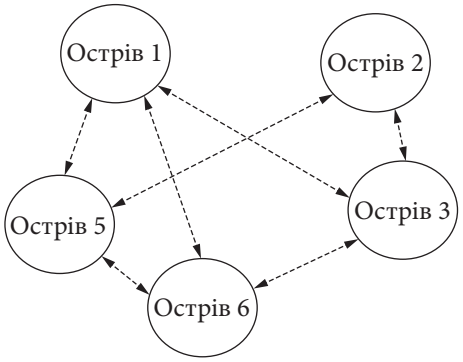


Рис. 7.6. Острівні ГА

3. Мультипопуляційні, або острівні, ГА, в яких використовується множина популяцій, кожна на окремому процесорі. Обмін інформацією між популяціями здійснюється завдяки міграційній політиці. Хоча острівні ГА було розроблено для паралельної реалізації на багатьох процесорах, їх можна реалізувати в однопроцесорній системі.

Розглянемо детальніше острівні ГА (рис. 7.6). У острівному генетичному алгоритмі низка популяцій розвивається паралельно кооперативній конструкції. У такій моделі ГА є множина островів, кожний острів є однією популяцією. Селекція, кросовер та мутації відбуваються в кожній популяції незалежно від інших. До того ж індивідам дозволяється мігрувати між островами (або популяціями), як показано штриховими лініями на рис. 7.6.

Невід'ємною частиною острівного ГА є міграційна політика, яка керує обміном між островами. Міграційна політика визначає [7.1, 7.2]:

1) комунікаційну топологію, яка встановлює шляхи міграції індивідів між островами. Наприклад, кільцева топологія (див. рис. 7.6) припускає обмін інформацією між сусідніми островами. Комунікаційна топологія визначає, наскільки швидко (або повільно) хороші рішення поширюються в інших популяціях. Для слабозв'язаної структури (як-от кільцева топологія) острови більш ізольовані один від одного і поширення інформації про добрі рішення відбувається повільніше. Слабозв'язані топології також сприяють появі безлічі рішень. Сильнозв'язані структури мають велику швидкість поширення інформації, яка може спричинити передчасну збіжність;

2) інтенсивність міграції, що, у свою чергу, визначає частоту міграцій. Тісно пов'язане з інтенсивністю міграції питання — коли має відбуватися міграція. Якщо міграція відбувається зарано, кількість хороших будівельних блоків у мігрантах може бути надто малою, щоб впливати на кінцеві результати. Зазвичай міграція відбувається, коли кожна популяція зійшлася на якомусь рішенні. Після обміну індивідами процеси розвитку популяції перезапускаються знову;

3) механізм селекції, який допомагає вирішити, які саме індивіди мають мігрувати;

4) стратегію заміни, що сприяє вирішенню, який індивід вузла призначення буде замінено мігрантом.

На основі стратегій селекції та заміни (заміщення) острівні ГА можна згрупувати на два класи алгоритмів — статичні та динамічні острівні ГА.

Для статичних острівних ГА застосовують детерміновані стратегії селекції та заміни, наприклад [7.38]:

- добрий мігрант замінює поганого індивіда;
- добрий мігрант замінює випадково обраного індивіда;
- випадково вибраний мігрант замінює поганого індивіда;
- випадково вибраний мігрант замінює випадково вибраного індивіда.

Для селекції хорошого мігранта можна використати будь-якого з фітнес-пропорційних операторів селекції. Наприклад, для стратегії елітизму найкращий індивід однієї популяції переходить в іншу популяцію.

Лоуренс Дж. Фогель (Lawrence J. Fogel) [7.8] використовує турнірну селекцію, розглядаючи два випадково обраних індивіди. Найкращий з цих двох мігрує, тоді як найгірший замінюється на індивіда, що виграв турнір, із сусідньої популяції.

Динамічні моделі не використовують топологію визначення шляхів міграції. Натомість рішення про міграцію виконуються імовірнісним способом. Міграція відбувається з певною ймовірністю. Якщо міграція з острова відбувається, то острів призначення також визначається ймовірно. Можна застосувати турнірну селекцію з використанням середніх значень показників фітнес-індексу популяції. Додатково можна звернутися до стратегії прийняття рішень, щоб з'ясувати, чи має бути прийнятий іммігрант. Наприклад, іммігранта може бути прийнято з деякою ймовірністю, якщо його фітнес-індекс (Ф-І) вищий ніж інші значення Ф-І цього острова.

Ще один важливий аспект, який слід розглянути щодо острівних ГА, — як ініціалізувати популяції. Звичайно, можна використовувати випадковий підхід, який призведе до того, що різні популяції покриватимуть ту саму частину простору пошуку. Найкращий підхід — ініціалізувати різні популяції так, щоб вони покривали різні частини пошукового простору і сприяли утворенню свого роду ніш в індивідуальних островах. Далі необхідний крок на метарівні для комбінування рішень з різних островів.

Іншим типом острівних ГА є кооперативний коеволюційний ГА (ККГА) [7.38]. У цьому разі замість розподілу всіх індивідів за декількома (суб)популяціями кожній популяції дається один або кілька генів (одна змінна рішення) для оптимізації. Субпопуляції є взаємовиключними, кожна має на меті розвиток одного або кількох генів. Тому кожна популяція оптимізує один (або кілька) генів. Тобто жодна підпопуляція не має необхідної інформації для вирішення проблеми автономно. Навпаки, інформація від усіх популяцій має бути скомбінована, щоб сформулювати рішення.

Умови зупинки. У генетичних алгоритмах можуть використовуватися різні умови зупинки. Серед тих, що найчастіше трапляються, слід зазначити такі:

- за кількістю ітерацій алгоритму;

• за досягнутим значенням фітнес-індексу у найкращого індивіда—особини;

• якщо значення фітнес-індексу у найкращої особини не змінюється впродовж заданого числа ітерацій.

Багатокритеріальна оптимізація. Значну увагу приділено застосуванню ГА для вирішення завдань багатокритеріальної оптимізації (БКО) виду

$$\min_x f_i(x; c_i), \quad i = \overline{1, k}. \tag{7.22}$$

Алгоритм VEGA. Одним із перших генетичних алгоритмів для розв'язання задач БКО є векторний генетичний алгоритм *VEGA (Vector Evaluated Genetic Algorithm)* [7.40], у якому кожну підпопуляцію пов'язують з однією цільовою функцією. Далі застосовують селекцію до кожної субпопуляції до створення пулу претендентів. У результаті процесу селекції найкращі індивіди з кожної цільової функції вносяться до пулу претендентів. Потім виконується кросовер шляхом вибору батьків із пулу претендентів.

Генетичний алгоритм із нішею Парето. Дж. Горн (Jeffrey Horn) зі співавторами Н.М. Нафплітісом (Nicola Nafpliotis) і Д.Е. Голдбергом (David E. Goldberg) розробили ГА з нішею Парето (ГАНП), в якому використовується оператор турнірної селекції, що адаптується, для знаходження недомінованих рішень [7.5]. Оператор Парето — оператор домінованої турнірної селекції — вибирає два індивіди-кандидати і створює множину порівняння. Якщо один кандидат домінується якимось індивідом із безлічі порівняння, а інший — ні, то обирається останній. Якщо жоден не домінується або домінуються обидва, то використовується процедура поділу на класи еквівалентності для вибору одного індивіда. Вибирається індивід з найменшою кількістю очок у ніші, де число очок — це число індивідів у ніші радіуса σ_{share} щодо цього кандидата. Така стратегія віддає перевагу рішенням на найменш «населеній» частині фронту Парето.

7.2. Застосування генетичного алгоритму для задачі структурного синтезу мереж

Важливим додатком ГА є завдання комбінаторної оптимізації, зокрема завдання структурного синтезу обчислювальних мереж, що задаються орієнтованими графами. Розглянемо практичне завдання синтезу обчислювальних мереж із перспективною технологією багатопротокольної комутації міток (*MPLS*) [7.22].

Задано множину вузлів мережі $X = \{x_j\}$, $j = \overline{1, n}$, — маршрутизаторів *MPLS* (так званих *LRS — Label Switching Routers*), їх розміщення на території регіону, набір пропускних здатностей каналів зв'язку $D = \{d_1, d_2, \ldots, d_k\}$, у тому чис-

лі виконується синтез їх питомих цін на одиницю довжини $C = \{c_1, c_2, \ldots, c_k\}$, визначені класи обслуговування *CoS* (*Class of Service*), відомі матриці вхідних вимог для k-го класу $H(k) = ||h_{ij}(k)||$, $i, j = \overline{1, n}$; $k = 1, 2, \ldots, K$, де $h_{ij}(k)$ — інтенсивність потоку k-го класу, який необхідно передавати з вузла i у вузол j за одиницю часу (кбіт/с). Крім того, введено обмеження на показники якості *QoS* для кожного класу k у вигляді обмеження на середню затримку $T_{\text{зат},k}$, $k = \overline{1, K}$.

Потрібно визначити структуру мережі як набору каналів зв'язку $E = \{(r, s)\}$, вибрати пропускні здатності (ПС) каналів зв'язку $\{\mu_{rs}\}$ та знайти розподіл потоків усіх класів $F(k) = [f_{rs}(k)]$ так, щоб забезпечити передання вимог усіх класів $H(k)$ у повному обсязі та із затримками $T_{\text{ср}}$, що не перевищують задані $T_{\text{зат},k}$, і при цьому виконувалися б обмеження на частку втрачених пакетів CLP_k, а вартість мережі була б мінімальною [26].

Складемо математичну модель цієї задачі синтезу.

Потрібно знайти таку структуру мережі E, для якої

$$\min_{E\{\mu_{rs}\}} C_{\Sigma}(E) = \Sigma_{(r,s)\in E} C_{rs}(\{\mu_{rs}\}) \tag{7.23}$$

за умов

$$T_{\text{ср}}(\{\mu_{rs}\};\{f_{rs}\}) \le T_{\text{зат},k},\ k = \overline{1, K}; \tag{7.24}$$

$$f_{rs} \le \mu_{rs} \text{ для всіх } (r, s); \tag{7.25}$$

$$\mu_{rs} \in D; \tag{7.26}$$

$$CLP_k(\{\mu_{rs}\}; \{f_{rs}\}) \le CLP_{k_{\text{зат}}}, \tag{7.27}$$

де $CLP_k(\{\mu_{rs}\}; f_{rs})$ — частка втрачених пакетів для потоку k-го класу (пріоритету); $CLP_{k_{\text{зат}}}$ — задане обмеження на цю величину.

Ця задача належить до так званих NP-важких задач оптимізації. Для її вирішення розроблено загальний алгоритм генетичної оптимізації, що використовує обидва оператори ГА — кросовер та мутації [7.22].

Визначаємо матрицю наявності каналів зв'язку як

$$K = ||k_{ij}||,$$

де $k_{ij} = \begin{cases} 1, \text{ якщо канал зв'язку } (i - j) \text{ є у структурі,} \\ 0 \text{ — у протилежному випадку,} \end{cases}$ для кожної зі структур (мереж).

Далі генеруємо початкову популяцію з n різних структур у заданому класі — багатозв'язні структури зі ступенем зв'язності 2. Для синтезу використовуватимемо напіводнорідний кросовер, що виправдано за невеликого розміру популяції. Батьків (структури $E_i(k)$) вибираємо випадково з імовірністю, обернено пропорційною $C_{\Sigma}(E_i(k))$, кожному з батьків відповідає матриця K^i, $i = 1, 2$. У процесі напіводнорідного кросовера кожному нащадку переходить рівно половина генів батьків. Маску кросовера пода-

ємо у вигляді матриці: $M = ||m_{ij}||$ (де $m_{ij} = \begin{cases} 0, p \geq p0; \\ 1, p < p0; \end{cases}$ $p0 = 0{,}5$ — параметр, а $p \in [0, 1]$ — випадкова величина). Схематично цей процес кросовера має вигляд $E(k)' = ||e(k)'_{ij}|| = \begin{cases} (i-j)^1, k_{ij}^{\ 1} = 1, m_{ij} = 0; \\ (i-j)^2, k_{ij}^{\ 2} = 1, m_{ij} = 1. \end{cases}$

У результаті кросовера з метою максимізації продуктивності алгоритму генеруємо одного нащадка. Якщо після операції кросовера залишаються ізольовані підграфи, то підключаємо їх прямими каналами зв'язку (КЗ) до кореня. Далі для одержаного нащадка — структури $E(k)$ вирішуємо завдання вибору пропускних здатностей та розподілу потоків (ВПС / РП), знаходимо нові пропускні здатності (ПЗ) та розподіли потоків (РП). У результаті виходячи з критерію приймаємо рішення про введення / невведення структури $E(k)$ у послідовність локально-оптимальних структур П.

Визначивши вигляд та порядок виконання операції кросовера, необхідно перейти до визначення схеми зміни мутації. Зауважимо, що базовий алгоритм передбачав використання безумовних мутацій. Мутації полягають у видаленні та введенні деяких нових каналів зв'язку в структуру мережі. У процесі вдосконалення алгоритму у парі з певним напіводнорідним кросовером було застосовано такі схеми зміни ймовірності мутації.

1. Детерміністична. У детерміністичній постановці ймовірність мутації визначається за допомогою функції, яка залежить від часу. Таким чином, змінюватимемо ймовірність мутації за законом: $\sigma(t) = 1 - ct/T$, де $0 \leq t \leq T$, а $c \in (0, 1)$ — коефіцієнт. Зауважимо, що з часом ймовірність зменшуватиметься. Зазначимо основні особливості такого підходу:

1) зміна ймовірності мутації залежить від успішності її застосування у процесі генетичного пошуку;

2) дослідник повністю контролює зміну ймовірності мутації за певною формулою;

3) зміна ймовірності мутації цілком передбачувана.

2. Адаптивна. Для реалізації адаптивної схеми зміни ймовірності мутації застосуємо правило Рехенберга (Rechenberg) [7.22]. Таким чином, закон для зміни ймовірності мутації матиме вигляд $\sigma(t) = \begin{cases} \sigma(t-1)/\lambda, \text{якщо } \varphi(t-) < 1/5; \\ \sigma(t-1)\lambda, \text{якщо } \varphi(t-1) > 1/5; \\ \sigma(t-1), \text{якщо } \varphi(t-1) = 1/5 \end{cases}$ (де $\phi(t)$ — відсоток хороших мутацій, а $\lambda = 1{,}1$ — навчальний коефіцієнт). Зазначимо основні особливості такого підходу:

1) зміна ймовірності мутації залежить від успішності її застосування у процесі генетичного пошуку;

2) дослідник має уявлення про те, як змінюватиметься ймовірність мутації;

3) зміна ймовірності мутації непередбачувана.

3. Самопристосовна. Самопристосовна мутація може бути організована лише на рівні хромосом (структур-мереж), і навіть генетично (каналів

зв'язку). Для цього задається закон для зміни ймовірності мутації $\sigma(t)$, а потім $\sigma(t)$ кодується в хромосомі як $\{E(k), \sigma(t)\}$ або $\{E(k) = ||e_{ij}(k)||, ||\sigma_{ij}(t)||\}$. Проте ціною реалізації подібного підходу є значне зниження продуктивності алгоритму, і в рамках поставленого завдання мутація, що самопристосовується, — не найкраща альтернатива. Зазначимо основні особливості такого підходу:

1) зміна ймовірності мутації є наслідком природного вибору;

2) дослідник практично не контролює процес зміни ймовірності мутації;

3) зміна ймовірності мутації непередбачувана.

На противагу схемою безумовного кросовера та динамічної мутації була реалізована схема безумовної мутації та динамічного кросовера. У процесі вдосконалення алгоритму [7.22] у парі з безумовною мутацією було застосовано такі схеми зміни ймовірності кросовера.

1. Детерміністична. Реалізація детерміністичної схеми зміни ймовірності кросовера ґрунтується на гіпотезі, що на різних етапах генетичного пошуку кросовер може бути більш-менш значущим. Тому як функцію зміни ймовірності кросовера має сенс вибрати немонотонну функцію. Таким чином, змінюватимемо ймовірність кросовера за таким законом: $\sigma(t) = |\sin(t)|$, де $0 \le t \le T$. Зауважимо, що з часом ймовірність буде то зменшуватися, то збільшуватися.

2. Адаптивна. Визначимо адаптивний кросовер як оператор, ймовірність застосування якого зменшується, якщо популяція є досить гомогенною, і збільшується, якщо популяція є гетерогенною. Мірою гомогенності / гетерогенності популяції вважатимемо $C_\Delta = \max(C_\Sigma(E_i(k)) - C_\Sigma(E_j(k))$, $i \in [1, ..., n]$, $j \in [1, ..., n]$, $i \ne j$ (де $n = 3$ — розмір популяції).

Логічно припустити, що за наявності досить схожих особин у популяції операція кросовера буде неефективною, і навпаки. Таким чином, в адаптивній постановці закон зміни ймовірності кросовера матиме вигляд

$$\sigma(t) = \begin{cases} \sigma(t-1)\lambda, \text{ якщо } C_\Delta > C^*; \\ \sigma(t-1)/\lambda, \text{ якщо } C_\Delta < C^*; \\ \sigma(t-1), \text{ якщо } C_\Delta = C^* 5 \end{cases}$$

(де C^* — граничне значення, а $\lambda = 1{,}1$ — начальний коефіцієнт).

3. Самопристосовна. З подібних причин реалізація кросовера, що самопристосовується, є недоцільною з точки зору продуктивності алгоритму, як і реалізація самопристосовуваної мутації.

Після проведення серії експериментів для проєкту мережі *MPLS* у масштабах України (за різних початкових умов та параметрів задачі, як-от обсяги вимог з трафіків, задані набори пропускних спроможностей каналів зв'язку тощо, а також за зміни коефіцієнтів обсягів вимог щодо трафіків) були отримані результати.

1. Комбінація безумовного кросовера та динамічної детерміністичної мутації. За $c = 0{,}3$ дана реалізація виявилася найбільш успішною (підви-

щення швидкодії на 15 %), що підтверджує гіпотезу про розподіл ролей між мутацією та кросовером на різних етапах генетичного пошуку. Отже, мутації відіграють ключову роль на початковому етапі пошуку під час дослідження простору рішень, а на заключному етапі пошуку найбільш ефективне використання кросовера для знаходження оптимального рішення на основі вже отриманих рішень.

2. Комбінація безумовного кросовера та динамічної адаптивної мутації. Ця реалізація не дала можливості досягти стабільного зменшення часу роботи алгоритму, що свідчить про неефективне застосування такого підходу в рамках поставленого завдання.

3. Комбінація безумовного кросовера і динамічної мутації, що самопристосовується. Ця реалізація є недоцільною, оскільки істотно ускладнює процес генетичного пошуку та веде до зниження продуктивності алгоритму.

4. Комбінація безумовної мутації та динамічного детерміністичного кросовера. Ця реалізація не дала змоги отримати стабільний приріст продуктивності алгоритму. Ймовірно, для отримання кращого результату необхідне тонше налаштування закону зміни ймовірності кросовера.

5. Комбінація безумовної мутації та динамічного адаптивного кросовера. Ця реалізація виявилася найуспішнішою (підвищення швидкодії на 20—22 %). Отже, підтверджується гіпотеза про те, що оператор кросовера має низку додатних властивостей, яких не має оператор мутації, проте застосування оператора кросовера виправдане лише у випадку, коли особини в популяції достатньо відрізняються одна від одної, тобто популяція є гетерогенною.

6. Комбінація безумовної мутації і динамічного кросовера, що самопристосовується. Ця реалізація є неефективною через невиправдану трудомісткість процесу генетичного пошуку.

7.3. Еволюційне програмування

Еволюційне програмування (ЕП) як науковий напрямок сформувалося завдяки роботі Л. Фогеля, А. Оеуна і М. Уолша [7.8], які використовували моделювання еволюції для розвитку штучного інтелекту. Хоча ЕП має спільну мету імітації природного процесу еволюції з генетичними алгоритмами та генетичним програмуванням, воно суттєво відрізняється від них тим, що ЕП ґрунтується на моделюванні адаптивної поведінки та наголошує на розвитку поведінкових моделей, а не генетичних. Тобто ЕП розглядає так звану *фенотипічну* еволюцію, а не *генотипічну*. В ЕП ітеративно застосовуються два оператори еволюції, а саме: мутації, що забезпечують необхідну різноманітність, та селекції. Оператори рекомбінації зазвичай у межах ЕП не використовуються.

Еволюційне програмування було винайдено на початку 60-х років XX століття як підхід, альтернативний штучному інтелекту, який концентрувався на моделях інтелекту людини. З його парадигми інтелект може розглядатись як «властивість, яка дає змогу системі адаптувати свою поведінку, щоб забезпечити досягнення бажаної мети в широкому діапазоні середовищ», і була створена модель, що імітує еволюцію характеристик поведінки. Еволюційний процес, спочатку розроблений для вдосконалення машин (автоматів з кінцевим числом станів), полягає у знаходженні набору оптимальних поведінок у просторі поведінок, що спостерігаються. Тому на противагу іншим еволюційним алгоритмам фітнес-індекс ЕП вимірює «поведінковий шум» індивіда відносно його середовища.

Еволюційне програмування використовує 4 основні компоненти еволюційних алгоритмів (ЕА) [7.8].

1. Ініціалізація. Подібно до інших парадигм ініціалізується початкова популяція індивідів так, щоб рівномірно покрити область проблем оптимізації.

2. Мутації. Оператор мутації має на меті ввести варіації в популяцію, тобто згенерувати нових кандидатів-рішень. Кожен з батьків виробляє одного або кількох нащадків за допомогою оператора мутації. Розроблено велику кількість операторів мутації.

3. Оцінка. Використовується фітнес-функція для кількісної оцінки поведінкового шуму індивідів. У той час як фітнес-функція забезпечує абсолютну міру пристосованості, щоб оцінити наскільки добре індивід вирішує проблему оптимізації, виживання в ЕП зазвичай ґрунтується на відносній мірі фітнес-індексу. Цей захід дає змогу порівнювати групу індивідів між собою, а індивіди, які виживають і переходять у наступне покоління, вибираються на основі цього заходу.

4. Селекція. Основна мета оператора — вибрати індивідів, які переходять у наступне покоління. Селекція — процес змагання, в якому батьки та їхні нащадки змагаються між собою і з іншими претендентами за виживання.

Оператори мутації та селекції застосовуються ітеративно доти, доки не буде виконано умову зупинки. Можуть бути використані будь-які умови зупинки, розглянуті для генетичних алгоритмів.

Порівнюючи ГА та ЕП, можна виокремити такі відмінності.

1. ЕП наголошує на фенотипічній еволюції замість генотипічної. Відповідно до викладеного вище, ЕП не використовує жодних операторів рекомбінації (кросовера): тут немає обміну генетичним матеріалом;

2. ЕП використовує відносні фітнес-функції, щоб кількісно оцінити функціонування щодо випадково обраної групи індивідів.

3. Селекція ґрунтується на змаганні. Ті індивіди, що функціонують (працюють) краще за групи суперників, отримують підвищену ймовірність уведення в нове покоління.

4. Батьки та нащадки борються за виживання (існування).

5. На поведінку індивідів впливають параметри стратегії, що визначають обсяг варіації між батьками та нащадками.

Оператори ЕП

Оператори мутації. Оскільки мутації — це єдиний засіб введення варіативності в популяцію ЕП, дуже важливо, щоб проєктування оператора мутацій забезпечувало компроміс між фазами дослідження—використання. Після початкової фази досліджень індивідам має бути дозволено використовувати отриману інформацію про простір пошуку для тонкого налаштування рішень.

Для цього припустимо, що задача полягає у мінімізації неперервної функції без обмежень . Якщо $x_i(t)$ позначає претендента на рішення, подане i-м індивідом, у якому t — час, то кожен $x_{ij}(t) \in R$, $j = \overline{1, n_x}$.

У загальному випадку мутація описується виразом

$$x'_{ij}(t) = x_{ij}(t) + \Delta x_{ij}(t), \tag{7.28}$$

де $x'_{ij}(t)$ — нащадок, згенерований з батька $x_{ij}(t)$ додаванням кроку розміром $\Delta x_{ij}(t)$ до батька.

На розмір кроку діє шум із деяким розподілом імовірностей, де девіація шуму визначається стратегічним параметром σ_{ij}. У цьому випадку розмір кроку обчислюється як

$$\Delta x_{ij}(t) = \Phi(\sigma_{ij})\zeta_{ij}(\mathrm{t}), \tag{7.29}$$

де Φ: $R \to R$ — функція, яка масштабує вплив шуму $\zeta_{ij}(t)$.

На основі характеристик функції масштабування Ф алгоритми ЕП можна розподілити на три категорії [7.2]:

1) *неадаптивні.* І тут $\Phi(\sigma) = \sigma$. У цьому разі відхилення у розмірі кроку — статичні;

2) *динамічні*, у яких відхилення у розмірі кроку змінюють з часом, використовуючи деяку детерміновану функцію Ф, зазвичай фітнес-функцію;

3) *самонавчальні*, у яких відхилення у розмірі кроку змінюють динамічно. Система навчається для знаходження найкращих значень σ_{ij} паралельно зі змінними рішення.

Оскільки відхилення σ_{ij} впливають на поведінку індивідів у разі динамічних і самонавчальних алгоритмів ЕП, ці відхилення розглядаються як стратегічні параметри.

Як і у випадку зі всіма еволюційними алгоритмами, ЕП використовує стохастичний процес пошуку. Стохастичність запроваджується обчисленням величини кроку як функції шуму ζ_{ij}, що генерується за допомогою деякого ймовірнісного розподілу. При цьому для ЕП використовуються такі розподіли.

1. Рівномірний. Шум є вибіркою з рівномірного розподілу:

$$\zeta_{ij}(t) \sim u(x_{ij})_{\min}. \tag{7.30}$$

При цьому слід зазначити, що $E\{\zeta_{ij}\} = 0$ для перешкоджання будь-якому зміщенню внаслідок шуму, де $E\{*\}$ — оператор математичного очікування.

К. Вонг (Kit Po Wong) і Дж. Юревич (Juri Yuryevich) запропонували рівномірний оператор мутацій [7.20]:

$$\Delta x_{ij}(t) = U(0, 1)(\hat{y}_j(t) - x_{ij}(t)), \tag{7.31}$$

де $\hat{y}_j(t)$ — найкращий індивід з поточної популяції $C(t)$; $U(0,1)$ — рівномірний розподіл. Цей оператор мутацій змушує всіх індивідів робити випадкові кроки до кращого індивіда, що дуже нагадує соціальний компонент, який використовується в ройових алгоритмах оптимізації (див. розд. 8).

2. Гауссів розподіл. Для гауссових операторів мутації шум генерується з нормального розподілу з нульовим середнім: $\zeta_{ij}(t) \sim N(0, \sigma_{ij}(t))$.

Для порівняння з іншими розподілами гауссова щільність розподілу визначається за формулою

$$f_G(x) = \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2\sigma^2}}. \tag{7.32}$$

3. Розподіл Коші. Для операторів мутації з розподілом Коші (Augustin-Louis Cauchy) маємо вираз

$$\zeta_{ij}(t) \sim C(0,\nu), \tag{7.33}$$

де ν — параметр масштабу, функція щільності ймовірностей Коші центрується щодо початку координат і задається виразом

$$f_C(x) = \frac{1}{\lambda} \frac{\nu}{\nu + x^2}, \tag{7.34}$$

де $\nu > 0$.

Відповідна функція розподілу задається формулою

$$(x) = \frac{1}{2} + \frac{1}{\pi} \arg \operatorname{tg}(\frac{x}{\nu}).$$

Розподіл Коші має ширші «хвости», ніж розподіл Гаусса, і тому виробляє більші мутації.

4. Розподіл Леві. Для розподілу Леві (Paul Pierre Lévy) виконується $\zeta_{ij}(t) \sim L(\nu)$.

Розподіл Леві з центром на початку координат визначається виразом

$$F_{L,\nu,\gamma}(x) = \frac{1}{\pi} \int_0^\infty e^{-\gamma q^\nu} \cos(qx) dq, \tag{7.35}$$

де $\gamma > 0$ — коефіцієнт масштабу та $0 < \nu < 2$ регулює форму розподілу.

Якщо ν = 1, то отримаємо розподіл Коші, якщо ν = 2 — то розподіл Гаусса.
Для $|x| \gg 1$ розподіл Леві може бути апроксимовано так:

$$f_L(x) \sim ax^{-(\nu+1)}.$$

5. Експоненційний. У цьому випадку $\zeta_{ij}(t) \sim E(0, \xi)$, а функція щільності експоненційного розподілу визначається виразом

$$f_{E,\xi}(x) = \frac{\xi}{2} e^{-\xi|x|}, \tag{7.36}$$

де $\xi > 0$ регулює варіацію (яка дорівнює $2/\xi^2$).

Випадкові числа експоненційного розподілу можна визначити як

$$x = \begin{cases} \frac{1}{\xi} \ln(2y), \text{ якщо } y \le 0{,}5; \\ \frac{1}{\xi} \ln(1-y), \text{ якщо } y > 0{,}5, \end{cases} \tag{7.37}$$

де $y \sim U(0, 1)$.

Легко помітити, що $E(0, \xi) = \frac{1}{\xi} E(0, 1)$.

6. Хаотичний. Хаотичний розподіл використовується для генерування шуму згідно з формулою

$$\zeta_{ij}(t) \sim R(0, 1), \tag{7.38}$$

де $R(0, 1)$ означає хаотичну послідовність у просторі (−1; +1), яка генерується згідно з рекурентним співвідношенням:

$$x_{t+1} = \sin(\frac{2}{x_t})x_t; \quad t = 0, 1, 2. \tag{7.39}$$

7. Комбінований. Запропоновано усереднений оператор мутацій, який є лінійною комбінацією розподілів Коші і Гаусса:

$$\zeta_{ij}(t) = \zeta_{N,ij}(t) + \zeta_{C,ij}(t), \tag{7.40}$$

де $\zeta_{N,ij}(t) \sim N(0, 1)$ і $\zeta_{C,ij}(t) \sim C(0, 1)$.

Підсумковий розподіл генерує більше дуже малих і великих мутацій порівняно з розподілом Гаусса.

Було запропоновано також адаптивний усереднений оператор мутацій:

$$\Delta x_{ij}(t) = \gamma_{ij}(t)(C_{ij}(0, 1)+\nu_{ij}(t)N(0, 1)), \tag{7.41}$$

де $\gamma_{ij}(t) = \sigma_{2ij}(t)$ — загальний коефіцієнт, що масштабується, а $\nu_{ij}(t) = \sigma_{1ij} / \sigma_{2ij}$ впливає на поведінку розподілу випадкових змін, зокрема на ймовірність виникнення великих мутацій. При цьому розподіл Коші завдяки великим «хвостам» генерує більше великих мутацій, ніж розподіл Гаусса.

Розподіл Леві має «хвости», що за ступенем спадання знаходяться між розподілами Гаусса та Коші. Це означає, що він характеризується повільнішим спаданням імовірності великих відхилень, ніж нормальний (Гауссів) розподіл, але швидшим ніж розподіл Коші. Таким чином, розподіл Коші доцільно використовувати для глобального пошуку, коли кандидати рішень знаходяться далеко від оптимуму, оскільки він забезпечує високу ймовірність великих кроків. Натомість нормальний розподіл (Гаусса) ефективніший у фазі локального пошуку поблизу оптимуму, оскільки генерує малі мутації для точного налаштування. Розподіл Леві, як проміжний варіант, забезпечує хороший баланс між глобальним пошуком і локальною оптимізацією, оскільки дозволяє як великі, так і малі кроки залежно від параметрів.

Ще одним важливим фактором, що впливає на баланс між дослідженням області (*exploration*) і її доопрацюванням (*exploitation*), є метод обчислення та адаптивного регулювання стратегічних параметрів. Ці параметри визначають масштаб і напрямок змінних, що безпосередньо впливає на розмір кроку та ефективність пошуку оптимального рішення.

Оператори селекції використовують в ЕП для вибору нащадків, які виживають і переходять у наступне покоління. У початкових роботах, присвячених ЕП та більшості його варіацій (розвитку), нове покоління вибирали з усіх батьків та нащадків, тобто батьки й нащадки змагалися за виживання. На відміну від інших ЕА змагання ґрунтується на відносних, а не абсолютних, фітнес-індексах (Ф-І). Абсолютні значення Ф-І стосуються дійсної фітнес-функції, яка кількісно оцінює, наскільки кандидат близький до оптимального рішення. З іншого боку, відносні показники фітнес-індексу визначають, наскільки відносно кращий цей індивід (порівняно з групою випадково обраних претендентів із батьків та нащадків).

У нашому розгляді використовуємо μ для позначення числа батьків (замість n_A), а λ — для числа нащадків.

Перший крок у процесі селекції — підрахунок відносного Ф-І для кожного з батьків $x_i(t)$ і нащадка $x'_i(t)$.

Позначимо $P(t) = C(t) \cup C'(t)$ — пул індивідів, що змагаються, і нехай $u_i(t) \in P(t)$, $i = 1, \ldots, \mu + \lambda$ — індивід у змагальному пулі. Тоді для кожного $u_i(t) \in P(t)$ група з n_p, що змагаються, випадково вибирається з $P(t) \setminus u_i(t)$. Для кожного $u_i(t)$ підраховується показник

$$S_i(t) = \sum_{l=1}^{n_p} S_{ie}(t), \tag{7.42}$$

де $S_{ie}(t) = \begin{cases} 1, \text{ якщо } f(u_i(t)) < f(u_l(t)); \\ 0 \text{ — інакше.} \end{cases}$

Було запропоновано альтернативну стратегію, в якій

$$S_{ie}(t) = \begin{cases} 1, \text{ якщо } r_1 < \dfrac{f(u_l(t)}{f(u_i(t)) < f(u_l(t))}; \\ 0 \text{ — інакше.} \end{cases} \tag{7.43}$$

де n_p вибирають як $n_p = [2\mu r_2 + 1]$, $r_1, r_2 \sim U(0, 1)$. У цьому випадку, якщо $f(u_i(t)) < f(u_l(t))$, що виконується, коли показник для u_i значно кращий ніж для u_1, то індивіду u_i приписується висока можливість отримання виграшного бала «1».

На основі показника S_i, приписаного кожному індивіду $u_i(t)$, можна обрати будь-який метод селекції.

1. Елітизм. Найкращих μ індивідів із об'єднаної популяції $P(t)$ вибирають у наступну популяцію $C(t + 1)$.

2. Селекція. Найкращих μ індивідів стохастично вибирають із використанням турніру.

3. Пропорційна селекція. Кожному індивіду приписується ймовірність бути обраним, яка визначається формулою

$$P_s(u_i(t)) = \frac{s_i(t)}{\sum_{l=1}^{\mu+\lambda} s_l(t)}. \tag{7.44}$$

4. Метод рулетки можна також використовувати для вибору μ індивідів у наступне покоління.

Було запропоновано різні методи вирішення того, які з батьків чи нащадків перейдуть у наступне покоління. Зокрема запропоновано, щоб кількість нащадків, що генеруються кожним із батьків, визначалася його фітнес-індексом. Чим вищий Ф-І батька, тим більше нащадків він генерує. Вибирають найкращого з нащадків, згенерованих цим батьком, і він змагається з батьком за виживання.

Змагання між батьком і нащадком засноване на *методі відпалу*. Нащадок $x_i'(t)$ виживає і переходить у наступне покоління, якщо $f(x_i'(t)) < f(x_i(t))$ або якщо виконується умова

$$e^{-\frac{f(x_i'(t)) - f(x_i(t))}{\tau(t)}} > U(0, 1), \tag{7.45}$$

де τ — так званий *температурний коефіцієнт,* при цьому $\tau(t) = \gamma\tau(t - 1)$, $0 < \gamma < 1$, інакше виживає батько.

Стратегічні параметри

Як зазначалося, розмір кроків залежить від стратегічних параметрів, що є невід'ємною частиною операторів мутації ЕП. Для подальшого викладу передбачається, що кожний компонент має свій власний стратегічний параметр і індивідів зображено у вигляді

$$x_i(t) = (x_i(t); \sigma_i(t)). \tag{7.46}$$

Статичні стратегічні характеристики. Найпростіший спосіб управління стратегічними параметрами — зафіксувати їх значення. У цьому разі функція стратегічних параметрів Φ(σ, X) — лінійна, тобто

$$\Phi(\sigma_{ij}(t)) = \sigma_{ij}. \tag{7.47}$$

Тоді нащадки обчислюються за формулою

$$x'_{ij}(t) = x_{ij}(t) + N_{ij}(0;\ \sigma_{ij}), \tag{7.48}$$

де $\Delta x_{ij}(t) = N_{ij}(0; \sigma_{ij})$

Недолік цього підходу полягає в тому, що надто мале значення σ_{ij} обмежує діапазони пошуку і уповільнює збіжність. З іншого боку, занадто велике значення для σ_{ij} обмежує можливості тонкого налаштування рішень.

Динамічні стратегії. Один із перших підходів до зміни значень стратегічних параметрів від часу — це пов'язати їх зі значеннями Ф-І, як демонструє вираз

$$\sigma_{ij}(t) = \sigma_i(t) = \gamma\, f(x_i(t)). \tag{7.49}$$

У такому разі нащадки генеруються згідно з виразом

$$x'_{ij}(t) = x_{ij}(t) + N(0;\ \sigma_i(t)) = x_{ij}(t) + \sigma_i(t)N(0, 1), \tag{7.50}$$

де $\gamma \in N(0, 1)$.

Якщо є інформація про глобальний оптимум, можна використовувати помилку індивіда замість абсолютного показника Ф-І. Однак така інформація зазвичай недоступна. Як альтернативу можна використати фенотипічну відстань від кращого індивіда:

$$\sigma'_{ij}(t) = \sigma_i(t) = \left| f(\hat{y}) - f(x_i) \right|, \tag{7.51}$$

де $\hat{y}$ — найбільш придатний індивід. Можна також використовувати відстань у просторі рішень $\sigma'_{ij}(t) = \sigma_i(t) = d(\hat{y}, x_i) = \left\| \hat{y} - x_i \right\|$, де $d(*, *)$ визначає евклідову норму між векторами.

Перевага такого підходу полягає в тому, що чим гірший індивід, тим сильніше він має бути мутований. З іншого боку, чим кращий індивід, тим менше нащадок повинен віддалятися від батька для більш тонкого налаштування позитивних рішень.

Описаний підхід має такі недоліки:

1. Якщо значення Ф-І дуже великі, то розміри кроків будуть також завеликими, що може призвести до «перескакування» через хороший мінімум індивіда.

2. Проблема ще більше зросте, якщо оптимальне значення Ф-І — велика невід'ємна величина. Якщо значення Ф-І у «хороших» індивідів великі,

це зумовить великі кроки і в результаті нащадки віддаляться від позитивних рішень. У разі, якщо доступна інформація про оптимальне значення функції, то використання величини похибки буде адекватним.

Багато пропозицій було зроблено для регулювання розмірів кроку залежно від значення фітнес-функції.

Фогель запропонував адитивний підхід, у якому

$$x'_{ij}(t) = x_{ij}(t) + \sqrt{\beta_{ij}(t)\, f(x_i) + \gamma_{ij}} + N_{ij}(0, 1), \tag{7.52}$$

де β_{ij} і γ_{ij} — константа пропорційності та параметр зміщення відповідно.

Для навчання рекурентних нейронних мереж було запропоновано підхід

$$x'_{ij}(t) = x_{ij}(t) + \beta\sigma_{ij}\, N_{ij}(0, 1), \tag{7.53}$$

де β — коефіцієнт пропорційності, а

$$\sigma_{ij}(t) = U(0, 1)\left[1 - \frac{f(x_i(t))}{f(x_{\max})}\right], \tag{7.54}$$

де $f_{\max}$ — максимальне значення Ф-І для поточної популяції.

У разі, коли вирішується задача максимізації, то $f(x_i(t))$ набуває додатних значень. Було запропоновано підхід (7.55), щоб девіації були пропорційні нормалізованим значенням Ф-І:

$$x'_{ij}(t) = x_{ij}(t) + \beta_{ij}\, \sigma_i(t) N_{ij}(0, 1), \tag{7.55}$$

де β_{ij} — коефіцієнт пропорційності, а відхилення обчислюється як

$$\sigma_i(t) = \frac{f(x_i(t))}{\sum_{l=1}^{\mu} f(x_i(t))}, \tag{7.56}$$

де μ — розмір популяції. У цьому підході вважається, що функція f мінімізується.

Дж. Юревич (Juri Yuryevich) і К. Вонг (Kit Po Wong) [7.20] запропонували використовувати у визначенні девіації інформацію про межі змінних та про Ф-І:

$$\sigma_{ij}(t) = (x_{\max_j} - x_{\min_j}) \left(\frac{f_{\max} - f(x_i(t))}{f_{\max}} + \gamma\right). \tag{7.57}$$

У цьому випадку $x_{\max}$ і $x_{\min}$ визначають межі простору рішень, а $\gamma > 0$ — параметр зміщення, щоб забезпечити ненульові девіації. Зазвичай γ приймають малим числом, тоді функція f максимізується.

Введення межі забезпечує великі мутації для компонентів з великим простором рішень і малі мутації для компонентів з малим діапазоном.

Було запропоновано, щоб величина відхилень була пропорційна відстані від кращого індивіда

$$\sigma_{ij}(t) = \beta_{ij} \left| \hat{y}_j(t) - x_{ij}(t) \right| + \gamma, \tag{7.58}$$

де $\gamma > 0$ — мала величина; коефіцієнт β_{ij} знаходять із умови $\beta_{ij} = \frac{\beta\sqrt{d(x_{\min}, x_{\max})}}{\pi}$, $\beta \in [0; 2]$, а $d(x_{\min}, x_{\max})$ визначає ширину простору пошуку як евклідову відстань між $x_{\min}$ та $x_{\max}$.

Адаптація (самонавчання) ЕП

В еволюційному програмуванні акцентують на розвитку моделей поведінки і ЕП реалізується моделюванням адаптивної поведінки. Раніше зазначали істотний вплив стратегічних параметрів на фітнес-функцію. Дві головні проблеми щодо стратегічних параметрів — це обсяг шуму мутацій, який має бути додано, та розмір кроку мутацій. Для того щоб забезпечити дійсно самоорганізовану поведінку, стратегічні параметри мають розвиватися (або навчатися) паралельно зі змінними рішеннями. Алгоритми ЕП, у яких використовуються такі механізми, називають самонавчальними. Вперше стосовно нейронних мереж ідеї самонавчання та самоорганізації були висловлені О.Г. Івахненком (Oleksiy Hryhorovych Ivakhnenko) та Ф. Розенблаттом (Frank Rosenblatt). Перші пропозиції щодо створення алгоритму ЕП, що самонавчається, були зроблені Л.Дж. Фогелем та ін [7.8].

1. ***Адитивний метод.*** Перший самонавчальний алгоритм ЕП, запропонований Фогелем [7.8], був адитивним:

$$\sigma_{ij}(t + 1) = \sigma_{ij}(t) + \eta\sigma_{ij}(t)N_{ij}(0, 1), \tag{7.59}$$

де η — швидкість (інтенсивність) навчання.

У першому застосуванні цього підходу $\eta = 1/6$. Якщо $\sigma_{ij}(t) \le 0$, то $\sigma_{ij} = \gamma$, де γ — мала додатна константа (зазвичай $\gamma = 0{,}001$) для забезпечення ненульових девіацій.

Як альтернативу Фогель запропонував алгоритм

$$\sigma_{ij}(t + 1) = \sigma_{ij}(t) + \sqrt{f_\sigma(\sigma_{ij}(t))}N_{ij}(0,1), \tag{7.60}$$

де $f_\sigma(a) = \begin{cases} a, \text{ якщо } a > 0; \\ \gamma, \text{ якщо } a < 0. \end{cases}$

2. ***Мультиплікативний метод.*** К. Деб і Дж. Ванг запропонували метод, у якому

$$\sigma_{ij}(t + 1) = \sigma(0)(\lambda_1 e^{-\lambda_2 \frac{t}{n_t}} + \lambda_3), \tag{7.61}$$

де λ_1, λ_2, λ_3 — регульовані параметри; n_t — максимальна кількість ітерацій.

3. ***Логнормальний метод*** запозичено з еволюційних стратегій

$$\sigma_{ij}(t+1) = \sigma_{ij}(t)e^{(\tau N_i(0,1) + \tau' N_{ij}(0,1))}, \tag{7.62}$$

де $\tau = \frac{1}{\sqrt{2n_x}}$; $\tau' = \frac{1}{\sqrt{2\sqrt{n_x}}}$; n_x — розмірність вектора x.

Алгоритми ЕП, що самонавчаються, виявляють небажану поведінку застою через те, що параметри стратегії швидко сходяться. Внаслідок цього девіації швидко стають надто малими, обмежуючи тим самим пошук рішень. Пошук загасає на деякий час, поки параметри стратегії не збільшаться достатньо завдяки випадковим варіаціям. Одне з вирішень цієї проблеми — накладання нижньої межі на значення σ_{ij}, однак це породжує іншу проблему, коли значення σ_{ij} можна вважати надто малими. Було запропоновано рішення з динамічними нижніми межами:

$$\sigma_{\min}\left(\frac{n_m(t)}{\xi}\right), \tag{7.63}$$

де $\sigma_{\min}$ — нижня межа в момент часу t; $\xi \in [0{,}25; 0{,}45]$ — відносна швидкість і n_m — число успішних послідовних мутацій (тобто число мутацій, що призводили до поліпшення значення Ф-І).

Слід зазначити, що у всіх розглянутих прикладах самонавчання алгоритму ЕП спочатку генерується нащадок, а потім адаптуються параметри. Цим вони відрізняються від еволюційних стратегій, у яких спочатку адаптуються стратегічні параметри, а потім генерується нащадок. Зауважимо, що такий порядок не має суттєвого значення, просто використання нових значень стратегічних параметрів у другому випадку затримується на одне покоління.

Реалізація алгоритмів ЕП. Розглянемо найвідоміші реалізації алгоритмів ЕП.

Класичне ЕП. Відповідно до [7.19], класичне ЕП — це еволюційне програмування з використанням гауссової мутації. Точніше, класичне ЕП використовує логнормальне самонавчання, що задається рівнянням (7.62), і генерує нащадків, використовуючи рівняння (7.50). Для конструювання нової популяції з поточної батьківської популяції та нащадків використовується стратегія елітизму.

Швидке ЕП. У [7.18] адаптували класичне еволюційне програмування для створення швидкого ЕП (*Fast EP*) заміною шуму мутацій з гауссового на Коші згідно з виразом (7.64), де $\nu = 1$. Нащадок генерується за формулою

$$x'_{ij}(t) = x_{ij}(t) + \sigma_{ij}(t)C_{ij}(0, 1), \tag{7.64}$$

у якій застосовується логнормальне самонавчання. Ширші «хвости» розподілу Коші виробляють великі розміри кроків і в результаті прискорюють

збіжність. Аналіз швидкого ЕП показав, що розміри кроків можуть виявитися надто великими для відповідного функціонування, тоді як мутації за гауссовим розподілом мають кращі можливості для тонкого налаштування рішень. Це спонукало до розроблення швидкого алгоритму ЕП. У цьому алгоритмі кожен з батьків генерує двох нащадків, одного — з використанням гауссових мутацій, а другого — мутацій за Коші. Вибирають найкращого нащадка, який змагається з батьками за виживання. Альтернативний підхід полягає в тому, щоб починати пошук, використовуючи мутації за розподілом Коші, а з наближенням до оптимуму переходити на розподіл Гаусса.

Оператори «середнього» забезпечують тонке рішення, використовуючи згортку розподілів Гаусса та Коші.

Експоненційне ЕП. Було запропоновано алгоритм ЕП з подвійним експоненційним розподілом ймовірностей [7.15]. Нащадки генеруються за виразом

$$x'_{ij}(t) = x_{ij}(t) + \sigma_{ij} \frac{1}{\xi} E_{ij}(0, 1), \tag{7.65}$$

де параметр σ_{ij} самонавчається, а варіація розподілу регулюється параметром ξ. Чим менше ξ, тим більша варіація, і навпаки. Щоб забезпечити добрі пошукові властивості на початку роботи, ξ має бути ініціалізоване малим значенням, яке постійно збільшується з часом.

Прискорене ЕП. З метою подальшого поліпшення швидкості збіжності еволюційного програмування було запропоновано прискорене ЕП, яке використовує оператори варіації [7.2, 7.7]:

• оператор напряму для визначення напряму пошуку на основі значень фітнес-функції;

• оператор гауссової мутації, що задається (7.16).

Індивіди мають такий вигляд:

$$x_i(t) = (x_i(t), \rho_{ij}(t); \quad a_i(t)), \tag{7.66}$$

де $\rho_{ij}(t) \in \{-1; 1\}$, $j = 1, \ldots, n_x$, дає напрям пошуку для кожної компоненти j-го індивіда, а a_i — вік індивіда. Вік використовують, щоб форсувати ширші дослідження (пошук), якщо нащадки гірші, ніж їхні батьки.

Генерування нащадків відбувається за два кроки. На першому кроці адаптується параметр віку кожного індивіда і визначаються напрями пошуку (у припущенні задачі мінімізації)

$$a_i(t) = \begin{cases} 1, \text{ якщо } f(x_i(t)) < f(x_i(t-1)); \\ a_i(t-1) + 1 \text{ у протилежному випадку;} \end{cases} \tag{7.67}$$

$$\rho_{ij}(t) = \begin{cases} \text{sign}(x_{ij}(t) - x_{ij}(t-1)), \text{ якщо } f(x_i(t)) < f(x_i(t-1)); \\ \rho_{ij}(t-1) \text{ у протилежному випадку.} \end{cases} \tag{7.68}$$

Якщо Ф-І індивіда поліпшився, то пошук продовжується у напрямі покращення. Якщо Ф-І не поліпшився, то вік отримує прирощення, яке результується у великих розмірах кроку відповідно до виразу [7.1]:

якщо $a_i(t) = 1$, то

$$\sigma_i(t) = \gamma_1 f(x_i(t));$$

$$x'_{ij}(t) = x_{ij}(t) + \rho_{ij}(t)N(0, \sigma_i(t)), \tag{7.69}$$

інакше, якщо $a_i(t) > 1$, то

$$\sigma_i(t) = \gamma_2 f(x_i(t));$$

$$x'_{ij}(t) = x_{ij}(t) + N(0, \sigma_i(t)),$$

де γ_1, γ_2, — додатні константи.

Процедура селекції полягає у змаганні нащадка безпосередньо з батьком з використанням абсолютного показника Ф-І.

ЕП із пам'яттю. Було запропоновано алгоритм ЕП з урахуванням навчання методом зворотного поширення нейронної мережі ВР (див. розд. 1). Обчислюється найкращий індивід поточної популяції $C(t)$:$\hat{y}(t) = x_i(t)$, де $x_i(t)$: $f(x_i(t)) = \min\limits_{l} f(x_l(t))$, $l = \overline{1, \mu}$, який є метою. Поточна похибка між метою та індивідом далі використовується оператором мутацій для покращення пошуку. Для кожного з батьків $x_i(t)$ генерується нащадок:

$$x'_{ij}(t) = x_{ij}(t) + \eta\Delta x_{ij}(t) + \alpha\tilde{x}_{ij}(t), \tag{7.70}$$

де

$$\Delta x_{ij}(t) = (\hat{y}_j(t) - x_{ij}(t))\left|N_{ij}(0, 1)\right|; \tag{7.71}$$

$$\tilde{x}_{ij}(t) = \eta\rho_i \Delta x_{ij}(t - 1) + \alpha\tilde{x}_{ij}(t - 1), \tag{7.72}$$

де $\eta > 0$ — швидкість навчання; $\alpha > 0$ — інтенсивність запам'ятовування;

$$\rho_i(t) = \begin{cases} 1, \text{ якщо } f(x'_i(t)) < f(x_i(t-1)); \\ 0 \text{ у протилежному випадку.} \end{cases} \tag{7.73}$$

Застосування ЕП. Спочатку ЕП було розроблено для еволюції машин з скінченним числом станів (скінченних автоматів) [7.8]. Мета цієї програми полягала у розвитку автоматів для наступного символа в послідовності раніше спостережуваних символів з деякого алфавіту.

Велике число алгоритмів ЕП було розроблено на навчання нейронних мереж зі зворотним поширенням сигналів (тобто ВР). Кожний індивід тут становить нейронну мережу, а кожен його компонент — вагу та поріг. Як показник Ф-І використовували СКП (MSE) або суму квадратів похибки (SSE). У разі задач класифікації можна використати відсоток помилкової класифікації.

Будь-які оператори мутацій, описані вище, можна використати для цього завдання. Найширше ЕП застосовують у таких сферах [7.2]:

- байєсівські мережі;
- робототехніка;
- ігри;
- енергосистеми;
- нейронні мережі;
- проєктування.

7.4. Диференціальна еволюція

Різницеві вектори. Позиції індивідів із популяції у просторі пошуку надають цінну інформацію про ландшафт фітнес-функції. За умови використання доброго випадкового методу ініціалізації з рівномірним розподілом для генерації популяції індивіди дають досить повне уявлення про цілий простір пошуку з відносно великою відстанню між індивідами. З часом у міру розвитку процесу пошуку відстань між індивідами стає меншою і всі індивіди сходяться до одного й того самого рішення. Зауважимо, що початкові відстані між індивідами залежать від розміру популяції. Чим більше індивідів у популяції, тим менша відстань. Відстань між індивідами є вагомим показником розмаїтості поточної популяції та розмірів кроку, які слід вибрати для того, щоб популяція сходилася до однієї точки. Якщо відстані між індивідами великі, то індивіди мають робити кроки великого розміру, щоб досліджувати якнайбільше пошукового простору. З іншого боку, якщо відстані між індивідами малі, то розміри кроків мають бути також малими для дослідження локальних зон. Саме така поведінка досягається за допомогою диференціальної еволюції (ДЕ) [7.17], підрахунком розмірів кроку мутацій як зважених відстаней між випадково обраними індивідами.

Використання вектора різниць має багато переваг. По-перше, інформація про ландшафт фітнес-функцій, надана поточною популяцією, використовується для управління пошуком. По-друге, відповідно до центральної граничної теореми, розміри кроків мутації наближаються до гауссового розподілу за умови, що популяція досить велика для забезпечення великої кількості векторів різниць.

Середнє значення розподілу, утвореного різницевими векторами, завжди дорівнює «0» за умови, що індивіди, використовувані для обчислення різницевих векторів, вибираються поступово з популяції. За умови рівноймовірного вибору індивідів ця властивість випливає з того факту, що вектори різниці $[x_{i1} - x_{i2}]$ та $[x_{i2} - x_{i1}]$ виникають поступово. Нульове середнє підсумкового розміру кроків забезпечує те, що популяція не «страждатиме» від генетичного дрейфу. Слід зазначити, що девіація (стандартне відхилення) цього розподілу визначається величиною різницевих векторів.

Мутації. Оператор мутацій ДЕ виробляє пробний вектор кожного індивіда з поточної популяції через мутації цільового вектора зі зваженою різницею. Цей пробний вектор оператор кросовера використає для генерування нащадка. Для кожного з батьків $x_i(t)$ пробний вектор $u_i(t)$ генерується в такий спосіб [7.17].

Цільовий вектор $x_{i1}(t)$ вибирають з популяції таким, що $i \neq x_1$. Далі з популяції випадково вибирають два індивіди — x_{i2} та x_{i3}, такі, що $i \neq i_1 \neq i_2$ і де $i_2, i_3 \sim U(1, n_\Delta)$. Використовуючи ці індивіди, підраховують пробний вектор, збурюючи цільовий вектор згідно з виразом

$$u_i(t) = x_{i1}(t) + \beta(x_{i2}(t) - x_{i3}(t)), \tag{7.74}$$

де $\beta \in (0, \infty)$ — масштабний коефіцієнт, що регулює диференціальну варіацію.

Кросовер. Оператор кросовера реалізує дискретну рекомбінацію пробного вектора $u_i(t)$ та батьківського вектора $x_i(t)$ для генерування нащадка $x'_i(t)$.

Кросовер реалізується так:

$$x'_{ij}(t) = \begin{cases} u_{ij}(t), \text{ якщо } j \in I; \\ x_{ij}(t) \text{ — у протилежному випадку,} \end{cases} \tag{7.75}$$

де $x_{ij}(t)$ позначає j-й елемент вектора $x_i(t)$, I — множина індексів елементів, які мають бути змінені (тобто множина точок кросовера).

Для визначення множини I можна використати різні методи, з яких найчастіше використовують такі.

1. Біноміальний кросовер. Точки кросовера вибираються випадковим чином із множини можливих позицій $\{1, 2, \ldots, n_x\}$, де n_x — розмірність вектора рішення. У підсумковому вигляді цей процес реалізовано в алгоритмі диференційної еволюції ДЕ. У цьому алгоритмі ймовірність включення кожної компоненти батьківського вектора до нащадка визначається параметром ймовірності кросовера p_r. Чим більше значення p_r, тим більше компонент пробного вектора буде передано нащадку і, відповідно, менше компонент залишиться від батьківського вектора. Якщо $p_r = 1$, усі компоненти нащадка будуть узяті з пробного вектора, а в разі $p_r = 0$ усі компоненти залишаться від батька.

Унаслідок того, що рішення з приводу внесення точки кросовера приймається з імовірністю, може статися, що жодну точку не буде обрано, в цьому разі нащадок просто збігатиметься з батьком. Така проблема виникає за особливо малої розмірності простору пошуку. Щоб забезпечити відмінність принаймні хоч одного елемента від батька, множина точок кросовера в ході ініціалізації має містити випадково обрану точку j^*.

2. Експонентний кросовер. Починаючи з випадково вибраного індексу, оператор експонентного кросовера вибирає послідовність суміжних точок кросовера, розглядаючи список потенційних точок кросовера як циклічний масив. При цьому вибирається принаймні одна точка кросовера і, по-

чинаючи з цього індексу, вибирається наступний, доки не виконається умова $U(0, 1) \geq p_r$ або $|I| = n_x$.

Селекція. Застосовують визначення, які індивіди візьмуть участь в операції мутації для генерування пробного вектора і який із батьків чи нащадків перейде в наступне покоління. Щодо оператора мутації використовують безліч методів селекції.

Випадкову селекцію зазвичай використовують для вибору батьків, у тому числі підраховують вектори різниць. Для більшості алгоритмів ДЕ цільовий вектор вибирають або випадково, або найкращого індивіда.

Для конструювання популяції нового покоління застосовують детерміновану селекцію: нащадок замінює батька, якщо його Ф-І кращий, ніж батьківський, інакше батько переходить у наступне покоління. Тим самим забезпечується, що середній Ф-І покоління не погіршується. Опишемо *загальний алгоритм* ДЕ [7.17].

1. Встановлюємо лічильник поколінь $t = 0$.

2. Ініціалізуємо контрольні параметри β та p_r.

3. Створюємо та ініціалізуємо популяцію $C(0)$ з n_s індивідів, поки не виконається умова зупинки:

   1) для кожного індивіда $x_i(t) \in C(t)$;

   2) оцінюємо Ф-І $f(x_i(t))$;

   3) створюємо пробний вектор $u_i(t)$, використовуючи оператор мутацій;

   4) створюємо нащадка $x'_i(t)$, використовуючи оператор кросовера;

   5) якщо $f(x'_i(t))$ краще, ніж $f(x_i(t))$, то додаємо нащадка $x'_i(t) \in C(t + 1)$ і завершуємо; інакше $x_i(t) \in C(t + 1)$, кінець.

4. Вибираємо індивідів з найкращим Ф-І як рішення.

Параметри керування. На додачу до розміру популяції на характеристики ДЕ впливають два параметри управління: масштабний коефіцієнт β та ймовірність рекомбінації p_r. Вплив цих параметрів розглянуто далі. Розмір популяції, згідно з рівнянням (7.26), безпосередньо впливає на властивості алгоритмів ДЕ. Чим більше індивідів у популяції, тим більше є доступних диференціальних векторів і більше напрямів можна досліджувати. Проте слід враховувати: чим більше індивідів у популяції, тим більші обчислювальні витрати на одне покоління. За допомогою емпіричних досліджень отримано такий орієнтир: $n_s = 10n_x$. Природа мутацій, однак, надає нижню межу числа індивідів $n_s > n_v + 1$, де n_v — число використовуваних диференціалів (різниць). Для n_v різниць потрібно $2\,n_v$ різних індивідів, по два на кожну різницю.

Масштабний коефіцієнт. Масштабний коефіцієнт $\beta \in (0, \infty)$ регулює коефіцієнт посилення варіації різниці $(x_{i2} - x_{i1})$. Чим менше β, тим менший розмір мутацій і тим більше часу потрібно для збіжності алгоритму. Великі значення β покращують пошукові можливості алгоритму, але можуть зумовити перескакування через точку задовільного оптимуму. Тому β має

бути досить малим, щоб мати змогу за допомогою різниць досліджувати вузькі долини (яри), а з іншого боку, досить великим, щоб забезпечити потрібну різноманітність.

Як зазначалося, що більше індивідів у популяції, то менше різниць і менші розміри векторів, то ближче індивіди будуть один до одного. Тому менші розміри кроків можна використовувати для дослідження долин. Збільшення кількості індивідів зменшує потребу у великих кроках мутацій. Емпіричні результати підтверджують, що великі значення як n_s, так і β часто призводять до передчасної збіжності. Якщо $\beta = 0{,}5$, забезпечується добре функціонування алгоритмів ДЕ.

Ймовірність рекомбінації. Імовірність рекомбінації p_r безпосередньо впливає на різноманітність ДЕ. Цей параметр регулює кількість елементів батьків $x_i(t)$, що змінюються. Чим вища ймовірність рекомбінації, тим більше варіабельність вводиться в нову популяцію, тим самим збільшуючи різноманітність та дослідницькі властивості. Збільшення p_r часто призводить до прискорення збіжності, тоді як зменшення p_r підвищує робастність пошуку. Більшість реалізацій стратегій ДЕ вважає параметри управління незмінними. Хоча емпіричні результати показали, що збіжність ДЕ відносно малочутлива до різних значень цих параметрів, характеристики якості алгоритму (точність і робастність) може бути поліпшено знаходженням оптимальних значень цих параметрів для кожної проблеми. Для відшукання оптимальних параметрів було розроблено стратегії ДЕ, що самонавчаються.

Геометрична ілюстрація роботи алгоритму ДЕ. Роботу оператора мутацій, описану в п. 7.3.2, показано на рис. 7.7, *а*. Оптимальне рішення позначено x^* і припущено, що $\beta = 1{,}5$. Оператор кросовера наведено на рис. 7.7, *б*, де показано, що нащадок складається з першого елемента пробного вектора $u_i(t)$ та другого елемента батьківського індивіда $x_i(t)$ [7.1, 7.2, 7.17].

Різновиди алгоритмів ДЕ. Розроблено багато модифікацій базового алгоритму ДЕ. Різні стратегії ДЕ розрізняються методом вибору цільового вектора, кроком векторів різниць і способом визначення точок кросовера. Для формального опису цих модифікацій було розроблено загальну нотацію у літературі ДЕ, а саме ДЕ/*x*/*y*/*z* [7.17]. У цій нотації *x* вказує метод вибору цільового вектора, *y* — число векторів різниць, що використовуються, а *z* — метод кросовера.

Наприклад:

1. DE/rand/1/bin означає випадковий вибір цільового вектора, один вектор різниць $[(x_{2i} - x_{3i})]$ та біноміальний кросовер.

2. DE/best/1/*z*. У цій стратегії як цільовий вектор вибирається найкращий індивід $\hat{x}(t)$ із поточної популяції. В цьому випадку пробний вектор $u_i(t)$ обчислюється так: $u_i(t) = \hat{x}(t) + \beta(x_{2i}(t) - x_{3i}(t))$. Можна використати будь-який метод кросовера.

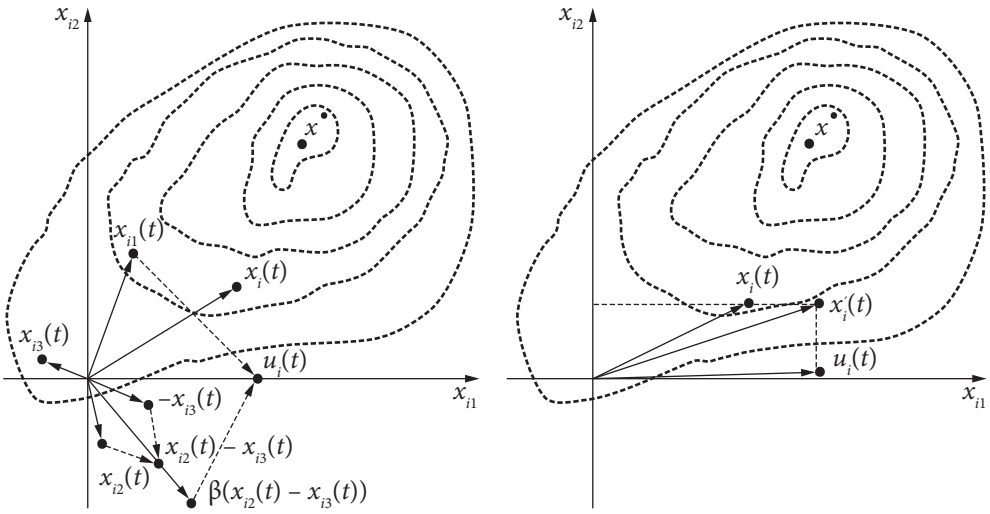


Рис. 7.7. Робота оператора мутацій

3. DE/rand-to-best/n_v/z. Ця стратегія комбінує випадкову стратегію (*rand*) та найкращу стратегію (*best*) для обчислення пробного вектора. При цьому використовується n_v векторів різниць:

$$u_i(t) = \gamma\hat{x}_i(t) + (1 - \gamma)\, x_{i1}(t) + \beta\Sigma_{k=1}^{n_v}(x_{i2,k}(t) - x_{i3,k}(t)), \qquad (7.76)$$

де x_{i1} вибирається випадково, а $\gamma \in [0, 1]$ і регулює «ступінь жадібності» оператора мутацій. Чим ближче γ до «1», тим більш «жадібним» стає процес пошуку, захоплюючи оптимальний вектор. Хороша стратегія — використовувати адаптивний коефіцієнт γ, поклавши $\gamma(0) = 0$, і встановити значення $\gamma(t)$, що монотонно зростає з кожним новим поколінням, яке прямує до «1».

4. DE/current-to-best/1 + n/z. За такої стратегії батько мутує, використовуючи принаймні два вектори різниці. Один вектор різниць обчислюється за найкращим вектором і поточним вектором батька, тоді як інші вектори різниць (числом n_v) обчислюються з використанням випадково вибраних векторів:

$$u_i(t) = x_i(t) + \beta(\hat{x}_i(t) - x_i(t)) + \beta\Sigma_{k=1}^{n_v}(x_{i1,k}(t) - x_{i2,k}(t)). \qquad (7.77)$$

Емпіричні дослідження показали, що алгоритм типу DE/rand/1/bin підтримує добру різноманітність, тоді як алгоритм DE/current-to-best/1 + n/z виявляє хорошу збіжність [7.38]. Відповідно до цього висновку було розроблено алгоритм ДЕ, який динамічно перемикається між цими стратегіями. Кожній із цих стратегій приписується певна ймовірність. Якщо $p_{s,1}$ — ймовірність застосування стратегії DE/rand/1/bin, то 1 – $p_{s,1}$ — імовірність за-

стосування стратегії DE/current-to-best/2/bin. Тоді

$$p_{s,1} = \frac{n_{s,1}\,(n_{s,2} + n_{f,2})}{n_{s,1}\,(n_{s,2} + n_{f,2}) + n_{s,2}(n_{s,1} + n_{f,1})}, \tag{7.78}$$

де $n_{s,1}$, $n_{s,2}$ — число нащадків, які вижили і пережили наступне покоління, для стратегії DE/rand/1/bin та DE/current-to-best/2/bin відповідно, а $n_{f,1}$, $n_{f,2}$ — число відкинутих нащадків для цих стратегій.

7.5. Самонавчання ДЕ

Як зазначалося, характеристики алгоритмів ДЕ можуть бути значно поліпшені у результаті оптимізації параметрів управління. Такі стратегії потребують додаткового процесу пошуку оптимальних значень параметрів управління для кожної окремої проблеми, і це зумовлює додаткові витрати часу.

Розглянемо такі стратегії.

Динамічні характеристики. Один із перших підходів для динамічної зміни параметрів управління в ДЕ був запропонований Цзянь-Шунь Чангом (Jian-Shung Chang) і Вень-Хунгом Ду (Wen-Hung Dew), в якому ймовірність рекомбінації лінійно зменшується від 1 до 0,7, а коефіцієнт масштабу лінійно збільшується від 0,3 до 0,5:

$$p_r(t) = p_r(t-1) - \frac{p_r(0) - 0{,}7}{n_t}; \tag{7.79}$$

$$\beta(t) = \beta(t-1) - \frac{0{,}5 - \beta(0)}{n_t}, \tag{7.80}$$

де $p_r(0) = 1{,}0$; $\beta(0) = 0{,}3$; n_t — максимальна кількість ітерацій.

Було також запропоновано підхід, у якому новий масштабний коефіцієнт вибирається за кожного застосування оператора мутацій. Масштабний коефіцієнт β вибирають випадково з нормального розподілу $N(0,1)$.

Багатокритеріальна оптимізація. За багатокритеріальної оптимізації потрібно, щоб декілька конфліктних цілей було одночасно оптимізовано. Було виконано ряд адаптацій алгоритмів ДЕ на вирішення завдань з множиною критеріїв, більшість із яких використовує концепцію домінування. Підходи до багатокритеріальної ДЕ охоплюють:

1) перетворення проблеми на мінімаксну;

2) методи зваженого агрегування; методи, що ґрунтуються на популяціях, як-от векторні оцінки ДЕ (*VEDE — Vector Evaluated Differential Evolution* (*Векторна оцінка диференційної еволюції*)); векторні оцінки за допомогою ГА (*VEGA — Vector Evaluated Genetic Algorithm* (*Векторна оцінка за допомогою генетичного алгоритму*)). Якщо потрібно мінімізувати k критеріїв, то використовується k підпопуляцій, кожна з яких оптимізує один із критеріїв. Такі

підпопуляції організовуються у кільцеву топологію. На кожній ітерації перед застосуванням операторів репродукції ДЕ найкращий індивід популяції c_k мігрує у популяцію c_{k+1}, а відповідний найкращий індивід c_{k+1} мігрує в c_k і використовується для формування пробних векторів цієї популяції;

3) методи, що ґрунтуються на функції Парето, які змінюють оператори ДЕ, щоб включити концепцію домінування.

Мутації. Запропоновано підхід, у якому мутації застосовуються лише до недомінованих рішень. Обчислюють диференціал як різницю між випадково обраними індивідом $x_{i_1}(t)$ та вектором $x_{i_2}(t)$, який домінує $x_{i_3}(t)$, тобто $x_{i_2}(t) \geq x_{i_1}(t)$. Якщо даний індивід не домінується жодним іншим індивідом з поточного покоління, то різниця покладається такою, що дорівнює 0.

Селекція. Можна використовувати ідеї з ГА для домінованого сортування, у яких недомінованe сортування та ранжування застосовуються до батьків і нащадка. Наступну популяцію вибирають з переважанням індивідів із вищим рангом.

7.6. Знаходження найкоротшого шляху на графі на основі ГА

Постановка задачі. У великих розподілених мережах, як-от *Internet* і мережі мобільного зв'язку, маршрутизація становить найважливішу проблему, від вирішення якої залежать характеристики функціонування мережі. Ідеальний алгоритм маршрутизації має бути спрямований на знаходження оптимального шляху передання пакетів за певний заданий час, щоб забезпечити бажану якість обслуговування. Є кілька алгоритмів пошуку найкоротших шляхів (*SP*), наприклад алгоритми Дейкстри (Edsger W. Dijkstra) [7.41] та Беллмана — Форда (Richard Bellman; Lester R. Ford Sr.). Ці алгоритми видають рішення за поліноміальний час, тому їх ефективно застосовують у мережах з фіксованою структурою в дротових та бездротових мережах. Однак вони мають неприпустимо високу обчислювальну складність у реальному часі, зокрема в мережах мобільного зв'язку з топологією, що швидко змінюється.

Розглянемо мобільні мережі як цільові системи. Оскільки всі вузли такої мережі спільно підтримують її зв'язність без будь-якої фіксованої структури, можливі динамічні зміни в топології мережі. Необхідно визначити оптимальний (найкоротший) маршрут за дуже короткий час (кілька мілісекунд), щоб підтримувати показники якості (*QoS*), пов'язані з тимчасовими обмеженнями, такими як аудіо-, відео- та телеконференції. Зазначені алгоритми не задовольняють цю вимогу. Як правило, зв'язки мережі мають ваги, що описують пропускну здатність каналу, затримку в черзі або відмову каналу в мережі. Тому проблему *SP*-маршрутизації можна розгля-

дати як проблему знаходження найкоротшого шляху мінімальної вартості між вузлом-джерелом (S) та вузлом призначення (D).

Сучасні підходи, засновані на ГА. Запропоновано низку ГА для вирішення проблеми *SP*. Зокрема, алгоритм Мунемото (Katsunori Munemoto) [7.36] практично застосовується для провідних та бездротових мереж. Він використовує хромосоми змінної довжини для кодування задачі. Точками кросовера є позиції, в яких ідентичні гени (тобто вузли) розміщені в однакових позиціях в обох хромосомах. Це призводить до ситуації, коли лише кілька позицій точок кросовера придатні для отримання допустимих рішень. Кандидати точок кросовера називаються потенційними місцями кросовера. Реальна точка кросовера вибирається випадково. На фазі мутації ген (тобто вузол мутації) вибирається випадково з хромосоми. Інший ген (вузол) вибирається випадково з хромосоми, що безпосередньо пов'язана з вузлом мутації, і генерується мутуванням хромосоми за допомогою комбінування її частин. Варто зауважити, що один частковий маршрут стосується найкоротшого шляху від вихідного до обраного вузла, а другий — від обраного вузла до вузла призначення. Однак цей алгоритм потребує відносно великого розміру популяції для ефективного пошуку оптимального рішення через обмеження, пов'язані з механізмом кросовера. Крім того, його ефективність знижується для мереж великої розмірності, оскільки він використовує алгоритм Дейкстри.

Інагакі [7.36] запропонував інший алгоритм, який використовує фіксовану довжину хромосом. Хромосоми в алгоритмі — послідовності цілих чисел, кожний ген слугує номером вузла (вузол *Id*), який вибирається випадково з множини вузлів, пов'язаних з вузлом, що відповідає номеру його позиції. На фазі кросовера один з генів одного з двох батьків вибирається на позиції стартового вузла батька і ставиться в ту саму позицію нащадка. Далі один із генів вибирається випадково на позицію з номером раніше обраного гена. Цей процес триває доти, доки не буде досягнуто вузла призначення. Описаний алгоритм потребує великої популяції для досягнення високої якості рішень через недосконалий механізм кросовера.

Математична модель задачі. Розглянуті топології мереж задаються орієнтованим графом $G = (V, A)$, де V — множина з $|V|$ вузлів; A — множина зв'язків (дуг або ребер). З кожним зв'язком (i, j) зв'язується c_{ij} — її вартість. Вартість зв'язків задається матрицею $C = ||c_{ij}||$, де $i, j = \overline{1, n}$; $i \neq j$; c_{ij} — вартість (час затримки) передання пакета по ЛЗ (i, j). Джерело позначається S, а адресат — D. Кожний зв'язок має індикатор J_{ij}, який відіграє роль хромосомної карти (або маски), надаючи інформацію: чи внесено зв'язок (i, j) у відповідний маршрут, чи ні. Це можна зобразити так:

$$J_{ij} = \begin{cases} 1, \text{ якщо зв'язок } (i, j) \text{ є в маршруті;} \\ 0 \text{ — інакше.} \end{cases} \qquad (7.81)$$

Очевидно, що діагональні елементи J мають дорівнювати «0». Використовуючи наведені визначення, проблему знаходження SP можна сформулювати як проблему комбінаторної оптимізації, тобто мінімізації цільової функції

$$\Sigma_{i=s}^{D}\Sigma_{\substack{j=s\\ j\neq i}}^{D} C_{ij}J_{ij} \tag{7.82}$$

за умов

$$\Sigma_{\substack{j=s\\ j\neq i}}^{D} J_{ij} - \Sigma_{\substack{j=s\\ j\neq i}}^{D} J_{ij} = \begin{cases} 1, \text{ якщо } i = S; \\ -1, \text{ якщо } i = D; \\ 0 \text{ — інакше} \end{cases}$$

і

$$\Sigma_{\substack{j=s\\ j\neq i}}^{D} J_{ij} = \begin{cases} \leq 1, \text{ якщо } i \neq D; \\ = 0, \text{ якщо } i = D; \end{cases} \quad J_{ij} \in \{0, 1\}. \tag{7.83}$$

Пропонований генетичний алгоритм розв'язання задачі. У роботі [7.36] Цзянь-Шунь Чангом запропоновано алгоритм, вільний від недоліків зазначених вище алгоритмів Мунемото та Інагакі. Хромосоми в пропонованому ГА складаються з послідовності цілих чисел, які становлять *ID*-вузлів на шляху. Кожна позиція (*locus*) хромосоми є порядковим номером вузла на маршруті.

Ген першої позиції завжди зарезервований за джерелом. Довжина хромосоми — змінна, але вона не повинна перевищувати максимального числа вузлів $n = |V|$. Хромосома (тобто шлях) кодує завдання, поміщаючи номери *ID*-вузлів від джерела до адресата на основі бази даних топологічної інформації (тобто таблиць маршрутизації). Цю інформацію можна отримувати і оновлювати в реальному часі, використовуючи протоколи маршрутизації *OSPF* та ін.

Приклад кодування хромосоми від вузла S до вузла D показано на рис. 7.8 та в табл. 7.1. Ген першого локусу кодує вузол джерела, ген другого локусу вибирається випадково або евристично з вузлів, пов'язаних із вузлом джерела. Вибраний вузол видаляють з бази даних топологічної інформації, щоб перешкодити вузлу бути обраним двічі. Цей процес триває до досягнення вузла призначення.

Ініціалізація популяції. Евристична ініціалізація може бути вигідною для *SP*-проблеми, оскільки топологічну інформацію вже зібрано на початок роботи алгоритму. Однак евристична ініціалізація може підвищити складність алгоритму і призвести до передчасної збіжності. Отже, краще використовувати випадкову ініціалізацію, тому початкову популяцію визначають випадково, використовуючи вищеописаний спосіб кодування хромосом.

Можлива ситуація, коли алгоритм досягає вузла, для якого всіх можливих сусідів уже було вибрано раніше. В цьому разі дефектна хромосома реініціалізується.

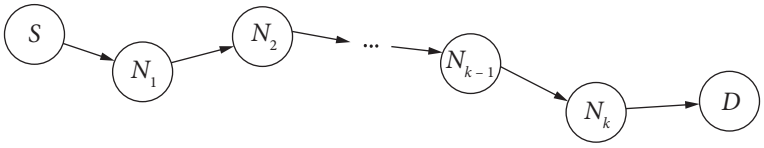


Рис. 7.8. Приклад кодування хромосоми від вузла *S* до вузла *D*

Фітнес-функцію (Ф-Ф), що визначає ефективність обраного шляху, знаходять за формулою

$$F_i = \left[\sum_{j=1}^{n_i-1} \cdot c_{g_i(j)\cdot g_i(j+1)}\right]^{-1}, \qquad (7.84)$$

де F_i — Ф-Ф i-ї хромосоми, n_i — її довжина, $g_i(j)$ позначає ген (тобто вузол) j-го локусу в i-й хромосомі.

Отже Ф-Ф F_i є цільовою функцією ГА. Вона має бути оптимізована (мінімізована).

Селекція. Серед схем селекції турнірна селекція без заміщення є найбільш перспективною, оскільки зберігає рівень шумів настільки низьким, наскільки можливо.

Загалом тиск турнірної селекції зростає з розміром турніру , а це може призвести до передчасної збіжності. Тож у запропонованому ГА використовується парна ($S = 2$) турнірна селекція.

Вилучають дві хромосоми і відбирають найкращу з них. Однак зауважимо, що та сама хромосома може бути обрана двічі як батько.

Кросовер використовує поточні рішення для знаходження найкращих. Фізично кросовер у *SP*-проблемі маршрутизації відіграє роль обміну частковими маршрутами для двох вибраних хромосом, отже, в результаті залишається лише один маршрут. Це визначає вибір одноточкового кросовера як придатний механізм для ГА. Один частковий маршрут з'єднує вузол-джерело з проміжним вузлом, тоді як другий сполучає проміжний вузол з адресатом (*D*).

На рис. 7.9 зображено механізм кросовера, однак не такий, як звичайного одноточкового кросовера. У пропонованій схемі дві хромосоми, вибрані для кросовера, повинні мати принаймні один загальний ген (тобто вузол), за винятком джерела та адресата, але не обов'язково, щоб вони знаходилися в однаковій позиції (локусі). Тобто кросовер не залежить від

Таблиця 7.1

Приклад кодування хромосоми від вузла *S* до вузла *D*

Локус	1	2	3	…	n–2	n–1	n
Хромосома	S	N_1	N_2	…	N_{k-1}	N_k	D

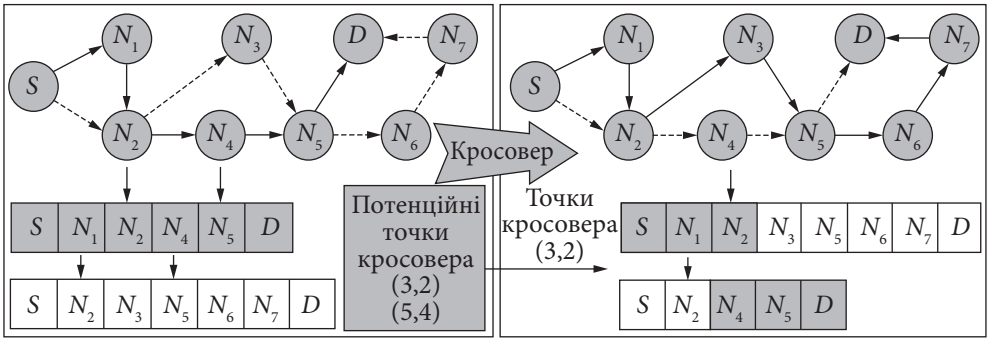


Рис. 7.9. Механізм кросовера, що не залежить від позицій вузлів у маршруті

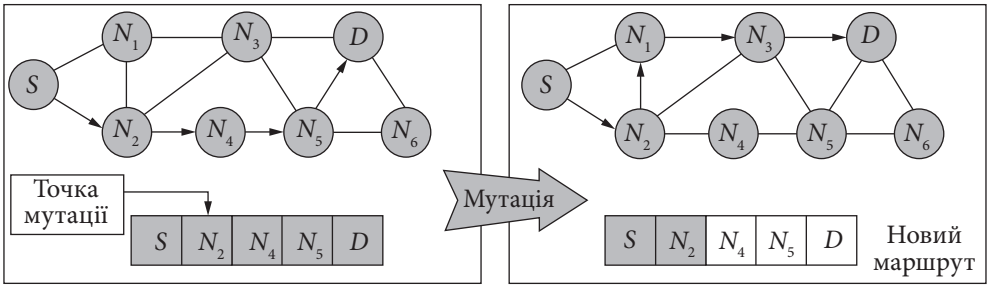


Рис. 7.10. Загальна процедура операції мутації

позицій вузлів у маршруті. Спочатку формуються пари вузлів (N_2, N_5), які є спільними для двох хромосом без відповідності їхніх позицій. Такі пари називаються потенційними точками кросовера.

Далі одна пара (3,2) (N_2, N_5) вибирається випадково, і локус кожного вузла стає точкою кросовера кожної хромосоми. Ці точки кросовера різних хромосом можуть бути різними, що протилежно схемі, прийнятій в алгоритмі Мунемото. Далі відбувається обмін частковими маршрутами між хромосомами-батьками в точках кросовера, що призводить до появи двох нових маршрутів. Можливе утворення петель у результаті кросовера. Тому слід передбачити простий контрзахід, для того щоб уникнути зниження швидкості збіжності та якості рішень. Такими контрзаходами можуть бути ремонт (*repair*) і штрафні функції, які описано далі.

Мутації. Населення зазнає мутацій через дійсну зміну одного з генів хромосом кандидатів, тим самим уникаючи локальних оптимумів. Практично вона (мутація) генерує альтернативний частковий маршрут із вузла мутації до адресата пропонованого ГА.

Загальну процедуру операції мутації наведено на рис. 7.10, з якого легко побачити, що для того, щоб виконати мутацію, ген — точка мутації (тобто вузол N_2) спочатку вибирається випадково з обраної хромосоми, а потім

вибирається випадково один із вузлів, безпосередньо пов'язаних з точкою мутації, як перший вузол альтернативного маршруту. Частина процедури, що залишилася, аналогічна описаній вище.

Однак вузли, вже внесені у верхній частковий маршрут, мають бути видалені з бази даних, для того щоб не вносити двічі один і той самий вузол у новий маршрут. Верхній частковий маршрут становить частину колишнього маршруту після мутації. Це часткова хромосома, яка тягнеться від першого гена до проміжного гена в точці мутації.

Функція ремонту. Як зазначалося раніше, кросовер може генерувати недопустимі хромосоми, що порушують обмеження, генеруючи цикли маршрутів. Слід зауважити, що жодна з хромосом початкової популяції або після мутацій не є недопустимою, оскільки щойно вузол було обрано, він виключається з вузлів-кандидатів, що формують залишок шляху.

Через істотні недоліки методу штрафних функцій, зокрема труднощі підбирання відповідної штрафної функції, у запропонованому ГА використовується метод ремонту. До того ж механізм, який вилучає летальні гени, що формують цикли в маршруті, не потребує значного збільшення обчислювальних витрат.

Пропонований алгоритм ремонту показано на рис. 7.11. Один із нащадків, отриманих після кросовера, стає недопустимим, оскільки новий маршрут містить цикл $N_2 \rightarrow N_3 \rightarrow N_1 \rightarrow N_2$. Функція ремонту виявляє цикл за допомогою простої процедури. Ця функція має обчислювальну складність n (n — довжина хромосоми). Після виявлення «летальні» гени (утворюють цикл) видаляються — це вузли $\{N_3, N_1, N_2\}$ у цьому прикладі.

Експериментальні дослідження. Експерименти з моделювання містили детерміновану зважену мережу з 20 вузлами, зображену на рис. 7.12 [7.36]. Товстою лінією вказано оптимальний маршрут. На рис. 7.12 показано найкоротші шляхи, знайдені за відповідними алгоритмами. Як бачимо, маршрут, знайдений вище описаним ГА, збігається з оптимальним, визначеним за допомогою алгоритму Дейкстри. Використовуючи алгоритми Мунемото та Інагакі, визначили субоптимальні шляхи.

Таблиця 7.2

Порівняння показників якості алгоритмів

Алгоритм	Частка невдач	Число оцінок фітнес-функції	
		μ_{eval}	σ_{eval}
Інагакі	0,2438	810,5440	181,9882
Мунемото	0,229	251,9105	58,9393
Пропонований	0,2014	122,3158	31,6627

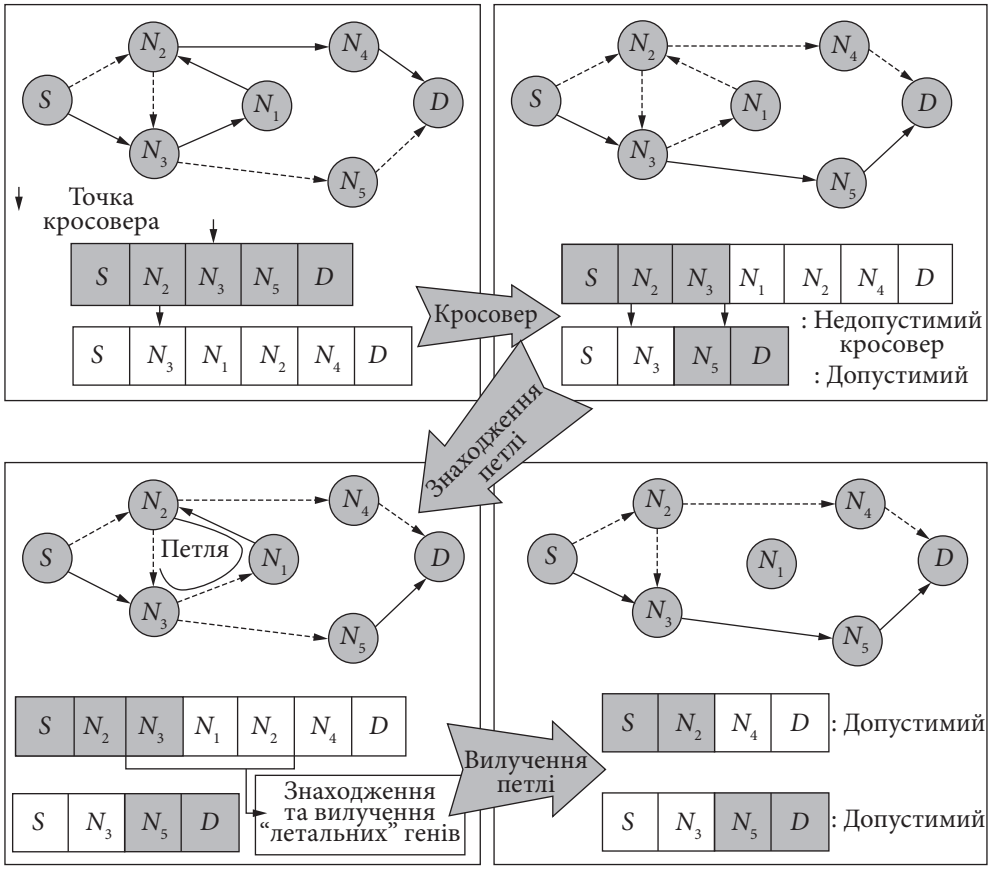


Рис. 7.11. Алгоритм ремонту

На рис. 7.13 наведено залежність цільової функції від числа ітерацій для досліджених алгоритмів. Як бачимо, запропонований ГА виявляє найбільшу швидкість збіжності. У цьому дослідженні всі ГА експериментально використовували однаковий розмір популяції.

У наступних експериментах вибирали мережі з кількістю вузлів 15—50 і випадково заданими вагами зв'язків. Спочатку досліджено якість отримуваних рішень для кожного ГА. Якість рішень оцінювалася як відсоток кількості дослідів, у яких було знайдено глобально оптимальне рішення.

Частка невдач є оберненою до частки успіхів. Її значення асимптотично дорівнює ймовірності того, що знайдений маршрут не є оптимальним. Загалом було розглянуто 1000 випадкових топологій мереж у кожному разі.

Підсумкові результати (порівняння показників якості алгоритмів) наведено у табл. 7.2. Запропонований алгоритм досягає частки невдач, що дорівнює 0,2014 (частка успіху 79,86 %), при цьому розмір популяції до-

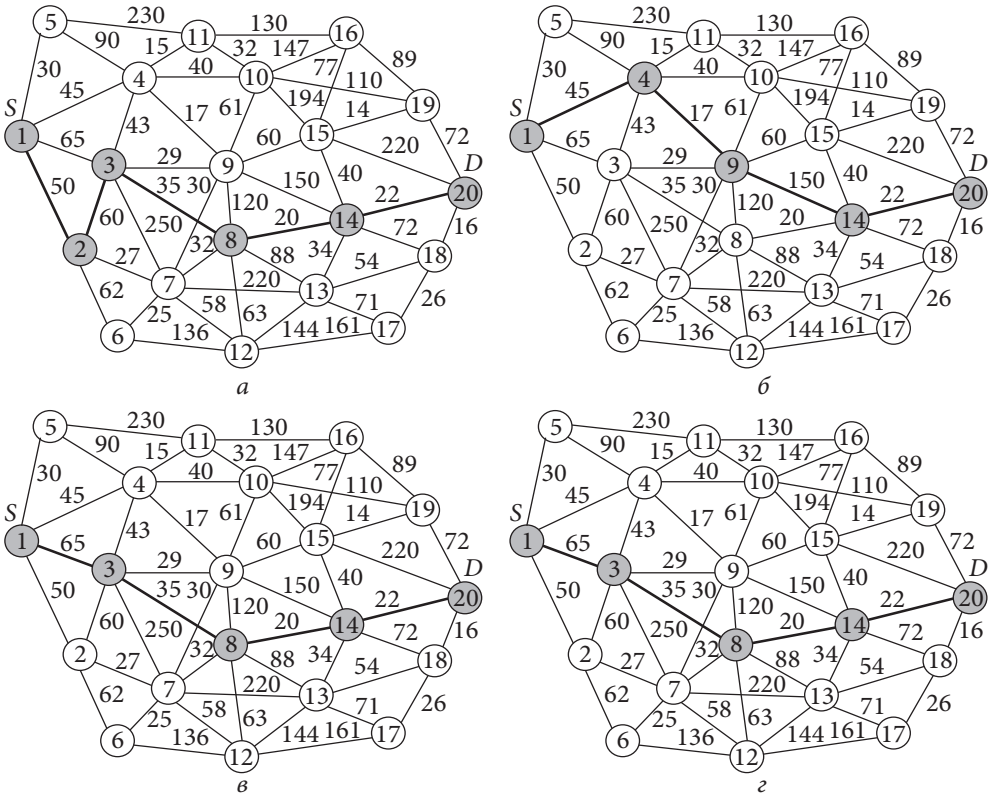


Рис. 7.12. Детермінована зважена мережа з 20 вузлами: *а* — результати алгоритму Мунемото (загальна вартість шляху — 187); *б* — результати алгоритму Інагакі (загальна вартість шляху — 234); *в* — результати запропонованого алгоритму (загальна вартість шляху — 142); *г* — приклад шляху з оптимальним маршрутом (загальна вартість шляху — 142) [7.36]

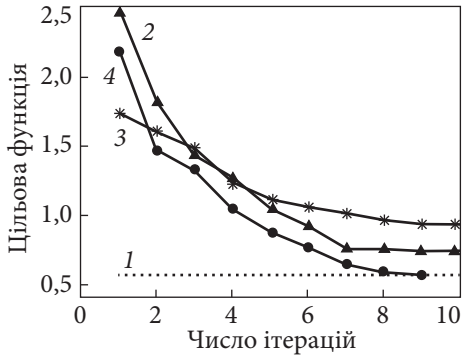


Рис. 7.13. Залежність цільової функції від числа ітерацій: *1* — алгоритм Дейкстри; *2* — алгоритм Мунемото; *3* — алгоритм Інагакі; *4* — запропонований алгоритм

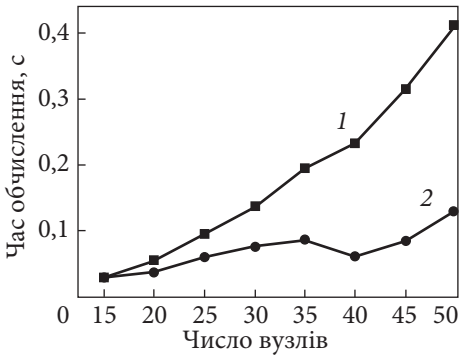


Рис. 7.14. Залежність прямого часу обчислень від розмірності мереж: *1* — алгоритм Дейкстри; *2* — запропонований алгоритм

рівнював числу вузлів у мережах. Зауважимо, що стандартне відхилення частки невдач для запропонованого алгоритму виявилося меншим, ніж для альтернативних алгоритмів Мунемото та Інагакі.

Щоб порівняти характеристики збіжності запропонованого ГА з алгоритмом Дейкстри, було визначено залежність прямого часу обчислень для зазначених алгоритмів від розмірності мереж. Відповідні результати наведено на рис. 7.14. Середній час обчислень за алгоритмом Дейкстри становив 0,14 с, тоді як за запропонованим — усього 0,067 с. Отже, час обчислень запропонованого ГА суттєво не зростає зі збільшенням розмірності мережі, як це спостерігається в алгоритмі Дейкстри.

Висновки

Цей розділ присвячено докладному розгляду та аналізу еволюційних методів, а саме — генетичних алгоритмів та еволюційного програмування в задачах обчислювального інтелекту.

Викладено основи теорії генетичних алгоритмів. Детально описано основні оператори ГА: кросовер, мутації та селекцію. Розглянуто способи подання індивідів — бінарне та з плаваючою точкою. Показано способи реалізації кросовера для бінарного подання і для плаваючої точки та проведено їх аналіз. Далі розглянуто оператори мутації та їхні властивості. Детально описано основні способи кросовера. Дано класифікацію основних алгоритмів ГА: детерміністичних, адаптивних та самопристосовних, проаналізовано їхні властивості та можливості застосування. Наведено види оператора селекції ГА та їхні властивості.

Розглянуто острівні генетичні алгоритми, їхні властивості та способи реалізації операторів ГА. Проаналізовано можливості практичного застосування ГА в задачах ШІ. Наведено приклад практичного застосування генетичних алгоритмів у важливій задачі синтезу структури розподілених комп'ютерних мереж з комунікаційною технологією *MPLS* за критерієм вартості за обмежень на показники якості передання та аналіз ефекту від його застосування.

Далі докладно розглянуто еволюційні методи в задачах ОІ — еволюційне програмування. Описано основні оператори ЕП — мутації та селекцію. Розглянуто основні види операторів мутації: рівномірний, гауссів, Коші, Леві та експоненціальний, проаналізовано їхні властивості та ефективність застосування для реалізації мутацій і пошуку оптимальних рішень. Показано, що на початку процесу пошуку оптимальних рішень доцільно використовувати розподіл Коші, який має широкі «хвости», тоді як на наступній фазі застосувань знайдених рішень краще використовувати гауссів розподіл, менш волатильний з вузькими «хвостами». Крім того, можливе

використання розподілу Леві, який є комбінацією двох розподілів — гауссового та Коші.

Далі наведено багато прикладів реалізації операторів мутацій та генерування нащадків, проаналізовано їхні властивості. Описано стратегічні параметри ЕП та способи їх налаштування. Дано класифікацію основних типів алгоритмів ЕП, визначено їх властивості та галузі ефективного застосування.

Розглянуто цікавий напрям в ЕП — диференціальну еволюцію, в якій крім базових операторів мутації та селекції використовується також третій оператор ГА — кросовер. Диференціальна еволюція має ширші можливості порівняно з класичним ЕП і є більш ефективною для вирішення практичних завдань. Наведено приклад реалізації ЕП. Наприкінці розділу наводиться приклад застосування ГА в популярній задачі пошуку оптимальних шляхів на графі. Виконано дослідження ефективності ГА, проаналізовано отримані рішення та проведено порівняння цього алгоритму з класичним алгоритмом Дейкстри для різних графів з різною кількістю вузлів за точністю та кількістю операцій.

СПИСОК ЛІТЕРАТУРИ

7.1. Keller J. M., Liu D., Fogel D. B. Fundamentals of Computational Intelligence: Neural Networks, Fuzzy Systems, and Evolutionary Computation. Wiley-IEEE Press, 2016. 378 p.

7.2. Zgurovsky M., Zaychenko Yu. Fundamentals of computational intelligence — system approach. Springer, 2016. 375 p.

7.3. Holland J.H. Adaptation in Natural and Artificial Systems. Ann Arbor: University of Michigan Press, 1975. 232 p.

7.4. Holland J.H. ECHO: Explorations of Evolution in a Miniature World. Proc. Second Conf. Artificial Life. 1990. URL: https://gwern.net/doc/ai/1992-langton-artificiallife-2.pdf

7.5. Goldberg D.E., Wang L. Adaptive Niching via Coevolutionary Sharing. Genetic Algorithms and Evolution Strategy in Engineering and Computer Science. Chichester: John Wiley and Sons, 1998. P. 21—38. URL: https://citeseerx.ist.psu.edu/document?doi=f872d929d49a2a5837a1361e543c41246532422d

7.6. Gordon V.S., Whitley D. Serial and Parallel Genetic Algorithms as Function Optimizers. *Proc. Fifth Intern. Conf. Genet. Algorithms*. 1993. P. 177—183. URL: https://www.researchgate.net/publication/2525200_Serial_and_Parallel_Genetic_Algorithms_as_Function_Optimizers

7.7. Hinterding R. Gaussian Mutation and Self-Adaption for Numeric Genetic Algorithms. *Proc. Intern. Conf. Evolutionary Comp.* 1995. No. 1. P. 384. URL: https://ieeexplore.ieee.org/xpl/conhome/3507/proceeding

7.8. Fogel L.J., Owens A., Walsh M. Artificial Intelligence through Simulated Evolution. John Wiley & Sons, 1966. 170 p.

7.9. Fraser A.S. Simulation of Genetic Systems by Automatic Digital Computers I: Introduction. *Australian J. Biolog. Sci.* 1957. No. 10. P. 484—491.

7.10. Gen M., Cheng R. Genetic Algorithms and Engineering Design. John Wiley & Sons, Inc., 1996. 410 p.

7.11. Fogel D. Review of «Computational intelligence: imitating life». *IEEE Trans. Neural Networks.* 1995. No. 6. P. 1562—1565.

7.12. Deb K. Multi-Objective Optimization Using Evolutionary Algorithms: An Introduction. Department of Mechanical Engineering Indian Institute of Technology Kanpur, India. February 10, 2011. P. 1—24. URL: https://www.egr.msu.edu/~kdeb/papers/k2011003.pdf

7.13. Miller B.L., Shaw M.J. Genetic Algorithms with Dynamic Niche Sharing for Multimodal Function Optimization. *Intern. Conf. Evolutionary-Comp.* 1996. P. 786—791. URL: https://citeseerx.ist.psu.edu/document?doi=8e78a8725eecbfb0d26c9ebc313adbd24d847470

7.14. Narihisa H., Taniguchi T., Thuda M., Katayama K. Efficiency of Parallel Exponential Evolutionary Programming. *Proc. Intern. Conf. Workshop Parallel Processing.* 2005. P. 588—595. URL: https://www.researchgate.net/publication/4162547_Efficiency_of_parallel_exponential_evolutionary_programming

7.15. Ono I., Kobayashi S. A Real-Coded Genetic Algorithm for Function Optimization using Unimodal Normal Distribution Crossover. *Proc. Seventh Intern. Conf. Genet. Algorithms*. 1997. P. 246—253. URL: https://dl.acm.org/doi/10.5555/2933923.2933970

7.16. Price K.V., Storn R.M., Lampinen J.A. Differential Evolution: A Practical Approach to Global Optimization. Springer, 2005. 558 p.

7.17. Storn R., Price K. Differential Evolution: A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. *J. Global Optim.* 1997. Vol. 11(4). P. 431—433.

7.18. Yao X., Liu Y. Fast Evolution Strategies. Evolutionary Program. VI. Berlin: Springer, 1997. P. 151—161.

7.19. Beyer H.-G., Schwefel H.-P. Evolution strategies: A comprehensive introduction. *Natural Computing. I.* Dordrecht: Kluwer Academic Publishers, 2002. P. 3—52.

7.20. Yuryevich J., Wong K.P. Evolutionary Programming Based Optimal Power Flow Algorithm. *IEEE Trans. Power Syst.* 1999. Vol. 14 (4). P. 1245—1250. URL: https://ieeexplore.ieee.org/document/801880/

7.21. Wei C., Yao S., He Z. A Modified Evolutionary Programming. *Proc. IEEE Intern. Conf. Evolutionary Comp.* 1996. P. 135—138. URL: https://ieeexplore.ieee.org/document/542326

7.22. El-Alfy, E. S. M. MPLS Network Topology Design Using Genetic Algorithms. *Proceedings of the ACS/IEEE International Conference on Computer Systems and Applications (AICCSA).* IEEE, 2006. P. 1058—1064. URL: https://faculty.kfupm.edu.sa/ics/alfy/files/publications/pdf/aiccsa06-mpls.pdf?utm_source=chatgpt.com

7.23. Zilinskas A., Zimmermann H.-J. (Eds.) Fuzzy Logic and Intelligent Systems. Springer, 2001. 304 p.

7.24. El-Alfy, E. S. M. Applications of Genetic Algorithms to Optimal Multilevel Design of MPLS-Based Networks. *Computer Communications*. 2007. Vol. 30. Iss. 9. P. 2010—2020. URL: https://www.researchgate.net/publication/222564980_Applications_of_genetic_algorithms_to_optimal_multilevel_design_of_MPLS-based_networks

7.25. Зайченко Ю.П. Основи проектування інтелектуальних систем. Київ: Вид. дім «Слово», 2004. 352 с.

7.26. Marks R.J. II, Dembski W.A., Ewert W. Introduction to Evolutionary Informatics. World Scientific Publishing Company, 2017. 336 p.

7.27. Natita W., Wiboonsak W., Dusadee. S. Appropriate Learning Rate and Neighborhood Function of Self-organizing Map (SOM) for Specific Humidity Pattern Classification over Southern Thailand. *Intern. International Journal of Modeling and Optimization (IJMO).* 2016. P. 61—65.

7.28. Breard G.T. Evaluating Self-Organizing Map Quality Measures as Convergence Criteria. Open Access Master's Theses. 2017. Paper 1033. URL: https://digitalcommons.uri.edu/theses/1033

7.29. Michalewicz Z. Genetic Algorithms + Data Structures = Evolution Programs. Springer-Verlag, 1996. 387 p. URL: https://link.springer.com/book/10.1007/978-3-662-03315-9

7.30. Eshelman L.J., Schaffer J.D. Real-Coded Genetic Algorithms and Interval Schemata. In: D. Whitley (Ed.), Foundations of Genetic Algorithms 2, Morgan Kaufmann, 1993. P. 187—202. URL: https://doi.org/10.1016/B978-0-08-094832-4.50018-0

7.31. Deb K., Joshi D., Anand A. Real-Coded Evolutionary Algorithms with Parent-Centric Recombination. *Proceedings of the 2002 Congress on Evolutionary Computation (CEC 2002).* IEEE, 2002. P. 61—66. URL: https://www.egr.msu.edu/~kdeb/papers/real_codedpaper.pdf

7.32. Hinterding R., Michalewicz Z., Peachey T. C. Self-Adaptive Genetic Algorithm for Numeric Functions. *IEEE International Conference on Evolutionary Computation.* 1995. P. 420—429.

7.33. Jones T. Crossover, Macromutation, and Population-Based Search. *Proceedings of the Sixth International Conference on Genetic Algorithms.* 1995. P. 73—80. URL: https://sfi-edu.s3.amazonaws.com/sfi-edu/production/uploads/sfi-com/dev/uploads/filer/11/84/11842c44-c934-4b9d-86c2-d177b7cfb925/95-02-024.pdf

7.34. Jong K.A., Potter M.A. Evolving Complex Structures via Cooperative Coevolution. *Proceedings of the Fourth Annual Conference on Evolutionary Programming.* MIT Press, 1995. P. 307—317. URL: https://direct.mit.edu/books/edited-volume/4503/chapter/192885/Evolving-Complex-Structures-via-Cooperative

7.35. Fogarty T.C. Varying the Probability of Mutation in the Genetic Algorithm. *Proc. of the Third Intern. Conf. Genet. Algorithms.* 1989. P. 104—109. URL: https://link.springer.com/chapter/10.1007/3-540-58483-8_9

7.36. Chang C.S. Advances in Evolutionary Algorithms: Theory, Design and Practice. Springer-Verlag, 2007. 166 p.

7.37. Chang C.S., Du D. Differential Evolution Based Tuning of Fuzzy Automatic Train Operation for Mass Rapid Transit System. *IEE Proc. Electric Power Appl.* 2000. Vol. 147. No. 3. P. 206—212. URL: https://digital-library.theiet.org/content/journals/10.1049/ip-epa:20000329

7.38. Engelbrecht A. Computational Intelligence. An Introduction: Sec. ed. John Wiley & Sons, Ltd., 2007. 630 p.

7.39. Dikshit Chauhan, Shivani, Donghwi Jung, Anupam Yadav. Advancements in Multimodal Differential Evolution: A Comprehensive Review and Future Perspectives. 2025. arXiv: https://arxiv.org/abs/2504.00717

7.40. Schaffer J.D. Multiple Objective Optimization with Vector Evaluated Genetic Algorithms. *Proc. First Intern. Conf. Genet. Algorithms*. 1985. P. 93—100. URL: https://www.researchgate.net/publication/220885605_Multiple_Objective_Optimization_with_Vector_Evaluated_Genetic_Algorithms

7.41. Dijkstra E.W. A note on two problems in connexion with graphs. *Numerische Mathematik.* 1959. Vol. 1. P. 269—271. URL: https://doi.org/10.1007/BF01386390

РОЗДІЛ 8

РОЙОВІ АЛГОРИТМИ ОБЧИСЛЮВАЛЬНОГО ІНТЕЛЕКТУ

8.1. Загальна характеристика кооперативної поведінки в ройових алгоритмах

8.2. Ройова оптимізація

8.3. Вибір параметрів алгоритмів РО

8.4. Гібридні алгоритми РО

8.5. Мурашині алгоритми

8.6. Алгоритми колоній мурах

8.7. Застосування мурашиних алгоритмів у задачі комівояжера

8.8. Аналіз результатів експериментів

Висновки

Список літератури

8.1. Загальна характеристика кооперативної поведінки в ройових алгоритмах

Останнім часом широкого застосування набули ройові алгоритми обчислювального інтелекту для завдань оптимізації та прийняття рішень.

Під роєм розуміють групу індивідів, які вирішують деяке загальне завдання (наприклад, пошук глобального мінімуму деякої функції) і обмінюються між собою локально доступною інформацією, тобто своїми результатами пошуку. Ця інформація розподіляється по всій групі і використовується членами рою (частинками), тому проблема вирішується ефективніше, ніж її можна було б вирішити одним індивідом. Взаємодія між членами рою — так званими агентами (або частинками) призводить до розподілених колективних стратегій вирішення проблеми [8.1]. Ройовий інтелект належить до поведінки з вирішення проблем, що з'являється в результаті взаємодії цих агентів, а ройовий обчислювальний інтелект — до алгоритмічних моделей такої поведінки. Ройовий інтелект називають також колективним інтелектом. Вивчення соціальних тварин і соціальних комах (мурах, бджіл) сприяло появі моделей ройового інтелекту.

Біологічними ройовими системами, які надихнули вчених на створення обчислювальних моделей, є мурахи, бджоли, павуки та зграї птахів. Усередині цих роїв окремі індивіди відносно прості за структурою, але їхня колективна поведінка зазвичай дуже складна. Складна поведінка рою — це результат взаємодії індивідів у рої з часом. Вона не є властивістю будь-якого окремого індивіда і зазвичай не є легко передбачуваною чи дедуктивно виведеною з простої поведінки індивідів. Ця властивість є емерджентністю. Більш формально емерджентність — це процес появи деяких нових когерентних структур, патернів та поведінкових властивостей у складній системі [8.1]. Такі структури, патерни і поведінка виникають без будь-якої координаційної системи управління, а внаслідок взаємодії індивідів зі своїми локальним середовищем (оточенням). Зв'язок між індивідами та колективною поведінкою такий: колективна поведінка індивідів формує поведінку рою. З іншого боку, поведінка рою впливає на умови, в яких кожен індивід виконує свої дії. Ці дії можуть змінити середовище — і через це поведінка

індивідів та їхніх сусідів може змінюватись також, що зі свого боку призведе до зміни колективної поведінки рою. Звідси випливає, що найважливішим інгредієнтом ройового інтелекту та стимулятором емерджентної поведінки є взаємодія, або кооперація, індивідів. Така взаємодія може бути як прямою (за допомогою фізичного контакту або візуального, аудіо- чи хімічного сприйняття), так і непрямою (через локальну зміну середовища). Можна навести численні приклади емерджентної поведінки з живої природи:

• терміти будують великі житла, які за складністю далеко переважають здатності (фізичні та інтелектуальні) окремого терміта;

• динамічні завдання розподіляються в колонії мурах без центрального менеджера чи координатора;

• птахи у зграї чи риби самоорганізовуються в оптимальні просторові зразки поведінки (патерни). Зграї риб визначають свій напрямок руху і швидкість з огляду на невелику кількість сусідів;

• хижаки, наприклад група левиць, здійснюють стратегії спільного полювання, щоб перехитрити свою здобич, тощо.

Мета ройових моделей обчислювального інтелекту — через моделювання простої поведінки індивідів та їх взаємодії зі своїми сусідами й середовищем досягти більш складної поведінки, яку можна використати для вирішення комплексних проблем, насамперед оптимізаційних [8.1, 8.3]. Наприклад, ройові моделі оптимізації моделюють два простих зразки поведінки:

1) кожен індивід рухається у напрямку свого найближчого найкращого сусіда;

2) кожен індивід рухається назад до стану, який виявився з досвіду найкращим для нього самого.

З іншого боку, оптимізаційні моделі на основі колоній мурах моделюють просту поведінку руху слідами, в яких кожна мурашка сприймає концентрацію т. зв. *феромонів* у своєму локальному оточенні і вибирає ймовірнісно напрямок їх найбільшої концентрації. Звідси виникає поведінка, спрямована на знаходження найкращої альтернативи (найкоротшого шляху) з множини альтернатив. Моделі локальної поведінки мурах, що відвідують цвинтарі, приводять до моделей складної поведінки групування подібних об'єктів у кластери.

8.2. Ройова оптимізація

Алгоритм ройової оптимізації (від англ. «*Particle Swarm Optimization», PSO*), або РО — це популяційний алгоритм пошуку, заснований на моделюванні соціальної поведінки птахів у зграї. Початковою метою концепції ройової поведінки було графоаналітичне моделювання граціозного та непередбачуваного польоту зграї птахів з метою виявлення патернів, які

управляють здатністю птахів летіти синхронно і раптово змінювати напрямок, перебудовуючись у зграю оптимальної форми. З цієї початкової мети концепція розвинулася у простий та ефективний алгоритм оптимізації. У ройовій оптимізації індивіди, які називають частинками (*particle*), рухаються (летять) у багатовимірному просторі пошуку. Зміни у положенні частинок у просторі пошуку засновані на соціально-психологічній тенденції індивідів емулювати успіх інших індивідів.

Базові концепції ройової оптимізації. Частинки (індивіди) у рої дотримуються дуже простого правила поведінки: емулювати успіхи сусідніх індивідів та власні успіхи. Колективна поведінка, що складається з цієї простої поведінки, полягає у виявленні оптимальних ділянок високорозмірного простору пошуку.

Алгоритм РО (АРО) моделює рій частинок, в якому кожна частинка надає потенційне рішення. За аналогією з еволюційною парадигмою рій відповідає популяції, тоді як окрема частинка — індивіду. Частинки рухаються в багатовимірному просторі пошуку, в якому позиція кожної частинки налаштовується відповідно до її власного досвіду та досвіду сусідів.

Нехай $x_i(t)$ позначає позицію частинки i у просторі пошуку в момент часу t, де t — дискретний час. Тоді позиція частинки змінюється внаслідок додавання швидкості $v_i(t)$ з урахуванням того, що інтервал часу $\Delta t = 1$, у такому виразі:

$$x_i(t + 1) = x_i(t)+v_i(t + 1), \tag{8.1}$$

де $x(0)$ рівномірно розподілено в інтервалі $[x_{\max}, x_{\min}]$, що записується так: $x(0) \sim U(x_{\max}, x_{\min})$.

Саме вектор швидкості надає руху процес оптимізації і відображає як експериментальні знання частинки, так і соціальну інформацію від її сусідів. Експериментальні знання частинки зазвичай називають когнітивним компонентом, який пропорційний відстані від свого найкращого становища (персональна найкраща позиція), знайденого з першого кроку. Інформація, отримана в результаті обміну з соціумом, називається соціальним компонентом рівняння швидкості.

Глобальна ройова оптимізація (ГРО)

В алгоритмі глобальної найкращої ройової оптимізації (*gbestPSO*) [8.1—8.5] сусідами для поточної частинки є весь рій; соціальна мережа, використовувана в алгоритмі ГРО, відображує топологію «зірка». Для зіркоподібної топології сусідства соціальний компонент швидкості частинки відображує інформацію, отриману від усіх частинок у рої. У цьому разі соціальна інформація є найкращою позицією, знайденою роєм, яка позначається $\hat{y}(t)$.

Для алгоритму gbestPSO швидкість частинки i розраховують так:

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_1(t)[y_{ij}(t) - x_{ij}(t)] + c_2 r_2(t)[\hat{y}_j(t) - x_{ij}(t)], \qquad (8.2)$$

де v_{ij} — швидкість частинки i в напрямку $j = 1, 2, \ldots, n_{xy}$ в момент часу t; $x_{ij}(t)$ — позиція частинки i; c_1 та c_2 — додатні константи прискорення, що використовуються для масштабування внеску когнітивної та соціальної компонент відповідно; $r_1(t)$ та $r_2(t)$ — випадкові величини, рівномірно розподілені в інтервалі [0, 1]. Ці випадкові величини вносять елемент стохастичності в алгоритм.

Персонально найкраща позиція y_i, пов'язана з частинкою i, — це найкраща позиція, яку займала частинка i, починаючи з нульового кроку.

Для задач мінімізації найкраща позиція майбутнього часу визначається за виразом

$$y_i(t+1) = \{y_i(t), \text{ якщо } f(x_i(t+1)) \geq f(y_i(t)); \quad x_i(t+1), \text{ якщо } f(x_i(t+1)) < f(y_i(t)), \qquad (8.3)$$

де $f: R^{n_x} \to R$ — фітнес-функція.

За аналогією з ЕА фітнес-функція оцінює близькість відповідного рішення до оптимуму, тобто оцінює показник якості частинки (рішення) i.

Глобально найкраща позиція у момент часу t визначається за виразом

$$\hat{y}(t) \in \{y_0(t), \ldots, y_{n_s}(t)\} \mid f(\hat{y}(t)) = \{f(y_i(t))\}, \qquad (8.4)$$

де n_s — число частинок у рої.

Глобально найкращу позицію можна обрати з частинок поточного рою, у цьому випадку так:

$$\hat{y}(t) = \{f(x_0(t)), \ldots, f(x_{n_s}(t))\}. \qquad (8.5)$$

Локально найкраща РО

Алгоритм локальної РО використовує кільцеву топологію соціальної мережі, в якій для кожної частинки визначено менші (порівняно з глобальним алгоритмом) зони сусідства. Соціальні компоненти надають інформацію в результаті обміну із сусідами, відображаючи локальні знання про середовище.

Щодо рівняння швидкості соціальний внесок у швидкість частинки пропорційний відстані між частинкою та найкращою позицією, знайденою сусідами частинки у напрямку j. Локально найкращу позицію $\hat{y}_i$, тобто найкращу позицію, знайдену в оточенні сусідів N_i, обчислюють за рівнянням

$$\hat{y}_i(t+1) \in \{N_i \mid f(\hat{y}_i(t+1))\} = \{f(x(t))\}, \qquad (8.6)$$

де зона сусідства розміром N_i має вигляд

$$N_i = \{y_{i-nN_i}(t), y_{i-nN_i+1}(t), \ldots, y_i(t), y_{i+1}(t), y_{i+N_i}(t)\}. \qquad (8.7)$$

Локально найкраща позиція також називається найкращою позицією сусідства.

Важливо зазначити, що в базових алгоритмах РО частинки в середині зони сусідства не взаємодіють одна з одною.

Вибір зони сусідства в такому разі ґрунтується лише на індексах частинок. Є дві основні причини, чому визначення зони сусідства на основі індексів частинок виявляється кращим.

1. Досить просте обчислення, оскільки не потрібне просторове впорядкування частинок. Для підходів, у яких використовують відстань між частинками, для визначення зони сусідства необхідно обчислити евклідову відстань між усіма парами частинок, яка має складність порядку $o(n_s^2)$.

2. Швидке поширення інформації щодо позитивних рішень на всі частинки незалежно від їх розміщення у просторі рішень. Слід зазначити, що зони сусідства перетинаються. Кожна частинка є елементом множини зон сусідства. Взаємодопомога зон сусідства полегшує також обмін інформацією між ними і забезпечує збіжність алгоритмів РО до єдиної точки, а саме, до глобально найкращої частинки позиції.

Компоненти швидкості. Відповідно до (8.2), для обчислення швидкості використовують три члени.

1. *Попередню швидкість* $v_i(t)$, яка відображає пам'ять про попередній напрямок руху, тобто в найближчому минулому. Цей член можна розглядати як миттєвий, який перешкоджає частинці різко змінювати напрямок, його можна трактувати як інерцію системи.

2. *Когнітивний компонент* $c_1 r_1 (y_i - x_i)$, який оцінює якість функціонування частинки щодо її минулого. У цьому сенсі когнітивний компонент відображає індивідуальну пам'ять частинки про позицію, яка була для неї найкращою в минулому. Ефект цього члена полягає в тому, що частинки наближаються до своїх найкращих позицій. Внаслідок відображення тенденції індивідів повертатися до ситуацій, де їм було найкраще в минулому, когнітивний компонент можна вважати свого роду ностальгією частинки [8.1, 8.4].

3. *Соціальний компонент* $c_2 r_2 (\hat{y} - x_0)$ для глобального алгоритму РО або $c_2 r_2 (\hat{y}_i - x_i)$ у разі локального алгоритму РО, який оцінює характеристики якості частинки i відносно групи частинок чи сусідів. Ефект соціального компонента в тому, що кожна частинка притягується також до кращої позиції, знайденої її сусідами.

Дії когнітивного і соціального компонентів зважуються стохастичною величиною $c_1 r_1$ або $c_2 r_2$ відповідно.

Приклад руху частинки у двовимірному просторі показано на рис. 8.1, де для зручності індекс i частинки опущено. Рис. 8.1, *а* ілюструє стан частинки в момент часу t. Зауважимо, що нова позиція $x(t + 1)$ ближче до гло-

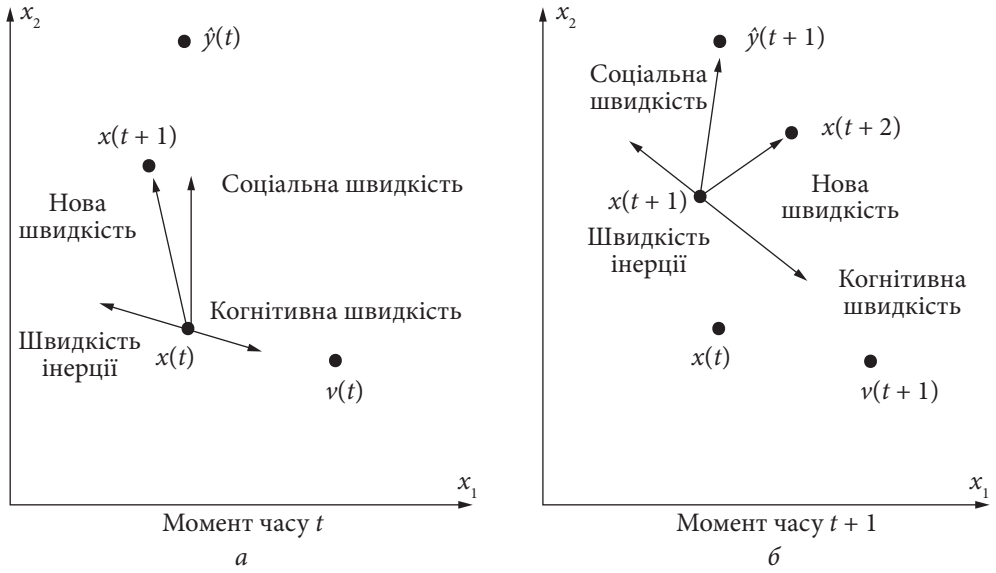


Рис. 8.1. Внесок трьох компонентів у рух частинки до глобально найкращого рішення

бального найкращого рішення $\hat{y}(t)$. На рис. 8.1, *б* у момент часу $t + 1$ передбачено, що особиста позиція частинки не змінилася.

Звичайно, можливо, що частинка «перескочить» через глобально найкращу позицію, головно внаслідок дії миттєвого члена. Це призводить до двох можливих сценаріїв.

1. Нова позиція як результат перескакування через поточне глобально найкраще рішення може виявитися кращою, ніж поточний глобальний оптимум, і всі частинки будуть рухатися в напрямку до нього.

2. Нова позиція гірша, ніж поточна глобально найкраща позиція. У наступні моменти часу когнітивний і соціальний компоненти змусять частинку змінити напрямок до глобально найкращої позиції. Кумулятивний ефект адаптації всіх позицій частинки полягає в тому, що кожна частинка прагне рухатись до точки на прямій, що з'єднує глобальну позицію рою з індивідуальною найкращою позицією частинки.

На рис. 8.2 відображено адаптацію позицій частинок у задачі мінімізації функції розмірності рішень $n = 2$ зі змінними x_1 та x_2 з використанням алгоритму глобальної РО. Тут оптимум знаходиться на початку координат і позначений символом «×». На рис. 8.2, *а* показано початкові позиції восьми частинок, де глобально найкращим рішенням є $\hat{y}$. Оскільки внесок когнітивного компонента для кожної частинки на момент часу $t = 0$ дорівнює 0, лише соціальний компонент впливає на величину адаптації. На рис. 8.2, *б* показано нові позиції всіх частинок після першої ітерації ($t = 1$). Знайдено нову позицію глобально оптимального рішення $\hat{y}$. Тож на рис. 8.2, *б* тепер

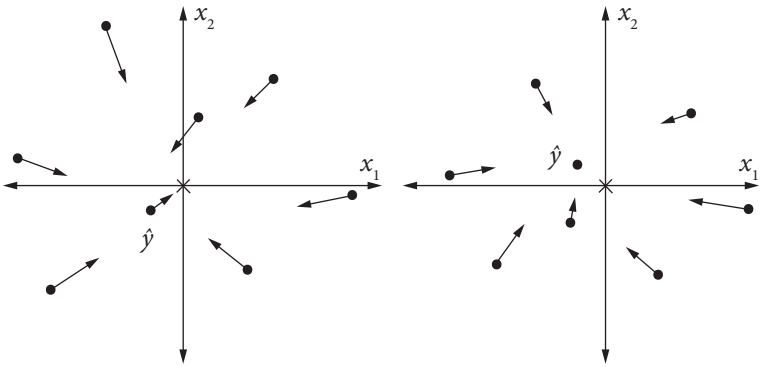


Рис. 8.2. Початкові і нові позиції частинок

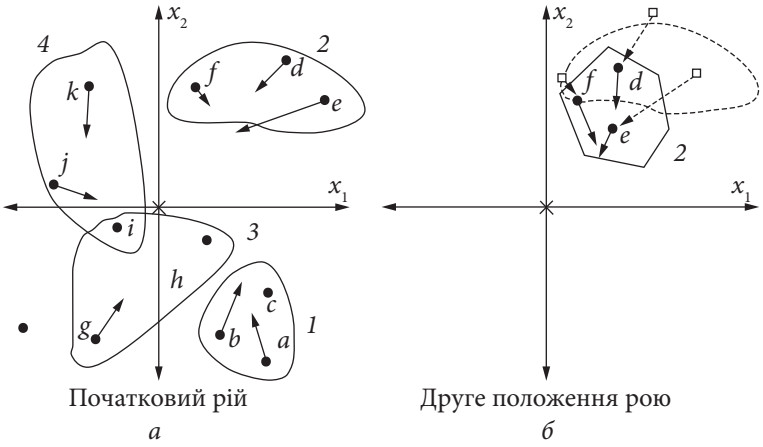


Рис. 8.3. Вплив на рух частинок їхніх безпосередніх сусідів

вказано вплив всіх компонент швидкості і частинки рухаються до нового глобального рішення.

Зрештою, на рис. 8.3 показано, як впливають на рух частинок їхні безпосередні (найближчі) сусіди. Щоб спростити рисунок, зображено лише частину рухів і вказано тільки загальний (підсумковий) напрямок швидкості. В зоні сусідства *1* обидві частинки — *a* і *b* рухаються у напрямку *c*, що є локально оптимальною позицією для цієї зони. В зоні *2* частинки *d* і *e* рухаються у напрямку *f*. На наступній ітерації позиція *e* буде найкращим рішенням для зони *2*. Тепер частинки *d* і *f* рухаються до *e*. При цьому показано лише частину простору рішень. Зауважимо, що загальний рух усіх частинок відбувається в напрямку шуканого мінімуму (початок координат).

8.3. Вибір параметрів алгоритмів РО

На першому кроці алгоритму РО необхідно ініціалізувати рій та параметри управління. У контексті базового алгоритму РО мають бути визначені константи прискорення c_1 і c_2, початкові швидкості, позиції частинок та індивідуальні найкращі позиції частинок. До того ж у локальному алгоритмі РО (*lbestPSO*) має бути визначено розміри зон сусідства. Зазвичай позиції частинок ініціалізуються так, щоб рівномірно покрити простір пошуку. Важливо зазначити, що на ефективність РО значно впливає початкова різноманітність частинок рою, тобто наскільки добре вони покривають пошуковий простір і наскільки рівномірно частинки розподілені по простору. Так, якщо деякі регіони простору пошуку не покриті початковим роєм, то алгоритми РО матимуть труднощі у відшуканні оптимуму, якщо він розміщений у деякій іншій зоні. Алгоритм РО виявить такий оптимум тільки у разі, якщо миттєвий член частинки приведе її до непокритої зони.

Припустимо, що оптимум розміщений у зоні, обмеженій двома векторами — $x_{\min}$ і $x_{\max}$.

Тоді ефективним методом ініціалізації для позицій частинок буде такий:

$$x(0) = x_j + r_j(x_j - x_j), \quad \forall j = 1, 2, ..., n_x,$$

де $r_j \sim U(0; 1)$.

Початкові швидкості можуть бути ініціалізовані нулями, тобто $r_i(0) = 0$. Індивідуальна найкраща позиція для кожної частинки ініціалізується в момент часу $t = 0$. Отже, $y_i(0) = x_i(0)$.

Розглянемо такі умови зупинення алгоритмів РО [8.4, 8.5].

1. Зупинка, коли досягнуто максимальну кількість ітерацій.

2. Зупинка, коли виявлено прийнятне рішення. Припустимо, що x^* є оптимальним рішенням для цільової функції f. Тоді цей критерій зупинить процес пошуку, щойно буде знайдено такий x_i, що $|f(x) - f(x^*)| \le \varepsilon$.

3. Зупинка, якщо протягом деякого числа ітерацій немає поліпшення критерію.

4. Зупинка, коли нормалізований радіус рою близький до нуля.

Базові параметри. На характеристики базового алгоритму РО впливає множина параметрів, а саме: розмірність задачі, кількість частинок рою, коефіцієнти прискорення, інерція, розміри зони сусідства, кількість ітерацій, випадкові величини, які масштабують внесок когнітивного та соціального компонентів.

Розглянемо роль деяких важливих параметрів РО.

1. *Розмір рою* n_s, тобто число частинок у рої. Чим більше частинок, тим більша початкова різноманітність рою за умови, що використовується при-

датна схема для ініціалізації частинок. Однак збільшення кількості частинок підвищує обчислювальні витрати. Емпіричними дослідженнями було показано, що алгоритми РО мають здатність знаходити оптимальні рішення з малими розмірами рою — від 10 до 30 частинок.

2. *Розміри зони сусідства*. Розміри зони сусідства визначають екстент соціальної взаємодії рою. Чим менші розміри зони сусідства, тим менша взаємодія між частинками. Великі розміри зони сусідства менш сприйнятливі до локальних мінімумів. Щоб отримати переваги малих і великих розмірів зон сусідства, слід починати з малих розмірів і збільшувати їх пропорційно збільшенню кількості ітерацій. Такий підхід забезпечує початкову високу різноманітність з підвищеною збіжністю переміщення частинок у перспективну ділянку пошуку.

3. *Кількість ітерацій*. Число ітерацій залежить від вирішуваної проблеми. Дуже мала кількість ітерацій може призвести до передчасного закінчення процесу. Занадто багато ітерацій спричинює додаткові обчислювальні витрати.

4. *Коефіцієнти прискорення*. Коефіцієнти прискорення c_1 і c_2 разом із випадковими векторами r_1 і r_2 регулюють стохастичний вплив когнітивного та соціального компонентів на загальну швидкість частинки. Константи c_1 і c_2 також розглядаються як параметри довіри, де c_1 відображає ступінь довіри частинки самій собі (своїм результатам), тоді як c_2 — ступінь довіри своїм сусідам.

За $c_1 = c_2 = 0$ частинки продовжують рухатися зі своєю поточною швидкістю, поки не вийдуть на межу зони пошуку (припущення відсутності інерції). Якщо $c_1 \gg 0$, а $c_2 = 0$, то всі частинки — незалежні шукачі мінімуму. Кожна частинка знаходить найкращу позицію у своїй зоні сусідства і замінює нею поточну найкращу позицію, якщо нова позиція краща. З іншого боку, якщо $c_2 > 0$, а $c_1 = 0$, всі частинки рою притягуються до однієї точки. Найкращий варіант у багатьох випадках — вибрати $c_1 \approx c_2$.

У низці робіт було запропоновано, щоб c_1 зменшувалося за лінійним законом, а c_2 зростало лінійно. Така стратегія фокусується на пошуку в початковій стадії оптимізації і на збіжність до доброго оптимуму наприкінці процесу оптимізації завдяки притягуванню частинки в зону найкращого (або глобально найкращого) рішення. Значення $c_1(t)$ і $c_2(t)$ у момент часу t обчислюють так:

$$c_1(t)=(c_{1\min} - c_{1\max}) \frac{t}{n_t} + c_{1\max};$$

$$c_2(t)=(c_{2\max} - c_{2\min}) \frac{t}{n_t} + c_{2\min}, \qquad (8.8)$$

де $c_{1\max} = c_{2\max} = 2{,}5$; $c_{1\min} = c_{2\min} = 0{,}5$.

8.4. Гібридні алгоритми РО

Розглянемо кілька алгоритмів РО, які використовують одну або кілька концепцій ЕП та ГА.

Алгоритми РО, засновані на селекції. У низці робіт було показано, що, використовуючи ідеї ГА і комбінуючи їх з алгоритмами РО додаванням процесу селекції, аналогічного тому, що відбувається в еволюційних алгоритмах, можна суттєво покращити їхні характеристики. Процедуру селекції виконують перед обчисленням корекції швидкості. Один із таких алгоритмів описано далі.

1. Обчислити фітнес-індекс усіх частинок.

2. Для кожної частинки i : ($i = \overline{1, n_s}$):

а) випадково вибрати n_{ts} частинок;

б) визначити характеристики частинки і порівняно з n_{ts} випадково вибраними частинками;

в) сортувати рій за цим показником;

г) замінити гіршу половину рою на кращу половину без зміни індивідуальних найкращих позицій.

Хоча найгірші частинки «штрафують» видаленням їх з рою, пам'ять про їхні найкращі позиції зберігається і процес пошуку триває, ґрунтуючись на раніше набутому досвіді. Було емпірично показано, що алгоритми РО, засновані на селекції, поліпшують локальні пошукові властивості. Однак такий підхід істотно зменшує різноманітність рою. Оскільки половина рою замінюється іншою половиною, то різноманітність зменшується на 50 % на кожній ітерації. Різноманітність можна поліпшити заміною найгірших індивідів на мутовані копії кращих індивідів. Функціональні характеристики можна також поліпшити, розглядаючи заміну тільки в тому разі, якщо нова частинка має краще значення Ф-І, ніж видалена. За такого підходу, запропонованого Ч.Е. Коай (Chin Ann Koay) і Д. Шрінівасаном (Dipti Srinivasan) [8.5], кожна частинка генерує нащадка шляхом мутації.

Репродукція. Репродукція — процес генерації нащадка з індивідів поточної популяції. Різні схеми репродукції було використано у рамках РО.

В одному з перших підходів [8.9] частинці дозволено генерувати нову частинку і вбивати себе або модифікувати коефіцієнти інерції та прискорення з урахуванням умов зовнішнього середовища. Якщо немає достатнього поліпшення в зоні сусідства частинки, то частинка породжує нову частинку в цій зоні. З іншого боку, якщо спостерігається достатнє поліпшення у сфері сусідства, то найгірша частинка цієї зони відбраковується (відкидається).

У разі використання такого підходу до репродукції та відбракування ймовірність додавання зменшується зі збільшенням розміру рою. З іншого боку, зменшення розміру рою збільшує ймовірність породження нових частинок. Аналогічний підхід було реалізовано в динамічній зміні розміру

рою. Він був розроблений за аналогією з природним процесом адаптації амеби. Коли амеба отримує позитивний зворотний зв'язок із середовищем (тобто є достатньо джерел їжі), вона здійснює репродукцію. І навпаки, коли джерела їжі збіднюються, репродукція зменшується. Використовуючи таку аналогію щодо проблеми оптимізації, було реалізовано підхід, у якому враховано, що якщо частинка перебуває в ділянці потенційного мінімуму, то кількість частинок у цій ділянці збільшується. В одному з таких підходів пропонується розмноження лише глобально найкращої частинки (у разі використання глобального алгоритму РО) для зменшення обчислювальної складності процесу розмноження. Вибір розмноження лише такої частинки можна мотивувати тим, що глобально найкраща частинка є першою частинкою, яку виявляють у ділянці потенційного оптимуму.

Алгоритм розмноження глобально найкращих частинок. Якщо $y(t)$ перебуває в ділянці потенційного мінімуму, то потрібно:

1) покласти $\hat{y} = y(t)$;

2) для $k = 1$ до n_k (числа нащадків) виконати $y_a = \hat{y}(t) + N(0, \sigma)$, якщо $f(y_a) \ll f(\hat{y})$, то $\hat{y} = y_a$; цей процес повторюється доти, доки не виконається $f(\hat{y}) \geq f(y(t))$, $y(t) = \hat{y}$; кінець.

Було запропоновано оператор арифметичного кросовера для генерування нащадка з двох випадково вибраних частинок. Припустимо, що частинки a і b обрано для кросовера. Тоді відповідні позиції (частинки) $x_a(t)$ і $x_b(t)$ замінюються на нащадків

$$x_{i_1}(t+1) = r(t)x_{i_1}(t) + (1 - r(t)x_{i_2}(t), \tag{8.9}$$

$$x_{i_2}(t+1) = r(t)x_{i_2}(t) + (1 - r(t)x_{i_1}(t) \tag{8.10}$$

з відповідними швидкостями:

$$v_{i_1}(t+1) = \frac{v_{i_1}(t) + v_{i_2}(t)}{\left\| v_{i_1}(t) + v_{i_2}(t) \right\|} \left\| v_{i_1}(t) \right\|; \tag{8.11}$$

$$v_{i_2}(t+1) = \frac{v_{i_1}(t) + v_{i_2}(t)}{\left\| v_{i_1}(t) + v_{i_2}(t) \right\|} \left\| v_{i_2}(t) \right\|, \tag{8.12}$$

де $r(t) \sim U(0, 1)^{n_x}$, $x_{i_1}(t) = x_a(t)$, $x_{i_2}(t) = x_b(t)$.

Індивідуально найкраща позиція нащадка ініціалізується його поточною позицією (тобто якщо частинка i_1 залучена в кросовер, то $y_{i_1}(t+1) = x_{i_1}(t+1)$).

Частинки вибирають для розмноження з імовірністю, визначеною користувачем. За умови, що ця ймовірність менша від 1, не всі частинки замінюються нащадками. Частинки вибираються як батьки за допомогою випадкового механізму, а не на основі їхніх Ф-І. Це перешкоджає домінуванню найкращих частинок у процесі розмноження. Якби найкращим частинкам було дозволе-

но домінувати, це призвело б до різкого скорочення різноманітності рою, що спричинило б передчасну збіжність. Було емпірично встановлено, що низька ймовірність розмноження $P_r = 0{,}2$ забезпечує добрі результати.

Мутації. Гауссові мутації додаються головно для налаштування позицій векторів після застосування адаптації швидкості додаванням випадкових значень, розподілених за гауссовим розподілом, до компонентів векторів позиції частинок $x_i(t)$.

Було запропоновано підхід, у якому компоненти векторів позицій частинок мутують з певною ймовірністю. Нехай $x'_i(t+1)$ вказує нову позицію частинки i після застосування рівнянь адаптації швидкості та позицій, нехай P_m — ймовірність мутації компоненти. Тоді для кожної компоненти $j = 1, 2, \ldots, n_x$, якщо $U(0; 1) < P_m$, то компонента $x'_{ij}(t+1)$ мутує згідно з виразом

$$x_{ij}(t+1) = x_{ij}(t+1) + N(0; \sigma)x'_{ij}(t+1), \tag{8.13}$$

де стандартне відхилення σ визначається за виразом

$$\sigma = 0{,}1(x_{j\max} - x_{j\min}). \tag{8.14}$$

У низці робіт запропоновано безпосереднє застосування гауссового оператора мутацій ЕП:

$$x_{ij}(t+1) = x_{ij}(t+1) + \eta_{ij}(t)N(0; 1),$$

де η_{ij} регулює розміри кроків мутації для кожної компоненти:

$$\eta_{ij}(t) = \eta_{ij}(t-1)e^{\tau' N(0,1)+\tau N_j(0,1)}, \tag{8.15}$$

а τ і τ' визначено за рівнянням (7.58) (див. розд. 7).

При $\eta_{ij}(0) \in (0, 1)$ розміри кроків мутацій зменшуються зі збільшенням часу t, що забезпечує початковий широкий пошук по ділянці з тонким підлаштуванням рішень у пізніший час. Зауважимо, що збіжність не можна забезпечити, якщо кроки мутацій не будуть зменшуватися з часом.

Усі компоненти однієї й тієї самої частинки мутують, використовуючи одне й те саме значення $N(0, 1)$. Однак для кожної компоненти j додається випадкове число $N(0, 1)$, тобто кожна компонента мутується на різну величину. Величина (рівень) мутації залежить від розмірності задачі, отриманої за допомогою обчислення τ і τ'. Чим більше n_x, тим менші розміри кроків мутації.

8.5. Мурашині алгоритми

Мурахи з'явилися близько 100 млн років тому, і їхня земна повна популяція налічує приблизно 10^{16} індивідів [8.13]. Проведена оцінка загальної ваги мурах має той самий порядок, що й загальна вага людей. Більшість мурах — соціальні тварини, що живуть у колоніях загальною чисельністю до

30 млн особин. Колективна поведінка у колоніях мурах завжди інтригувала людство. Було проведено багато досліджень мурашиних колоній, спрямованих на краще розуміння колективної поведінки. Досліджували колективну поведінку щодо пошуку продовольства, поділу праці, організації цвинтарів, турботи про потомство, конструкції гнізд.

Поведінка мурах щодо забезпечення продовольством

Метаевристики оптимізації мурашиної колонії. Одним з перших типів поведінки, дослідженим ентомологами, є здатність мурах знаходити найкоротший шлях між їхнім гніздом та джерелом їжі. В результаті досліджень поведінки мурах під час пошуку їжі було розроблено алгоритми, які називають *метаевристиками оптимізації колонії мурах* (ACO-MH) [8.13].

Мета дослідження поведінки колоній мурах щодо забезпечення продовольством полягала у з'ясуванні механізму: як мурахи шукають найкоротший шлях між своїм гніздом (мурашником) та джерелом їжі без будь-яких видимих активних механізмів координації. Дослідження поведінки кількох видів мурах розкрили спочатку випадкову чи хаотичну їх діяльність у пошуках їжі. Однак щойно джерело їжі виявлено, зразки поведінки стають організованішими і все більше й більше мурах рухаються до їжі тим самим найкоротшим шляхом. Невдовзі майже всі мурахи магічно йдуть тим самим найкоротшим шляхом. Механізми регулювання різняться для різних видів мурах і можуть здійснюватися за допомогою безпосереднього контакту або опосередковано.

Більшість різновидів мурах використовують непряму форму контакту через *феромонові сліди* (*хімічні мітки, які мурахи залишають на поверхні, коли рухаються, щоб передавати інформацію іншим особинам своєї колонії*). Коли мураха знаходить джерело їжі, вона тягне на собі частину їжі в мурашник і при цьому розкладає феромони по своїх слідах. Фуражні мурахи вирішують, яким шляхом рухатись на основі концентрації феромонів. Шляхи з більшою концентрацією феромонів мають більшу можливість бути обраними. У міру того, як все більше мурах йде певним шляхом, бажаність цих шляхів підкріплюється великою кількістю феромонів, залишених мурахами, які приваблюють все більше мурах рухатися цим шляхом. Колективна поведінка, яка тут результується, є формою «автокаталітичної» поведінки, коли позитивний зворотний зв'язок за їжею зумовлює зростання кількості мурах, що йтимуть по цьому шляху.

В Аргентині було проведено експерименти з фуражної поведінки мурах з використанням мосту [8.10]. Як показано на рис. 8.4, мурашник відділений від джерела їжі мостом з двома довгими гілками однакової довжини. Кінець кінцем було обрано один з шляхів і більшість мурах рухалися цим шляхом, хоча там було два маршрути однакової довжини.

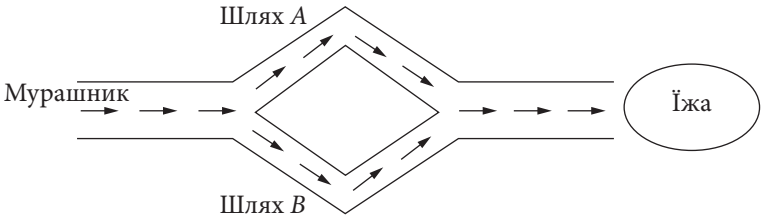


Рис. 8.4. Експерименти з фуражної поведінки мурах

На основі цього експерименту, що дістав назву *подвійний міст*, було розроблено просту формальну модель для опису процесу вибору шляху. З цією метою припустимо, що всі мурахи відкладають однакову кількість феромонів і феромони не випаровуються. Нехай $n_A(t)$ і $n_B(t)$ позначають число мурах, які вибрали шлях A або B відповідно в момент часу t. Ж.-Л. Денеборг (Jean-Louis Deneubourg), С. Арон (Serge Aron), С. Госс (Simon Goss), Ж.-М. Пастелс (Jean-Marie Pasteels) [8.10] емпірично визначили, що можливість для наступної мурахи вибрати шлях A в момент часу $(t + 1)$ задається рівнянням

$$p_A(t+1) = \frac{c + n_A(t))^{\alpha}}{(c + n_A(t))^{\alpha} + (c + n_B(t))^{\alpha}} = 1 - p_B(t+1), \qquad (8.16)$$

де c — кількісна оцінка ступеня залучення недослідженого шляху; α — зміщення (схильність використовувати феромони-депозити в процесі прийняття рішень). Чим більше α, тим вища ймовірність того, що наступна мураха піде шляхом з більшою концентрацією феромонів. Чим більше c, тим більше феромонів потрібно відкласти для невипадкового вибору маршруту. У ході досліджень визначено, що $\alpha = 2$ і $c \approx 20$ забезпечують найкраще узгодження з експериментом.

З використанням імовірності, визначеної за рівнянням (8.16), вирішальне правило мурахи, що прибуває до бінарного мосту, виглядає так: якщо $U(0, 1) \le p_A(t + 1)$, то йти за маршрутом A, інакше — за маршрутом В. Було розширено експеримент з бінарним мостом, коли одна з гілок мосту виявляється довшою, ніж друга (рис. 8.5, на якому точками позначено мурах). Спочатку шлях мурахи вибирають випадково, і приблизно однакова кількість мурах рухається обома шляхами. Однак з плином часу все більше мурах рухаються коротшим шляхом. Як бачимо, селекція зміщена у бік коротшого шляху (див. рис. 8.5, *б*). Це відбувається тому, що мурахи, які рухаються найкоротшим шляхом, повертаються в гніздо раніше, ніж мурахи, що рухаються довшим шляхом. Тому феромони на коротшому шляху поповнюються швидше. Встановлено, що можливість вибору коротшого шляху збільшується зі зростанням відношення довжин шляхів обох маршрутів. Цей ефект було названо ефектом *диференціальної довжини шляху* [8.9]. Отже, підсумовуючи результати дослідів з реальними мурахами, можна

стверджувати, що поведінка вибору найкоротших шляхів пояснюється автокаталізом (додатним зворотним зв'язком) та ефектом диференціальної довжини шляху. Мураху можна розглядати як агент, що реагує на стимули, — мураха спостерігає концентрації феромонів і виконує дію на основі їх вимірювання. Тому мураху можна вважати простим обчислювальним агентом.

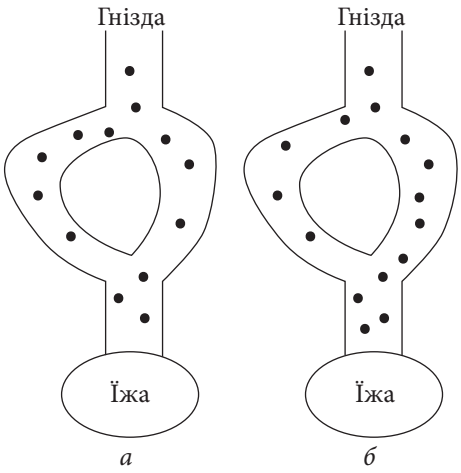


Рис. 8.5. Розширений експеримент з бінарним мостом

Стигмергія і штучні феромони. Взагалі стигмергія — це клас механізмів, які є медіаторами взаємодії тварина—тварина. Слово «стигмергія» походить від двох грецьких слів: «стигма» (означає знак) та «ергон» (праця) [8.1]. Індивіди спостерігають сигнали, що зумовлюють специфічні реакції чи дії. Ці дії можуть підкріплювати або змінювати сигнали, які впливають на дії інших індивідів. Було визначено дві форми стигмергії [8.10]: семотектонічна та на основі сигналів (сигнальна). Сематектонічна стигмергія належить до комунікацій через зміни у фізичних характеристиках оточення. Прикладами діяльності, що здійснюється за допомогою сематектонічної стигмергії, є будівництво спільних будинків, сортування потомства. Стигмергія, *заснована на сигналах,* сприяє комунікаціям через сигнальний механізм, який реалізується за допомогою хімічних компонентів, що відкладаються мурахами. Наприклад, поведінка пошуку їжі з'являється у мурах, які рухаються слідами феромонів, відкладених іншими мурахами.

Мурашині алгоритми — це алгоритми поведінки систем з популяцій, створення яких засноване на результатах спостереження за колоніями мурах. Кооперація між індивідами у мурашиних алгоритмах досягається з використанням комунікаційних механізмів, що спостерігаються у колоніях живих мурах.

Отже, алгоритмічне моделювання поведінки мурах ґрунтується на концепції штучної стигмергії, яку М. Доріґо (Marco Dorigo) і Дж. Ді Каро (Gianni Di Caro) [8.12] визначали як «непряму» комунікацію, медіатором якої виступають численні модифікації станів середовища, доступні лише локально комунікаційним мурахам. Сутність моделювання поведінки мурах — знайти математичну модель, яка точно визначає стигмергетичні показники відповідних особин мурах. У контексті поведінки із заготівлі їжі штучні феромони відіграють роль стигмергетичних змінних.

Як розглянуто вище, мурахи мають здатність завжди знаходити найкоротші шляхи між мурашником і їжею. У міру свого руху від джерела їжі

до житла деяка кількість феромонів відкладається кожною мурахою. Наступні мурахи вибирають шляхи, ймовірно, на основі кількості феромонів. Чим вища концентрація феромонів, тим більше шансів, що відповідний шлях буде обрано. З часом короткі шляхи матимуть більші концентрації феромонів, оскільки ними мурахи повертаються швидше. Штучні феромони відображають характеристики реальних феромонів і вказують на «популярність» вирішення проблеми оптимізації.

Проста оптимізація колонії мурах (*SACO*)

Перший розроблений мурашиний алгоритм був моделлю мурашиної системи, і від того часу було створено кілька його модифікацій. Ці алгоритми моделюють більш складний процес прийняття рішення, ніж описаний вище. Розглянемо спочатку простий алгоритм оптимізації колонії мурах *SACO (Simple Ant Colony Optimization)*. *SACO* є алгоритмічною реалізацією описаного вище експерименту подвійного мосту Денеборга і використовується нами для ілюстрації основних компонентів і поведінки ACO-MH [8.13].

Розглянемо загальну задачу знаходження найкоротшого шляху між двома вузлами у графі $G = (V, E)$, де V — множина вершин (вузлів), E — матриця зв'язків між вузлами. Граф має $n_g = |V|$ вузлів. Довжина шляху L^k, побудованого мурахою, розраховується як кількість переходів по дорозі від початкового вузла до вузла призначення.

Приклад графа та обраного шляху показано на рис. 8.6. Довжина зазначеного маршруту дорівнює 2 (хоп-переходи). Концентрація феромонів τ_{ij} пов'язана з кожним ребром (i, j) графів.

В алгоритмі *SACO* кожному ребру присвоюють невелику випадкову величину, щоб вказати початкову концентрацію феромонів $\tau_{ij}(0)$. Якщо бути точними, у ребра немає ніякої концентрації феромонів на першому кроці. Мураха випадково вибирає, по якому ребру йти далі. Використовуючи просту логіку рішення, наведену в алгоритмі [8.15], для спрощення реалізації алгоритму ініціалізуємо концентрацію феромонів у кожній ланці невеликою випадковою величиною. Кількість мурах, розміщених у вихідному вузлі, так само $k = 1, \ldots, n_k$. На кожній ітерації *SACO* кожна мураха поступово будує шлях (рішення) до вузла призначення. На кожному вузлі кожна мураха приймає рішення, щоб визначити наступну ланку шляху. Якщо мураха k у певний час перебуває у вузлі i, то вона вибирає наступний вузол $j \in N_i^k$, спираючись на ймовірність переходу

$$p_{ij}^k(t) = \begin{cases} \dfrac{\tau_{ij}^{\alpha}(t)}{\sum_{j \in N_i^k} \tau_{ij}^{\alpha}(t)}, & \text{якщо } j \in N_i^k; \\ 0, & \text{якщо } j \notin N_i^k, \end{cases} \tag{8.17}$$

де N_i^k — множина можливих вузлів, пов'язаних із вузлом i щодо мурахи k.

Якщо для будь-якого вузла i та мурахи k виконується $N_i^k = \emptyset$, то попередній вузол для i включається в N_i^k. Зауважимо, що це може призвести до виникнення петель під час побудови шляху. Ці петлі видаляються, щойно буде досягнуто вузла призначення.

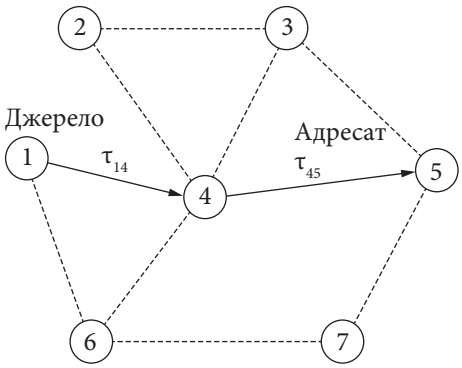


Рис. 8.6. Приклад графа та обраного шляху

У рівнянні (8.17) α — додатна константа, яка використовується для посилення впливу концентрації феромонів. Великі значення α зумовлюють надмірний вплив феромонів, особливо на стадії ініціалізації початкових випадкових концентрацій феромонів, що може спричинити швидку збіжність до неоптимальних шляхів.

Після того, як усі мурахи побудували повний шлях від початкового вузла до вузла призначення, а всі петлі було видалено, кожна мураха детерміновано простежує шлях до вихідного вузла і відкладає кількість феромонів

$$\Delta\tau_{ij}^k(t) \propto \frac{1}{L^k(t)} \tag{8.18}$$

для кожної ланки (i, j) відповідного шляху, де $L^k(t)$ — довжина шляху, побудованого мурахою k у момент часу t; знак $\propto$ означає пропорційність. Тобто

$$\tau_{ij}(t+1) = \tau_{ij}(t) + \sum_{k=1}^{n_k} \Delta\tau_{ij}^k(t), \tag{8.19}$$

де n_k — кількість мурах.

З використанням рівняння (8.13) загальна інтенсивність (тобто концентрація) феромонів буде пропорційною бажаності шляхів, що містять ланку (ij), на основі довжини відповідних шляхів. Кількість феромонів $\Delta\tau_{ij}^k(t)$ розраховується за рівнянням (8.18) і виражає якість відповідного рішення.

Для алгоритму *SACO* якість рішення (побудований шлях) визначається як величина, обернена довжині шляху (за кількістю переходів на шляху). Можна використати також будь-яку іншу метрику, наприклад вартість проходження шляху або довжину шляху.

Загалом, якщо $x^k(t)$ позначає рішення в момент часу t, то $f(x^k(t))$ виражає якість рішення. Якщо $\Delta\tau^k$ не пропорційне якості розв'язання і всі мурахи відкладають однакову кількість феромонів (тобто $\Delta\tau_{ij}^1 = \Delta\tau_{ij}^2 = \ldots = \Delta\tau_{ij}^{n_k}$), то це лише диференціальна довжина шляху, яка зміщує вибір у бік найкоротшого шляху — дуже нагадує спостереження Денеборга та його співавторів у експерименті «подвійний міст» [8.10]. Цей розгляд приводить до

двох основних форм оцінки рішень, що використовуються в мурашиних алгоритмах, а саме:

• неявні оцінки, де мурахи використовують диференціальну довжину шляху для ефекту зміщення напрямку пошуку інших агентів;

• явні оцінки, де кількість феромонів пропорційна деякому показнику якості побудованого рішення.

Якщо кількість феромонів, що зберігаються, обернено пропорційна якості рішення, то чим більше $f(x^k(t))$ (тобто найгірше побудоване рішення), тим менше $1/f(x^k(t))$, отже, менша кількість феромонів, що відкладаються у ланці. Таким чином, довший шлях зумовлює перетворення всіх ланок на менш бажані як компоненти остаточного рішення. Цей випадок справедливий для будь-якої міри якості f, що має бути мінімізована.

Будь-яку умову зупинки можна використати в алгоритмі (і в інших мурашиних алгоритмах, розглянутих далі), наприклад:

• зупинка, коли максимальну кількість ітерацій n_t було перевищено;

• зупинка, коли прийнятне рішення було знайдено і $f(x^k(t)) \le \varepsilon$;

• зупинка, коли всі мурахи (або більшість мурах) йдуть тим самим шляхом.

Простий алгоритм колонії мурах *SACO*.

Спочатку потрібно:

1) ініціалізувати $\tau_{ij}(0)$ малими випадковими значеннями;

2) покласти $t = 0$;

3) помістити n_k мурах на вихідному вузлі; $k = 1$.

Кроки алгоритму є такими.

1. Вибрати мураху k: $x_k(t) = \emptyset$.

2. Вибрати наступний вузол j на основі ймовірності, визначеної за рівнянням (8.17).

3. Додати зв'язок (i, j) до маршруту $x_k(t)$; $x_k(t) = x_k(t) \cup (i, j)$.

4. Перевірити умову: j = вузол призначення? Якщо так, то перейти на крок 5, інакше — на крок 2.

5. Видалити всі цикли з $x_k(t)$ та обчислити довжину шляху $f(x^k(t))$.

6. $k = k + 1$ і крок 1. Кроки 1—6 повторити для всіх мурах $k = 1, \ldots, n_k$.

7. Для кожного зв'язку (i, j) графа врахувати випаровування феромонів: зменшити величину феромону $\tau_{ij}(t)$, використовуючи рівняння (8.20).

8. Для кожної мурахи $k = 1, \ldots, n_k$ виконати такі кроки:

9. Для кожного зв'язку (i, j) маршруту $x_k(t)$ покласти $\Delta\tau_k = 1$.

10. Адаптувати τ_{ij}, використовуючи рівняння (8.19).

11. $t = t + 1$. Перевірити умову зупинки. Якщо не виконується, перейти на крок 1, поклавши $k = 1$, інакше — на крок 12.

12. Повернути шлях $x_k(t)$ з найменшим $f(x^k(t))$ як рішення.

Початковими експериментами з проблемою подвійного мосту виявлено, що мурахи швидко приходять до рішення і мало часу витрачають на дослідження альтернативних шляхів. Щоб змусити мурах виконувати дослідження довше і запобігти передчасній збіжності, феромонам на ланках «дозволено» «випаровуватися» у кожній ітерації алгоритму в міру збільшення з урахуванням побудованих шляхів. Нехай для кожної ланки (i, j)

$$\tau_{ij}(t) \leftarrow (1 - \rho)\tau_{ij}(t), \tag{8.20}$$

де $\rho \in [0; 1]$. Константа ρ визначає швидкість, з якою феромони випаровуються, змушуючи мурах «забути» раніше ухвалені рішення. Інакше кажучи, ρ визначає міру впливу історії пошуку. Для великих значень ρ феромони випаровуються швидко, тоді як малі значення ρ призводять до сповільнення швидкості випаровування. Чим інтенсивніше феромони випаровуються, тим випадковішим стає пошук, що сприяє поліпшенню розвідки. Для $\rho = 1$ пошук абсолютно випадковий.

У цей момент важливо підкреслити, що конструювання рішення є результатом кооперативної поведінки, яка виникає на основі простої поведінки окремих мурах: кожна мураха вибирає таку ланку свого шляху на основі інформації, наданої іншими мурахами у вигляді феромонів. У своїх експериментах М. Доріґо, Дж. Ді Каро виявили такі особливості [8.13]:

1) *SACO* добре працює для дуже простих графів, у яких найкоротший шлях вибирається найчастіше;

2) для великих графів пошук в алгоритмі погіршується, він стає менш стабільним і більш чутливим до параметрів вибору;

3) прагнення до найкоротшого шляху характерне для невеликої кількості мурах, тоді як занадто велика кількість мурах виявляє різну поведінку;

4) явище випаровування стає найважливішим для складніших графів. Якщо $\rho = 0$, тобто немає випаровування, то алгоритм не сходиться. Якщо феромони випаровуються надто швидко (використовується велике ρ), то алгоритм найчастіше сходиться до субоптимальних рішень складних проблем;

5) для малих α алгоритм загалом сходиться до найкоротшого шляху. Для складних проблем велике значення α призводить до погіршення збігу поведінки.

З цих досліджень простого алгоритму ACO стає очевидною важливість компромісу між фазами пошуку та застосування. У разі застосування такого механізму слід виявляти обережність, щоб уникнути використання мурахами великої концентрації феромонів та запобігти передчасному зупиненню алгоритму на неоптимальному рішенні, щоб мурахи були змушені досліджувати альтернативні шляхи. Далі розглянуто різні механізми вибору шляху, щоб збалансувати етапи пошуку та застосування алгоритму.

Простий алгоритм *ACO*, розглянутий раніше, показав деякий прогрес у пошуку найкоротшого шляху у графах. Ефективність алгоритму можна значно поліпшити за допомогою простих змін у ньому. Ці зміни крім евристичної інформації для визначення ймовірності вибору ланки стосуються пам'яті для запобігання циклам, а також різних правил оновлення кількості феромонів за допомогою використання локальної та/або глобальної інформації про навколишнє середовище. Далі наведено огляд ранніх мурашиних алгоритмів, що ґрунтуються на зазначених змінах простого алгоритму *ACO*.

Мурашина система (*AS*)

Перший мурашиний алгоритм був розроблений М. Доріґо і називався системою мурах (*AS*) [8.15]. Алгоритм *AS* покращує *SACO* завдяки зміні ймовірності переходів p_{ij}^k, яка використовує евристичну інформацію за допомогою додавання пам'яті та увімкнення списку табу. В *AS* імовірність переходу від вузла *i* до вузла *j* визначається за виразом

$$p_{ij}^k(t) = \begin{cases} \dfrac{\tau_{ij}^{\alpha}(t)\eta_{ij}^{\beta}(t)}{\sum_{u \in N_i^k} \tau_{iu}^{\alpha}(t)\eta_{iu}^{\beta}(t)}, & \text{якщо } j \in N_i^k; \\ 0, & \text{якщо } j \notin N_i^k, \end{cases} \tag{8.21}$$

де τ_{ij} — апостеріорна ефективність руху від вузла *i* до вузла *j*, яка виражається через інтенсивність феромонів відповідної ланки (*i, j*); η_{ij} — апріорна ефективність переходу від вузла *i* до вузла *j* (тобто привабливість, або бажаність, переходу), яка обчислюється за допомогою деякої евристики.

Імовірність переходу за рівнянням (8.21) відрізняється від *SACO* за рівнянням (8.17) двома аспектами.

1. Імовірність переходу, що використовується в алгоритмі *AS*, є балансом між інтенсивністю феромонів (тобто історією попередніх успішних ходів) τ_{ij} та евристичною інформацією (яка висловлює бажаність руху) η_{ij}. Це ефективно врівноважує компроміс між етапами пошук—застосування. Процес пошуку заохочує дії, які були знайдені в минулому і довели свою ефективність, використовуючи здобуті знання про простір пошуку. З іншого боку, щоб виявити такі дії, пошук має вивчати раніше невідомі дії (напрями руху), досліджуючи цим простір пошуку. Оптимальний баланс між пошуком та використанням досягається завдяки належному вибору параметрів α і β. Якщо $\alpha = 0$, то інформація про феромон не використовується, тобто нехтуємо попереднім досвідом пошуку. Потім пошук стає «жадібним» стохастичним. Якщо $\beta = 0$, то привабливістю (або потенційною вигодою) ходів нехтуємо і алгоритм пошуку стає подібним до алгоритму *SACO* і набуває пов'язаних із цим проблем.

Евристична інформація надає явної схльності до найбільш привабливих рішень і, отже, проблемно-залежних функцій. Наприклад, для завдань, де відстань (або вартість) шляху має бути зведена до мінімуму, маємо

$$\eta_{ij} = \frac{1}{d_{ij}},$$

де d_{ij} — відстань (або вартість) між вузлами i та j.

2. Набір визначає множину допустимих вузлів для мурахи k за його розміщення у вузлі i. Множина допустимих вузлів може містити лише найближчих сусідів вузла i. Крім того, для запобігання появі петель N_i^k може містити всі вузли, які мураха k ще не відвідала. Для цього зазвичай ведеться список табу для кожної мурахи. Як тільки мураха відвідала новий вузол, його буде додано до списку табу мурахи. Вузли списку табу видаляються з N_i^k після переконання, що вузол було відвідано більше одного разу.

В. Маніеццо (Vittorio Maniezzo), А. Колорні (Antonio Colorni) [8.1, 8.9] використовували іншу формулу оцінки ймовірності для визначення наступного вузла:

$$p_{ij}^k(t) = \begin{cases} \dfrac{\alpha\tau_{ij}(t) + (1-\alpha)\,\eta_{ij}}{\sum_{u\in N_i^k}(\alpha\tau_{iu}(t) + (1-\alpha)\,\eta_{iu}(t)}, \text{ якщо } j \in N_i^k; \\ 0, \text{ якщо } j \notin N_i^k, \end{cases} \tag{8.22}$$

де параметр α визначає відносну важливість концентрації феромонів $\tau_{ij}(t)$ щодо бажаності $\eta_{ij}(t)$ вибору ланки (i, j); імовірність p_{ij}^k — компроміс між бажаністю використання переходу (для малих α) та концентрацією феромонів (інтенсивністю). Ця формула усуває необхідність використання параметра β.

Випаровування феромонів реалізовано за рівнянням (8.20). Після завершення проходження шляху кожною мурахою феромон на кожній ланці оновлюється так:

$$\tau_{ij}(t+1) = \tau_{ij}(t) + \Delta\tau_{ij}(t), \tag{8.23}$$

де $\Delta\tau_{ij}(t) = \sum_{k=1}^{n_k})\Delta\tau_{ij}^k(t)$, $\Delta\tau_{ij}(t)$ — кількість феромонів, які зберігає мураха k у ланці (i, j) у момент часу t.

М. Доріго зі співавторами [8.15] розробили три варіанти алгоритму *AS*, кожен з яких відрізняється способом обчислення $\Delta\tau_{ij}^k(t)$.

1. Алгоритм *AS* «мурашиний цикл» (для задач мінімізації), використовує такий вираз:

$$\Delta\tau_{ij}^k(t) = \begin{cases} \dfrac{Q}{f(x^k(t))}, \text{ якщо ланка } (i, j) \text{ трапляється на шляху } x^k(t); \\ 0 \text{ — інакше.} \end{cases} \tag{8.24}$$

Для реалізації мурашиного циклу внесок феромонів обернено пропорційний якості $f(x^k(t))$ повного шляху, побудованого мурахою. Тому глобаль-

на інформація використовується для оновлення концентрації феромонів і Q є додатною константою.

2. Алгоритм *AS* «мурашиний цикл» (для задач максимізації) використовує вираз

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} Qf(x^k(t)), \text{ якщо ланка } (i, j) \text{ трапляється на шляху } x^k(t); \\ 0 \text{ — інакше.} \end{cases} \quad (8.25)$$

3. Алгоритм *AS* «мурашина щільність» використовує вираз

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} Q, \text{ якщо ланка } (i, j) \text{ трапляється на шляху } x^k(t); \\ 0 \text{ — інакше.} \end{cases} \quad (8.26)$$

Кожна мураха відкладає однакову кількість феромонів на кожній ланці побудованого шляху. Такий підхід суттєво спрощує підрахунок кількості мурах, які прямують до ланки (i, j). Чим вища інтенсивність руху ланкою, тим бажаніше, щоб ланка виявилася складовою остаточного рішення.

4. Алгоритм *AS* «кількість мурах» використовує вираз

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} \dfrac{Q}{d_{ij}}, \text{ якщо ланка } (i, j) \text{ трапляється на шляху } x^k(t); \\ 0 \text{ — інакше.} \end{cases} \quad (8.27)$$

У цьому разі для оновлення концентрації феромонів використовується лише локальна інформація d_{ij}. Ланки з нижчою вартістю стають бажаніши-ми. Якщо критерій становить відстань між вузлами, то вся кількість мурах алгоритму *AS* вибирає найкоротші зв'язки.

Алгоритм мурашиної системи. Під час ініціалізації процесу розміщення мурах диктується завданням, яке вирішують. Якщо мета полягає у знаходженні найкоротшого шляху між вихідним вузлом і вузлом призначення, всі n_k мурах розміщені у вихідному вузлі. З іншого боку, якщо метою є побудова найкоротшого гамільтонового шляху (тобто шляху, який з'єднує всі вузли лише один раз), то n_k мурах випадково будуть розподілені по всьому графу. Розміщуючи мурах на випадково обраних вузлах, досягаємо поліпшення дослідницьких властивостей алгоритму пошуку. Феромони ініціалізуються або константою τ_0, або невеликими випадковими значеннями в діапазоні $[0, \tau_0]$.

Спочатку потрібно:

1) покласти $t = 0$;

2) ініціалізувати усі параметри — α, β, ρ, Q, n_k, τ_0;

3) розмістити всіх мурах $1, \ldots, n_k$;

4) для кожного ребра (i,j) вибрати початкове значення $\tau_{ij}(t)$: $\tau_{ij}(t) \sim U(0;\tau_0)$, покласти $k = 1$.

Тоді основні кроки алгоритму матимуть такий вигляд.

1. $x_k(t) = \emptyset$; $i = 1$.

2. З поточного вузла i обираємо наступний вузол j з імовірністю, визначеною за рівнянням (8.21):

$$x_k(t) = x_k(t) \cup \{(i, j)\}.$$

3. Перевіряємо, чи не побудовано повний шлях. Якщо так, переходимо на наступний крок, інакше — на крок 2.

4. Обчислюємо довжину шляху $f(x^k(t))$.

5. $k = k + 1$. $k \le n_k$? Якщо так, то переходимо на крок 1, інакше — на крок 6.

6. Для кожного зв'язку (i, j) застосовуємо випаровування феромону, використовуючи рівняння (8.20).

Обчислюємо $\Delta\tau_{ij}(t)$, використовуючи рівняння (8.18).

7. Адаптуємо концентрацію феромону, використовуючи рівняння (8.19):

$$\tau_{ij}(t + 1) = \tau_{ij}(t) + \Delta\tau_{ij}(t); \quad t = t + 1.$$

І переходимо на крок 1 наступної ітерації, доки не виконається умова зупинки.

8. Вибираємо оптимальний маршрут $x_k(t)$: $f(x_k(t)) = \min\limits_i \{f(x_k(t))\}$.

Використовуючи описаний вище алгоритм, М. Доріґо, В. Маніеццо та А. Колорні [8.15] експериментували з трьома версіями алгоритму *AS*. З експериментів, які були зосереджені на вирішенні задачі комівояжера (як симетричні, так і асиметричні версії) та задачі квадратичного призначення, було зроблено висновок про те, що алгоритм *AS* «мурашиний цикл» є найкращим, оскільки використовує глобальну інформацію про якість рішення для визначення кількості відкладених феромонів. Для алгоритмів «мурашина щільність» та «кількість мурах» пошук не спрямований показником якості рішень.

М. Доріґо зі співавторами [8.15] запропонували також елітарну стратегію, де на додачу до підкріплення феромонів на основі рівняння (8.23) величина, пропорційна довжині найкращого шляху, додається до феромонів на всіх ланках, які є частиною найкращого шляху. Оновлене рівняння феромонів має вигляд

$$\tau_{ij}(t + 1) = \tau_{ij}(t) + \Delta\tau_{ij}(t) + n_e\Delta\tau^e_{ij}(t), \tag{8.28}$$

де

$$\Delta\tau^e_{ij}(t) = \begin{cases} \dfrac{Q}{f(\overline{x}(t))}, & \text{якщо ланка } (i, j) \text{ трапляється на шляху } x^k(t); \\ 0 \text{ — інакше.} \end{cases} \tag{8.29}$$

У рівнянні (8.28) n_e — число елітних мурах; $\overline{x}(t)$ — поточний найкращий маршрут з $f(\overline{x}(t)) = \{f(x^k(t))\}$.

Елітарна стратегія має на меті спрямовувати пошук усіх мурах на побудову рішення так, щоб воно містило ланки поточного найкращого маршруту.

8.6. Алгоритми колоній мурах

Система колонії мурах (*ACS*)

Ця система була розроблена М. Доріґо та Л.М. Гамбарделлою (Luca Maria Gambardella) для підвищення продуктивності алгоритму *AS* [8.14]. *ACS* відрізняється від *AS* за чотирма аспектами:

1) використовуються інші правила переходу;
2) визначаються інші правила оновлення феромонів;
3) вводиться локальне оновлення феромонів;
4) використовується список кандидатів на користь конкретних вузлів.

Розглянемо кожну із цих модифікацій.

Правило переходу *ACS*, також відоме як псевдовипадкове пропорційне правило дії, було розроблено для збалансування властивостей режимів пошуку та використання алгоритму. Мураха k, яка у певний час знаходиться у вузлі i, вибирає наступний вузол j, використовуючи правило

$$j = \begin{cases} \arg\max\{\tau_{iu}(t)\eta_{iu}^{\beta}(t)\}, \text{ якщо } r \le r_0; \\ J, \text{ якщо } r > r_0, \end{cases} \tag{8.30}$$

де $r \sim U(0, 1)$ і $r_0 \in [0, 1]$ — заданий користувачем параметр; $J \in N_i^k(t)$ є вузлом, випадково вибраним відповідно до ймовірності:

$$p_{ij}^k(t) = \frac{\tau_{ij}(t)\eta_{ij}^{\beta}(t)}{\sum_{u \in N_i^k} \tau_{iu}(t)\eta_{iu}^{\beta}(t)}, \tag{8.31}$$

де $N_i^k(t)$ — набір допустимих вузлів для відвідування.

Правило переходу у рівнянні (8.30) створює перекіс у бік вузлів, з'єднаних короткими ланками з великою кількістю феромонів. Параметр r_0 застосовується для балансування пошуку та використання: якщо $r \le r_0$, то алгоритм працює в режимі використання, віддаючи перевагу кращій ланці, якщо $r > r_0$ — в режимі дослідження. Важливо зазначити, що правило переходу *ACS* те саме, що і в алгоритмі *AS* при $r > r_0$, а також, що це правило використовує $\alpha = 1$, а отже, цей параметр вилучається з рівняння (8.31).

На відміну від алгоритму *AS* тільки глобально найкращій мурасі (наприклад, мурасі, яка побудувала найкоротший шлях $x^+(t)$) дозволено підвищувати концентрацію феромонів у ланках відповідного найкращого шляху. Феромони оновлюються за глобальним правилом:

$$\tau_{ij}(t+1) = (1 - \rho_1)\tau_{ij}(t) + \rho_1 \Delta\tau_{ij}(t), \tag{8.32}$$

де

$$\Delta\tau_{ij}(t) = \begin{cases} \dfrac{Q}{f(x^+(t))}, \text{ якщо } (i, j) \in x^+(t); \\ 0 \text{ — інакше,} \end{cases} \tag{8.33}$$

та $f(x^+)(t) = |x^+(t)|$ у разі знаходження найкоротших шляхів.

Глобальне правило оновлення *ACS* робить пошук більш спрямованим, заохочуючи мурах шукати на околицях не знайдене досі найкраще рішення. Ця стратегія сприяє процесу використання і застосовується після того, як усі мурахи побудували рішення.

М. Доріґо та Л.М. Гамбарделла [8.14] реалізували два методи вибору шляху $x^+(t)$:

1) найкращий на ітерації, де $x^+(t)$ є найкращим шляхом, знайденим протягом поточної ітерації t, який позначається як $\hat{x}(t)$;

2) глобально найкращий, де $x^+(t)$ є найкращим шляхом, знайденим починаючи з першої ітерації алгоритму, який позначається як $\tilde{x}(t)$.

Для глобально найкращого способу процес пошуку реалізується за допомогою більш глобальної інформації.

Випаровування феромонів також розглядається дещо інакше, ніж в алгоритмі *AS*. Враховуючи рівняння (8.32) за малих значень ρ_1 наявні концентрації феромонів на ланках випаровуються повільно, тоді як вплив найкращого маршруту слабшає. З іншого боку, за великих значень ρ_1 попередньо відкладені феромони випаровуються швидко, але вплив найкращого шляху посилюється. Ефект великих ρ_1 полягає в тому, що попереднім досвідом нехтують на користь пізнішого досвіду. Хоча значення ρ_1, як правило, фіксоване, стратегія, де ρ_1 регулюється динамічно від великих до малих значень, сприятиме розвідці у початкових ітераціях пошуку, тоді як увага фокусується на використанні найкращих знайдених шляхів на більш пізніх ітераціях.

На додачу до глобального правила оновлення *ACS* використовує локальне правило оновлення:

$$\tau_{ij}(t) = (1 - \rho_2)\tau_{ij}(t) + \rho_2\tau_0, \tag{8.34}$$

де ρ_2 також належить (0, 1) і τ_0 — невелика додатна константа.

Експериментальні результати з вирішення різних завдань комівояжера (*TSP*) показали, що вибір $\tau_0 = (n_G L)^{-1}$ дає добрі результати, при цьому n_G — число вузлів у графі G; L — довжина шляху, яку знаходять евристичним алгоритмом «найближчого сусіда» в задачі комівояжера *TSPs* (L може бути будь-яким грубим наближенням до оптимальної тривалості циклу).

Алгоритм *ACS* також переглядає значення околів множини, з якої вибирають наступний вузол. Множина вузлів $N_i^k(t)$ організована для складання списку вузлів-кандидатів. Ці вузли-кандидати є найкращими вузлами для відвідування насамперед. Нехай n_l, де $n_l < |N_i^k(t)|$, позначає кількість вузлів у списку кандидатів, найближчих (за відстанню або вартістю) до вузла i та впорядкованих за зростанням відстані. Коли вибрано наступний вузол, найкращий вузол додається до списку кандидатів. Якщо список кандидатів

порожній, то вузол j вибирається з частинки, що залишилася в $N_i^k(t)$. Вибір вузла, який є кандидатом, може бути заснований на рівнянні (8.31) або альтернативно на виборі найближчого кандидата $j \in N_i^k(t)$.

Алгоритм системи мурашиної колонії (*ACS*)

Маємо виконати низку підготовчих кроків:

1) покласти $t = 0$; ініціалізувати параметри β, ρ_1, ρ_2, r_0, τ_0, n_k;

2) розмістити всіх мурах $k = 1, \ldots, n_k$;

3) для кожного зв'язку (i, j) вибрати початкове значення $\tau_{ij}(t)$:

$$\tau_{ij}(t) \sim U(0, \tau_0);$$

$$\hat{x}(t) = \varnothing;$$

$$f(\hat{x}(t)) = 0;$$

вибрати початковий вузол $k = 1$.

Кроки основного алгоритму:

1. Для кожної мурахи $k = 1, \ldots, n_k : x_k(t) = \varnothing$.

2. У кожному вузлі i вибрати черговий вузол j зі списку кандидатів $N_i^k(t)$, якщо список кандидатів не порожній, використовуючи рівняння (8.30) та (8.31). Інакше вибрати не кандидата $j \in N_i^k(t)$: $x_k(t) = x_k(t) \cup \{(ij)\}$.

3. Застосувати локальну адаптацію за допомогою рівняння (8.34). Перевірити умови, чи побудовано повний шлях. Якщо так, то вирахувати $f(x^k(t))$; інакше — перейти на крок 2.

4. Перевірити умови $k < n_k$. Якщо так, то $k = k + 1$, перевести наступну ітерацію на крок 1.

5. $x = x_k(t)$: $f(x_k(t)) = \min\limits_i \{f(x_i(t))\}$. Обчислити $f(x)$.

6. Якщо $f(x) < f(\hat{x}(t))$, то $\hat{x}(t) = x$; $f(\hat{x}(t)) = f(x)$.

7. Для кожного зв'язку застосувати глобальну адаптацію, використовуючи рівняння (8.32).

8. Для кожного зв'язку (i, j) виконати:

$$\tau_{ij}(t + 1) = \tau_{ij}(t);$$

$$\hat{x}(t + 1) = \hat{x}(t);$$

$$f(\hat{x}(t + 1)) = f(\hat{x}(t)).$$

9. Перевірити умову зупинки, якщо не виконується, оновити $t = t + 1$ і перейти на крок 1, поклавши $k = 1$, інакше — на крок 10.

10. Вибрати $\hat{x}(t)$ як розв'язок задачі.

Для того, щоб застосувати алгоритм *ACS* для вирішення великих і складних проблем, М. Доріґо та Л.М. Гамбарделла додали локальні процедури пошуку для багаторазового покращення рішень.

Максимінна мурашина система (*MMAS*)

Алгоритм *AS* під час вирішення складних проблем виявляє передчасну стагнацію. За умов, що всі мурахи йдуть точно таким самим шляхом, передчасна стагнація виникає, коли фаза дослідження у мурах мала і вони занадто рано переходять на фазу застосування та швидко використовують високу концентрацію феромонів. М. Доріґо і Т. Штютцле (Thomas Stützle) запропонували максимінну мурашину систему (*MMAS*) для вирішення проблеми передчасної стагнації алгоритму *AS* [8.16]. Основна різниця між *MMAS* і *AS* у тому, що інтенсивність феромонів обмежена в задані інтервали часу. Додаткові відмінності полягають у тому, що тільки найкраща мураха може підвищити концентрацію феромонів, початкова концентрація феромонів має максимально допустиме значення і використовується механізм згладжування феромонів.

Глобальне оновлення феромонів подібне до наведеного в алгоритмі *ACS* рівнянням (8.32), де $\Delta\tau_{ij}(t)$ розраховується на основі одного з глобально найкращих шляхів або ітераційно найкращих шляхів. Перша версія *MMAS* зосереджена на ітераційно найкращому шляху, де $\bar{x}(t)$ є найкращим шляхом, знайденим під час поточної ітерації. Пізніші версії також містять глобально найкращі шляхи $\hat{x}(t)$, ґрунтуючись на таких стратегіях.

1. Використання тільки глобально оптимального шляху для визначення $\Delta\tau_{ij}(t)$. У цьому випадку $\hat{x}(t)$ — найкращий загальний шлях, знайдений з першої ітерації. Використовуючи лише глобально найкращий шлях, можна швидко сконцентрувати пошук навколо глобально найкращого рішення, обмежуючи цим пошук. Це завдання редукується, якщо використовується ітераційно найкращий шлях. Найкращі шляхи можуть значно змінюватися від однієї ітерації до іншої, забезпечуючи тим самим пошук.

2. Застосування змішаних стратегій, де обидва ітераційно найкращі і глобально найкращі шляхи використовуються для відновлення концентрації феромонів. Зазвичай ітераційно найкращі шляхи використовуються для пошуку, а глобально найкращий шлях — періодично. Крім того, частота, з якою використовується глобально оптимальний шлях для оновлення феромонів, збільшується з часом.

3. У точці стагнації всі концентрації феромонів $\tau_{ij}(t)$ створюються заново до максимально допустимого значення, після якого використовується лише ітераційно найкращий шлях для обмеженої кількості ітерацій, при цьому λ — коефіцієнт розгалуження ($\lambda = 0{,}05$), який застосовують для визначення точки стагнації. М. Доріґо і Л.М. Гамбарделла визначили середнє значення коефіцієнта розгалуження як ознаку розмірності простору пошуку [8.14]. Згодом середнє значення λ зменшується до малої величини у точці стагнації.

Протягом виконання алгоритму *MMAS* усі τ_{ij} обмежені в певному діапазоні. Для першої версії *MMAS* $\tau_{ij} \in [\tau_{\min}, \tau_{\max}]$ для всіх ланок (i, j), де оби-

два — τ_{min} і τ_{max} — проблемні статичні параметри. Якщо після застосування глобального правила оновлення $\tau_{ij}(t+1) > \tau_{max}$, то $\tau_{ij}(t+1)$ явно встановлюють таким, що дорівнює τ_{max}. З іншого боку, якщо $\tau_{ij}(t+1) < \tau_{min}$, то $\tau_{ij}(t+1)$ встановлюється як τ_{min}. Використання верхньої оцінки для концентрації феромонів допомагає уникнути появи стагнації — запобігає надмірному зростанню концентрації, що обмежує ймовірність швидкої збіжності до найкращого шляху. Наявність нижньої межі τ_{min} визначає ту перевагу, що всі ланки мають ненульову можливість бути внесеними в рішення. Справді, якщо τ_{min} і $n_{ij} < \infty$, то можливість вибору конкретної ланки як частини рішення ніколи не дорівнює нулю. Додатна нижня оцінка сприяє поліпшенню пошуку рішень.

Т. Штютцле (Thomas Stützle) і Х. Гус (Holger H. Hoos) [8.17] формально вивели, що максимальна концентрація феромонів асимптотично обмежена. Тобто для будь-якого τ_{ij} виконується рівність

$$\lim_{t\to\infty} \tau_{ij}(t) = \tau_{ij} \le \frac{1}{1-\rho}\frac{1}{f^*}, \tag{8.35}$$

де f^* — вартість (наприклад, довжина) теоретичного оптимального рішення.

Це дає теоретичне значення для верхньої межі τ_{max}. Оскільки оптимальне рішення, як правило, невідоме і, отже, f^* є невідомим, то *MMAS* ініціалізує оцінку для f^*, вважаючи $f^* = f(\hat{x}(t))$, де $f(\hat{x}(t))$ — вартість глобально найкращого шляху. Це означає, що максимальне значення концентрації феромонів змінюється, як тільки знайдено нову глобально найкращу позицію. Концентрація феромонів має верхню межу

$$\tau_{max}(t) = \frac{1}{1-\rho}\frac{1}{f(\hat{x}(t))} \tag{8.36}$$

і, отже, залежить від часу.

Т. Штютцле і Х. Гус отримали формально, що розумні значення для нижньої межі концентрації феромонів можна розрахувати за формулою

$$\tau_{min} = \frac{\tau\sqrt{\hat{p}}_{max}}{(n_G/2-1)\sqrt{\hat{p}}n_G}, \tag{8.37}$$

де $\hat{p}$ — ймовірність, з якою побудовано найкраще рішення; n_G — число вузлів у рішенні. Зауважимо, що за $\hat{p} < 1$ маємо $\tau_{min}(t) > 0$. Також зазначимо, що якщо $\hat{p}$ є занадто малим, то $\tau_{min}(t) > \tau_{max}(t)$. У таких випадках у *MMAS* встановлюють $\tau_{min}(t) = \tau_{max}(t)$, наслідком якого є ефект використання тільки евристичної інформації η_{ij}. Значення $\hat{p}$ є параметром користувача, який має бути оптимізований для кожної нової програми.

Під час кроку ініціалізації алгоритму *MMAS* усі концентрації феромонів ініціалізуються максимально допустимими значеннями τ_{max}.

Максимальний підхід, застосований до алгоритму *ACS*, називається *MMACS*. У цьому випадку чим більше значення r_0 (див. рівняння (8.30)), тим чіткіше має бути обрано межі концентрації феромонів. Найменше співвідношення $\tau_{\max_{\min}}$ має бути використано для великих r_0, щоб запобігти частому переважанню мурах зв'язку з високою концентрацією феромонів.

Навіть якщо концентрація феромонів затиснута в межах певного діапазону, стагнація все ще відбувається, хоча й рідше, ніж для алгоритму *AS*. Щоб вирішити цю проблему, використовують стратегію згладжування феромонів для зменшення різниці між високою та низькою їх концентраціями. У точці стагнації концентрація феромонів збільшується пропорційно різниці з максимальною оцінкою, тобто $\Delta\tau_{ij}(t) \propto (\tau_{\max}(t) - \tau_{ij}(t))$.

За допомогою цієї стратегії згладжування вища концентрація феромонів підкріплюється пропорційно менше, ніж слабші концентрації. Це збільшує шанси на обрання ланки з низькою концентрацією феромонів як частини шляху і тим самим підвищує дослідницьку здатність алгоритму.

Швидка мурашина система

Е. Тайллард (Éric D. Taillard) і Л.М. Гамбарделла розробили швидку мурашину систему (*FANT*) [8.18], зокрема, для вирішення квадратичної задачі про призначення (*QAP*). Основні відмінності між *FANT* та іншими алгоритмами *ACO* полягають у тому, що *FANT* використовує тільки одну мураху, також у ньому застосовують різні правила оновлення феромонів, до яких не входять випаровування.

Використання лише однієї мурахи значно зменшує обчислювальну складність. *FANT* застосовує як правило переходу рівняння (8.28), але за $\beta = 0$. Не використовується евристична інформація. Правило оновлення феромонів визначається рівнянням

$$\tau_{ij}(t+1) = \tau_{ij}(t) + w_1 \Delta\bar{\tau}_{ij}(t) + w_2 \Delta\hat{\tau}_{ij}(t), \tag{8.38}$$

де w_1 і w_2 — параметри для визначення відносного підкріплення за умови, що поточне та найкраще рішення на ітерації t знайдено на цей момент. Додані феромони розраховують за виразами

$$\Delta\bar{\tau}_{ij}(t) = \begin{cases} 1, \text{ якщо } (i,j) \in \bar{x}(t); \\ 0 \text{ — інакше;} \end{cases} \tag{8.39}$$

$$\Delta\hat{\tau}_{ij}(t) = \begin{cases} 1, \text{ якщо } (i,j) \in \hat{x}(t); \\ 0 \text{ — інакше;} \end{cases} \tag{8.40}$$

де $\bar{x}(t)$ і $\hat{x}(t)$ — відповідно кращий шлях, знайдений на ітерації t, і глобально найкращий шлях, знайдений від початку пошуку.

Феромони ініціалізуються як $\tau_{ij}(0) = 1$. Щойно знайшли новий $\hat{x}(t)$, всі феромони реініціалізуються до $\tau_{ij}(0) = 1$. Цей крок використовує пошук на межах глобально оптимального шляху $\hat{x}(t)$. Якщо в момент часу t буде знайдено рішення таке саме, що й за поточного глобально найкращого рішення, то значення w_1 збільшується на одиницю. Цей крок сприяє пошуку завдяки зменшенню внеску $\Delta\tau_{ij}(t)$, пов'язаного з найкращим глобальним шляхом.

Табу. Єва Рукс (Eva Rooks) і Омер Каї (Omer Kayi) [8.9] адаптували алгоритм *AS*, щоб увімкнути локальний пошук за допомогою пошуку табу для уточнення рішень, побудованих на кожній ітерації *AS*. На додачу до використання пошуку табу як локальної процедури пошуку глобальне правило оновлення змінилося так, що феромони кожної мурахи зберігаються на кожній ланці побудованого шляху і їх концентрація пропорційна якості шляху. Кожна мураха k оновлює концентрацію феромонів, використовуючи співвідношення

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \frac{\rho}{f(x^k(t))}\frac{f(\bar{x}(t)) - f(x^k(t))}{f(\hat{x}(t))}, \tag{8.41}$$

де $f(\bar{x}(t))$ — вартість найгіршого шляху, знайденого досі; $f(\hat{x}(t))$ — вартість найкращого шляху, знайденого до цього часу; $f(x^k(t))$ — вартість шляху, знайденого мурахою k.

Рівняння (8.41) застосовується для кожної мурахи k для кожної ланки $(i, j) \in x^k(t)$.

Алгоритм з урахуванням *AS*-рангів

Було запропоновано модифікацію алгоритму *AS* на основі рангів, яка полягає у таких змінах:

- дозволити лише найкращим мурахам оновлювати концентрацію феромонів на ланках глобально найкращого шляху;
- використовувати елітарних мурах;
- дозволити мурахам оновлювати концентрацію феромонів на основі рейтингу.

Для *AS*-рангу правило глобального оновлення змінюється за принципом:

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + n_e\Delta\hat{\tau}_{ij}(t) + \Delta\tau^r_{ij}(t), \tag{8.42}$$

де

$$\Delta\hat{\tau}_{ij}(t) = \frac{Q}{f(\hat{x}(t))};$$

$\hat{x}(t)$ — найкращий шлях, побудований до цього часу. Якщо використовується n_e елітних мурах і n_k мурах сортуються так, що $f(x^1(t)) \le f(x^2(t)) \le \ldots \le f(x^{n_k}(t))$, то

$$\Delta\tau^r_{ij}(t) = \sum_{\sigma=1}^{n_e}\Delta\tau^\sigma_{ij}(t), \tag{8.43}$$

де

$$\Delta\tau_{ij}^{\sigma}(t) = \begin{cases} \dfrac{(n_e - \sigma)Q}{f(x^{\sigma}(t))}, \text{ якщо } (i, j) \in x^{\sigma}(t); \\ 0 \text{ — інакше,} \end{cases} \tag{8.44}$$

а параметр σ вказує ранг відповідної мурахи.

Ця елітарна стратегія відрізняється від застосовуваної в алгоритмі *AS* тим, що внесок елітних мурах у концентрацію феромонів прямо пропорційний рейтингу продуктивності: чим кращий ранг (тобто менше σ), тим більший внесок.

Дерево пошуку *ANTS*

Наближене дерево недетермінованого пошуку (*ANTS*) було розроблено В. Маніеццо (Vittorio Maniezzo) та А. Карбонаро (Alessandro Carbonaro) [8.9] як розширення алгоритму *AS*. Алгоритм *ANTS* відрізняється від *AS:*

- розрахунком ймовірності переходу;
- правилом глобального поновлення;
- підходом, для уникнення стагнації.

Алгоритм *ANTS* використовує можливість переходу, як це визначено рівнянням (8.22). Набір N_i^k містить усі можливі переходи з поточного вузла *i*. Інтенсивність феромонів оновлюється після того, як усі мурахи завершили будівництво своїх шляхів. Концентрація феромонів оновлюється за допомогою рівнянь (8.21) та (8.23), але за виразом

$$\Delta\tau_{ij}^{k}(t) = \tau_0\left(1 - \frac{f(x^k(t)) - e}{\bar{f}(t) - e}\right), \tag{8.45}$$

де $f(x^k(t))$ — вартість відповідного шляху $x^k(t)$ мурахи k на ітерації t; $\bar{f}(t)$ — ковзне середнє за вартістю останніх $\hat{n}_t$ глобально найкращих рішень, знайдених за допомогою алгоритму.

Якщо $f(\hat{x}(t))$ означає вартість глобально найкращого рішення на ітерації t, то

$$\bar{f}(t) = \frac{\sum_{t'=t-\hat{n}_t}^{t} f(\hat{x}(t'))}{\hat{n}_t}. \tag{8.46}$$

Якщо $t < \hat{n}_t$, то ковзне середнє розраховується за наявними t найкращим рішенням. У рівнянні (8.45) «*e*» є нижньою межею вартості оптимального рішення.

Розрахунок кількості феромонів $\Delta\tau_{ij}^{k}$, що відкладаються на зберігання кожною мурахою, дає можливість ефективно уникнути передчасної стагнації: концентрація феромонів знижується, якщо вартість рішення мурахи нижча від ковзного середнього, в іншому випадку концентрація

феромонів збільшується. Механізм оновлення дає змогу розрізняти дрібні досягнення на кінцевій ітерації алгоритму пошуку та уникнути експлуатації на ранніх ітераціях.

Налаштування параметрів. Кожний з алгоритмів, розглянутих вище, використовує низку контрольних параметрів, що впливають на продуктивність алгоритмів. Тут продуктивність стосується якості знайдених рішень (якщо такі буде знайдено) і часу для досягнення цих рішень. Контрольні параметри алгоритмів колоній мурах *ACO* наведено в табл. 8.1 [8.9].

Для більшості алгоритмів *ACO* спільними є такі параметри:

1) n_k — *кількість мурах.* Починаючи з першого дослідження алгоритму *ACO*, було вивчено вплив кількості мурах на продуктивність. Очевидний вплив кількості мурах становить обчислювальну складність. Чим більше мурах використовується, тим більше шляхів має бути побудовано і більше феромонів має зберігатися. Наприклад, обчислювальна складність алгоритму *ACO* становить ($n_c\, n_G^2\, n_k$), де $n_c = n_t n_k$ — загальна кількість циклів; n_t — загальна кількість ітерацій; n_G — число вузлів у рішенні (за умови, що всі рішення мають однакову кількість вузлів).

Успіх алгоритмів *ACO* полягає у кооперативній поведінці кількох мурах. Через феромони, що зберігаються, мурахи діляться своїм досвідом і знаннями про простір пошуку з іншими мурахами. Чим менше мурах використовується, тим менші пошукові можливості дослідження алгоритмів і, отже, тим менше інформації про пошуковий простір доступно для всіх мурах. Невеликі значення n_k можуть призвести до вибору неоптимального рішення або ранньої стагнації. Надто багато мурах не обов'язково забезпечують добрий результат (як це було виявлено, наприклад, в алгоритмі *SACO*). За великих n_k значно більше часу витратиться для підтримання на високому рівні інтенсивності феромонів на хороших ланках, ніж на поганих.

М. Доріґо зі співавторами [8.13, 8.14] виявили, що вибір $n_k \approx n_G$ для вирішення завдання комівояжера *TSP* працює добре. Для алгоритму *ACS*

Таблиця 8.1

Контрольні параметри алгоритмів колоній мурах *ACO*

Параметр	Значення	Коментар
n_k	Кількість мурах	—
n_t	Максимальна кількість ітерацій	—
τ_0	Початкова кількість феромонів	—
P	Стійкість феромонів	ρ_1, ρ_2 для *ACS*
A	Інтенсифікація феромонів	$\alpha = 1$ для *ACS*
B	Евристична інтенсифікація	Не для *SACO, ANTS*

М. Доріґо та Л.М. Гамбарделла визначили, що оптимальну кількість мурах можна обчислити за формулою

$$n_k = \frac{\log(\varphi_1 - 1) - \log(\varphi_2 - 1)}{r_0 \log(1 - \rho_2)},$$

де $\varphi_1(\tau_0)$ — середня концентрація феромонів на ребрах останнього найкращого шляху до глобального оновлення та $\varphi_2(\tau_0)$ — після глобального оновлення.

На жаль, оптимальні значення φ_1 і φ_2 невідомі. А для задачі комівояжера в результаті емпіричного аналізу показано, що *ACS* працює найкраще, коли $(\varphi_1 - 1)/(\varphi_2 - 1) \approx 0{,}4$, що дає $n_k = 10$ [8.14].

Важливо, що конкретний алгоритм використовується для вирішення конкретного класу завдань. Значення n_k необхідно оптимізувати для різних алгоритмів та проблем;

2) n_t — *максимальна кількість ітерацій.* Легко побачити, що n_t відіграє важливу роль у забезпеченні якості рішень. Якщо n_t занадто мале, то мурахам може не вистачити часу, щоб досліджувати різні шляхи і зупинитися на одному з них. Якщо n_t завелике, можна зробити непотрібні обчислення;

3) τ_0 — *початкова концентрація феромонів.* Під час ініціалізації кроку всі феромони або ініціалізуються незмінною величиною τ_0 (для *MMAS* $\tau_0 = \tau_{max}$), або є випадковим значенням у діапазоні $[0, \tau_0]$. У разі вибору випадкових величин τ_0 надають невеликого додатного значення. Якщо для τ_0 приймають велике значення і випадкові значення вибирають з рівномірного розподілу, то концентрація феромонів у різних ланках може значно різнитися. Це може спричинити перекіс у бік ланок з високою початковою концентрацією як компонент остаточного рішення.

Є проста відповідь на досить важливе питання про те, як вибрати найкращі значення для контролю параметрів. Хоча ці параметри були предметом багатьох емпіричних досліджень, запропоновані значення та евристики для обчислення параметрів потрібно розглядати з обережністю. Вони призначені для конкретних алгоритмів і конкретних проблем і їх не слід приймати в загальному випадку. Для досягнення максимальної ефективності будь-якого з алгоритмів значення відповідних параметрів контролю має бути оптимізовано для конкретної задачі.

8.7. Застосування мурашиних алгоритмів у задачі комівояжера

Проблема комівояжера (*Travelling Salesman Problem, TSP*) — перша задача, до якої було застосовано алгоритм мурашиної колонії (*ACO*). Ця задача є комбінаторною — *NP*-важкою проблемою і найчастіше використовуваним завданням у літературі з тематики *ACO*.

Постановка задачі. Задано множину з n_π міст. Потрібно знайти цикл мінімальної довжини шляху, що проходить через усі міста по одному разу (гамільтонів цикл). Нехай π становить рішення як перестановку з міст {1, 2, …, n_π}, в якій π(*i*) — номер міста, відвіданого на *i*-му кроці. Тоді $\Pi(n_\pi)$ — множина всіх перестановок з {1, 2, …, n_π}, тобто простір пошуку. Формально завдання *TSP* визначається як знаходження оптимальної перестановки π^*:

$$\pi^\star = \mathrm{Arg}f(\pi), \quad \pi \in \Pi(n_\pi),$$

де $f(\pi) = \sum_{i,j=1}^{n_\pi} d_{ij}$; d_{ij} — відстань (або вартість проїзду) між містами *i* та *j*. Позначимо через $D = \left\| d_{ij} \right\|_{i,j=1,\ldots,n_\pi}$ матрицю відстаней.

Залежно від матриці *D* визначено дві версії задачі комівояжера:

- якщо $d_{ij} = d_{ji}$ для всіх $i, j = 1, \ldots, n_\pi$, — це симетричне завдання *TSP*;
- якщо $d_{ij} \neq d_{ji}$, — асиметричне завдання *TSP*.

Граф, що зображує задачу, задається трійкою $G = (V, E, D)$, де $V = [v_i]$ — множина вузлів (міст); $E = \{(i, j)\}$ — множина зв'язків (шляхів) між містами; *D* — матриця відстаней. Рішення є впорядкованою послідовністю $\pi = (i_1, i_2, ..., i_{n_\pi})$, що вказує на порядок відвідування міст.

«Евристична бажаність» відвідування міста *j* після міста *i* обчислюється так:

$$n_{ij}(t) = \frac{1}{d_{ij}(t)}.$$

Зазначення кроку часу *t* внесено у формулу для вирішення динамічних завдань, у яких відстані можуть змінюватися з часом.

Задача *TSP* містить два обмеження:

1) усі міста мають бути відвідані;

2) кожне місто має бути відвідане один раз.

Щоб забезпечити обмеження 2, створюють список табу для кожного окремого рішення, що містить усі міста, які вже відвідано. Нехай Γ_k позначає список табу для *k*-ї мурахи. Тоді $N_i^k(t) = V/\Gamma_k$ — множина міст, які ще не відвідано після досягнення міста *i*. Обмеження задовольняється вимогою, що кожне рішення містить рівно *n* міст і список табу.

Конструювання рішення. Мурахи розташовуються у випадково обраних містах, і кожна мураха крок за кроком будує рішення, використовуючи перехідні ймовірності будь-якого з описаних алгоритмів.

Локальний пошук. Найпопулярніші пошукові евристики в цій задачі —2-opt та 3-opt. Обидві вони містять обмін зв'язками, доки не буде знайдено локальний мінімум. Евристика 2-opt робить два розриви у маршруті і рекомбінує зв'язки між відповідними вузлами єдино можливим способом. Якщо вартість маршруту зменшилася, ця зміна приймається. Евристику 2-opt показано на рис. 8.7, *а*.

Евристика 3-opt розриває маршрут по трьох зв'язках і рекомбінує зв'язки між вузлами, щоб утворити два альтернативні маршрути. Найкращий маршрут з оригіналу та двох альтернативних маршрутів зберігається. Роботу евристики 3-opt показано на рис. 8.7, *б*. Для асиметричного завдання *TSP* використовують варіацію евристики 3-opt, щоб дозволити лише зміни, які змінюють порядок відвідання інших міст.

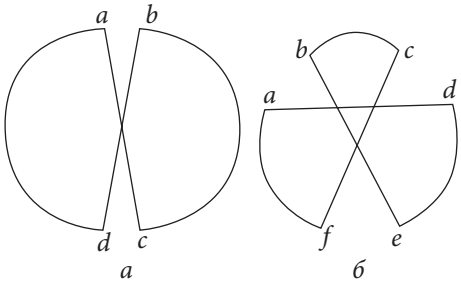


Рис. 8.7. Робота евристик: *а* — 2-opt; *б* — 3-opt

Мурашиний алгоритм для вирішення задачі комівояжера. Застосуємо пропонований мурашиний алгоритм для вирішення задачі комівояжера [8.1]. Спочатку розглянемо випадок, коли ребра графа завдання комівояжера неорієнтовані, тобто вартості прямих і зворотних шляхів із двох сусідніх вершин однакові.

Передумови алгоритму. Кожна вершина графа містить три множини: множину мурах для подальшого пересування — *FD*, множину мурах для повернення в гніздо після успішного обходу всього маршруту без «тупиків» і петель — *BK* і множину мурах, яким не вдалося обійти маршрут без «тупиків» і петель, — *BD*.

Перебуваючи у конкретній вершині, мурасі необхідно прийняти рішення, по якій дузі переходити до наступної вершини. Нехай n — загальна кількість дуг, що виходять з цієї вершини і йдуть до вершин, які мураха ще не відвідувала. Тоді для ухвалення рішення мурахою пропонується такий показник:

$$D_i = c_1 \cdot \left(1 - \frac{L_i}{\sum_{j=1}^{n} L_j}\right) + c_2 \cdot \frac{\tau_i}{\sum_{j=1}^{n} \tau_j} + c_3 \cdot U(0, 1), \tag{8.47}$$

де $i = \underline{1, \ldots, n}$ — порядковий номер дуги, що має зазначену якість; L_i — вартість переходу мурахи по цій дузі (або довжина шляху); τ_i — концентрація феромону на цій дузі; $U(0, 1)$ — генератор випадкових чисел, рівномірно розподілених на відрізку [0; l].

Величини c_1, c_2 і c_3 мають залежати від номера поточної ітерації t, в цьому алгоритмі їх пропонується використовувати в такій формі:

$$c_1 = a_1 \frac{M - t}{M}; \tag{8.48}$$

$$c_2 = \frac{t}{M}; \tag{8.49}$$

$$c_3 = a_2 \frac{M - t}{M}. \tag{8.50}$$

У формулах (8.48)—(8.50) використано позначення: M — деяка константа, яка до того ж задовольняє умову $M > t$; a_1 і a_2 — константи, що задовольняють умови $a_1 \geq 0$, $a_2 \geq 0$, $a_1 + a_2 = 1$. Неважко помітити, що $c_1 + c_2 + c_3 = 1$.

Мураха переходить на наступну вершину з найбільшим значенням показника D_i.

Останньою передумовою алгоритму є переміщення мурах у всі вершини графа. Це еквівалентно їх переміщенню в множині *FD* кожної з вершин графа.

Опис алгоритму

1. Кожна мураха з множини *FD* у кожній вершині шукає такі ребра, які ведуть до вершин, де вона ще не була. Якщо пошук завершується успішно, то перехід на крок 3, інакше — на крок 2.

2. Якщо пройдено всі вершини, мураха переміщується до множини *BK* цієї вершини, інакше — до множини *BD*.

3. Для дуг, що ведуть до вершин, які мураха ще не відвідувала, розраховується показник D_i з формули (8.47). Для дуги з найбільшим значенням показника кількість феромону збільшується на 1 і мураха переміщується до множини *FD* вершини на другому кінці дуги.

4. Усі мурахи з множини *BK* всіх вершин переміщуються в множини *BK* попередніх відвіданих вершин, при цьому збільшуючи кількість феромону в дугах, по яких вони переходять, на 1. Якщо вершина, в яку має перейти мурашка, — її мурашник, то вона розміщується в множину *FD*.

5. Усі мурахи з множини *BD* усіх вершин переміщуються в множини *BD* попередніх відвіданих вершин, при цьому зменшуючи кількість феромону в дугах, по яких вони переходять, на 1. Якщо вершина, в яку має перейти мурашка, — її мурашник, вона розміщується в її множину *FD*.

6. Перевірка умови перевищення допустимої кількості ітерацій. Якщо число ітерацій перевищено — завершення роботи алгоритму, інакше — крок 1.

Примітка 1. Як випливає з алгоритму, кожна мураха повинна запам'ятовувати послідовність відвіданих нею вершин і на зворотному шляху видаляти вершини з кінця цієї послідовності.

Примітка 2. У разі орієнтованих ребер алгоритм не змінюється.

Опис експериментів. У процесі експериментів досліджували три варіанти алгоритму [8.1]:

1) за однакової зміни величини феромону в усіх ребрах на 1;

2) за локальної зміни величини феромону на ребрах обернено пропорційно довжині (вартості) ребра;

3) у разі глобальної зміни величини феромону на всіх ребрах (обернено пропорційно довжині (ціні) шляху).

В експериментах варіювалася кількість мурах, що розміщуються у вузлах мережі: 1—5, 10, 30, 50. Спочатку мурахи розміщувалися в усіх вузлах мережі. У результаті експериментів генерувалися різні топології мережі, варіювалося число ітерацій і фіксувалося число ітерацій, у яких досягався оптимальний шлях.

Деякі результати експериментів наведено далі [8.1]. У табл. 8.2—8.13 «0» означає, що цикл не було знайдено; «1» — знайдено неоптимальний цикл; «2» — знайдено оптимальний цикл.

8.8. Аналіз результатів експериментів

1. Збільшення числа мурах у вузлах мережі від 1 до 2—5 сприяє поліпшенню характеристик алгоритму і досягненню мінімуму за меншої кількості ітерацій. Подальше збільшення числа мурах (до 10, 30, 50) не дає ефекту. Так, з однією мурахою оптимальний маршрут у низці експериментів для алгоритму за однакової зміни кількості феромону знаходили за 1000 ітерацій, з двома — за 250—500 ітерацій, а з трьома і більше мурахами — за 50—100 ітерацій (табл. 8.2—8.13).

2. Використання алгоритмів з локальною зміною кількості феромону покращує характеристики алгоритму порівняно з однаковою зміною, а в деяких випадках алгоритм з однаковою зміною концентрації феромону не досягає мінімуму і за 1500 ітерацій (з однією мурахою).

3. Використання алгоритму з глобальною зміною кількості феромону в середньому скорочує число ітерацій для досягнення мінімуму порівняно з локальною зміною феромону (рис. 8.8—8.11). Разом з тим, у низці експериментів локальний алгоритм виявився найкращим (див. рис. 8.11).

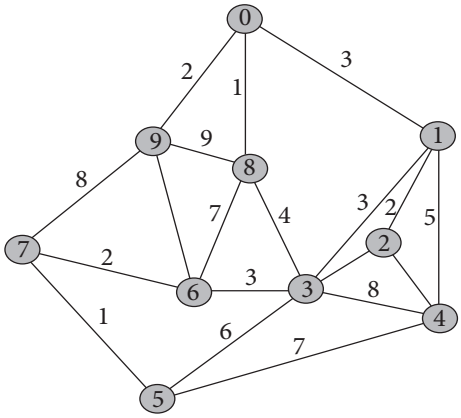


Рис. 8.8. Топологія 1

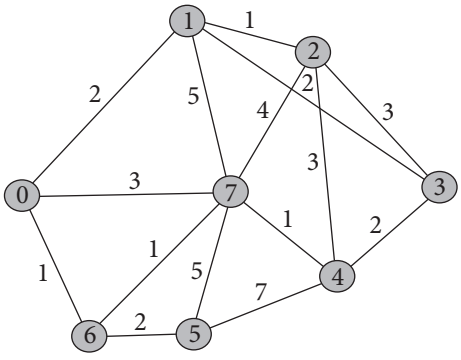


Рис. 8.9. Топологія 2

Таблиця 8.2

Результати експерименту за однакової зміни величини феромонів на кожній дузі на 1

Номер ітерації	Число мурах у вузлах перед запуском алгоритму							
	1	2	3	4	5	10	30	50
5	0	0	0	0	0	2	2	2
50	0	0	2	2	2	2	2	2
100	0	2	2	2	2	2	2	2
250	0	2	2	2	2	2	2	2
500	2	2	2	2	2	2	2	2
1000	2	2	2	2	2	2	2	2
1500	2	2	2	2	2	2	2	2

Таблиця 8.3

Результати експерименту за локальної зміни величини феромону

Номер ітерації	Число мурах у вузлах перед запуском алгоритму							
	1	2	3	4	5	10	30	50
5	0	0	0	0	0	0	0	0
50	0	2	2	2	2	2	2	2
100	2	2	2	2	2	2	2	2
250	2	2	2	2	2	2	2	2
500	2	2	2	2	2	2	2	2
1000	2	2	2	2	2	2	2	2
1500	2	2	2	2	2	2	2	2

Таблиця 8.4

Результати експерименту за глобальної зміни кількості феромону

Номер ітерації	Число мурах у вузлах перед запуском алгоритму							
	1	2	3	4	5	10	30	50
5	0	0	0	0	0	0	0	0
50	0	2	2	2	2	2	2	2
100	2	2	2	2	2	2	2	2
250	2	2	2	2	2	2	2	2
500	2	2	2	2	2	2	2	2
1000	2	2	2	2	2	2	2	2
1500	2	2	2	2	2	2	2	2

Таблиця 8.5

Результати експерименту за однакової зміни величини феромону на 1

Номер ітерації	Число мурах у вузлах перед запуском алгоритму							
	1	2	3	4	5	10	30	50
5	0	0	0	0	0	0	0	0
50	0	0	0	0	0	2	0	0
100	0	0	0	0	0	2	2	2
250	0	0	0	2	2	2	2	2
500	0	2	0	2	2	2	2	2
1000	0	2	2	2	2	2	2	2
1500	1	2	2	2	2	2	2	2

Таблиця 8.6

Результати експерименту за локальної зміни кількості феромону

Номер ітерації	Число мурах у вузлах перед запуском алгоритму							
	1	2	3	4	5	10	30	50
5	0	0	0	0	0	0	0	0
50	0	0	0	0	0	0	0	0
100	0	0	0	0	0	0	0	0
250	0	0	0	0	0	0	0	0
500	2	2	0	0	0	0	2	2
1000	2	2	2	2	2	2	2	2
1500	2	2	2	2	2	2	2	2

Таблиця 8.7

Результати експерименту за глобальної зміни величини феромону

Номер ітерації	Число мурах у вузлах перед запуском алгоритму							
	1	2	3	4	5	10	30	50
5	0	0	0	0	0	0	0	1
50	0	0	0	0	1	2	2	2
100	0	0	0	0	2	2	2	2
250	0	2	2	2	2	2	2	2
500	2	2	2	2	2	2	2	2
1000	2	2	2	2	2	2	2	2
1500	2	2	2	2	2	2	2	2

Таблиця 8.8

Результати експерименту за однакової зміни величини феромону

Номер ітерації	Число мурах у вузлах перед запуском алгоритму							
	1	2	3	4	5	10	30	50
5	0	0	0	0	0	0	0	0
50	0	0	2	0	2	2	0	2
100	0	0	2	2	2	2	2	2
250	0	2	2	2	2	2	2	2
500	0	2	2	2	2	2	2	2
1000	2	2	2	2	2	2	2	2
1500	2	2	2	2	2	2	2	2

Таблиця 8.9

Результати експерименту за локальної зміни величини феромону

Номер ітерації	Число мурах у вузлах перед запуском алгоритму							
	1	2	3	4	5	10	30	50
5	0	0	0	0	0	0	0	0
50	0	2	2	2	0	0	2	0
100	0	2	2	2	0	0	2	0
250	0	2	2	2	2	2	2	0
500	2	2	2	2	2	2	2	2
1000	2	2	2	2	2	2	2	2
1500	2	2	2	2	2	2	2	2

Таблиця 8.10

Результати експерименту за глобальної зміни величини феромону

Номер ітерації	Число мурах у вузлах перед запуском алгоритму							
	1	2	3	4	5	10	30	50
5	0	0	0	1	0	0	0	0
50	0	0	2	2	2	2	2	2
100	0	0	2	2	2	2	2	2
250	0	2	2	2	2	2	2	2
500	0	2	2	2	2	2	2	2
1000	2	2	2	2	2	2	2	2
1500	2	2	2	2	2	2	2	2

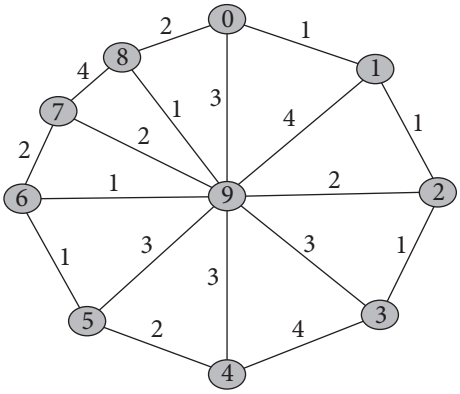


Рис. 8.10. Топологія 3

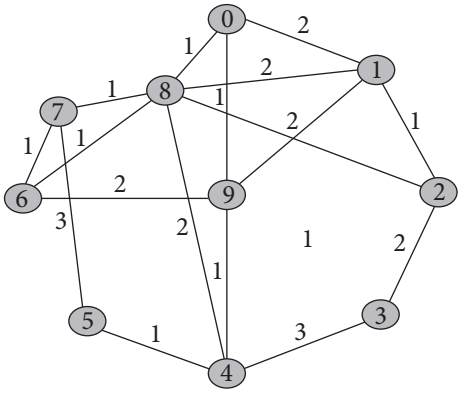


Рис. 8.11. Топологія 4

Таблиця 8.11

Результати експерименту за однакової зміни величини феромону

Номер ітерації	Число мурах у вузлах перед запуском алгоритму							
	1	2	3	4	5	10	30	50
5	0	0	0	0	0	0	0	0
50	0	0	0	0	0	2	2	2
100	0	0	0	2	0	2	2	2
250	0	0	2	2	2	2	2	2
500	0	2	2	2	2	2	2	2
1000	2	2	2	2	2	2	2	2
1500	2	2	2	2	2	2	2	2

Таблиця 8.12

Результати експерименту за локальної зміни кількості феромону

Номер ітерації	Число мурах у вузлах перед запуском алгоритму							
	1	2	3	4	5	10	30	50
5	0	0	0	0	0	0	0	2
50	2	2	2	2	2	2	2	2
100	2	2	2	2	2	2	2	2
250	2	2	2	2	2	2	2	2
500	2	2	2	2	2	2	2	2
1000	2	2	2	2	2	2	2	2
1500	2	2	2	2	2	2	2	2

Таблиця 8.13

Результати експерименту за глобальної зміни величини феромону

Номер ітерації	Число мурах у вузлах перед запуском алгоритму							
	1	2	3	4	5	10	30	50
5	0	0	0	0	0	0	0	0
50	0	0	0	2	2	2	2	2
100	0	0	0	2	2	2	2	2
250	0	0	2	2	2	2	2	2
500	2	2	2	2	2	2	2	2
1000	2	2	2	2	2	2	2	2
1500	2	2	2	2	2	2	2	2

Отже, результати експериментів показали, що збільшення кількості мурах у вузлах мережі від 1 до 3—5 значно скорочує кількість ітерацій, необхідних для знаходження оптимального рішення. Якщо для однієї мурахи алгоритму потрібно до 1000 ітерацій, то зі збільшенням їх кількості до 2—5, оптимальний шлях знаходиться у 250—100 ітерацій. Однак подальше збільшення кількості мурах до 10, 30 чи 50 не дало значного покращення, що свідчить про наявність певного порогу ефективності.

Дослідження також продемонстрували, що спосіб зміни кількості феромону суттєво впливає на ефективність алгоритму. Використання локальної адаптивної зміни феромону дає змогу досягти кращих результатів порівняно з алгоритмом, що змінює феромон рівномірно. У випадках, коли застосовувалася однакова зміна концентрації феромону, алгоритм інколи не міг знайти оптимальне рішення навіть за 1500 ітерацій.

Ще одним важливим фактором виявилося використання глобальної зміни кількості феромону, яке в середньому скорочує кількість ітерацій для знаходження мінімального шляху порівняно з локальним підходом. Проте у низці експериментів саме локальний алгоритм показав найкращий результат, що може свідчити про залежність ефективності методів від структури конкретної задачі.Отже, результати експериментів підтвердили ефективність адаптивних стратегій оновлення феромону та важливість оптимального вибору кількості мурах для підвищення продуктивності алгоритму. Визначено, що найкращий баланс між швидкістю і точністю пошуку досягається для 3—5 мурах у вузлах мережі та за використання комбінованих підходів до оновлення феромону. Це відкриває можливості для подальшого вдосконалення алгоритму, його адаптації до складніших топологій та застосування у реальних оптимізаційних задачах.

Висновки

Розділ 8 присвячений одному з найцікавіших напрямів в обчислювальному інтелекті — так званим ройовим алгоритмам обчислювального інтелекту. На розроблення цих алгоритмів вчених надихнули спостереження за поведінкою соціальних тварин і комах. Рій — це група індивідів (частинок), які відокремлено діють за простими алгоритмами, але під час взаємодії в колективі виявляють складну поведінку та колективний інтелект. Це мурахи, бджоли, птахи та інші види соціально активних популяцій тварин. Наприклад, мурахи діють за дуже простими алгоритмами, але в колективі вони будують житло, знаходять їжу, ховають мертвих та виявляють інші зразки інтелектуальної поведінки без спеціального центрального координатора, а завдяки взаємодії та обміну інформацією.

У результаті спостережень за поведінкою соціально активних комах і птахів було розроблено моделі та алгоритми ройової оптимізації. Було побудовано моделі руху частинок для вирішення задач оптимізації. Ці моделі містять когнітивну компоненту, що використовує інформацію про найкраще минуле самої частинки, та соціальну компоненту, що містить інформацію про найкраще досягнуте рішення для всіх частинок рою. Розроблено алгоритми глобальної ройової оптимізації та локальної оптимізації. В алгоритмах глобальної оптимізації всі частинки рухаються в напрямку найкращої позиції, знайденої всіма частинками рою за весь попередній час, та найкращої позиції, знайденої самою частинкою. В алгоритмах локальної ройової оптимізації замість найкращої позиції всього рою обирається найкраща позиція (рішення) групи сусідів частинки.

Розглянуто приклади демонстрації руху частинок рою в задачах пошуку глобального мінімуму деякої функції в алгоритмі глобальної ройової оптимізації та локальної оптимізації. Описано проблему вибору оптимальних параметрів алгоритмів РО, а також гібридні алгоритми РО, що є комбінацією алгоритмів РО та генетичних алгоритмів, їх властивості та приклади застосування.

Друга частина цього розділу присвячена докладному опису та аналізу мурашиних алгоритмів РО на прикладі поведінки мурах під час пошуку їжі. Під час свого руху в напрямку знаходження їжі кожна мураха відкладає феромони вздовж усього шляху, а потім повертається до житла. Якщо ж є декілька шляхів до джерела їжі, спочатку кожна мураха з однаковою ймовірністю обирає різні шляхи і однакова кількість мурах рухається усіма шляхами. Але потім усе більше мурах починають рухатися коротшим шляхом — і, отже, знаходить найкоротший шлях, на якому зростає кількість феромонів. Розроблено різні алгоритми руху мурах у напрямку мети (їжі).

Було розроблено простий алгоритм оптимізації *SACO (Simple-Ant Colony Optimization)* і досліджено його властивості та ефективність у досягненні мети. У подальшому впроваджено більш досконалий та ефективний алгоритм мурашиної системи *AS (Ant System)*. Він містить інше правило оновлення феромонів і має декілька варіантів метрики вибору наступної ланки переходів: мурашина щільність, кількість мурах та мурашиний цикл. Далі було розроблено мурашиний алгоритм колоній мурах *ACS,* який відізняється від *AS* іншими правилами оновлення феромонів та вибору переходу.

З метою підвищення швидкості роботи мурашиного алгоритму розроблено алгоритм *FANT* (швидка мурашина система). Розглянуто цікаву комбінаторну задачу на графах — задачу комівояжера і проведено дослідження ефективності її рішення за різними варіантами мурашиних алгоритмів для різної кількості мурах: 1,2—5, 10, 30, 50, та різних стратегій зміни феромонів: за однакової зміни кількості феромону на всіх переходах у маршруті, за локальної зміни та глобальної зміни кількості феромонів на всіх переходах найкращого шляху. Змінювали кількість ітерацій (від 100 до 1500), структуру графів. Наведено результати експериментів та проведено аналіз різних алгоритмів і стратегій зміни кількості феромонів, а також числа ітерацій для вирішення задач за різної кількості мурах.

СПИСОК ЛІТЕРАТУРИ

8.1. Zgurovsky M.Z., Zaychenko Yu.P. Fundamentals of computational intelligence: System approach. Springer, 2016. 275 p.

8.2. Zgurovsky M.Z., Zaychenko Yu.P. Big Data: Conceptual Analysis and Applications. Springer Nature Switzerland AG, 2019. 306 p.

8.3. Kennedy J., Eberhart R. Particle Swarm Optimization. *Proc. IEEE Intern. Conf. on Neural Networks*. Perth, 1995. P. 1942—1948.

8.4. Kennedy J., Eberhart R.C., Shi Y. Swarm Intelligence. Morgan Kaufmann, 2001. 512 p. URL: https://archive.org/details/swarmintelligenc0000kenn

8.5. Koay C.A., Srinivasan D. Particle Swarm Optimization-Based Approach for Generator Maintenance Scheduling. *Proc. IEEE Swarm Intelligence Symp*. 2003. P. 167—173.

8.6. Eberhart R.C., Shi Y. Particle Swarm Optimization: Developments, Applications and Resources. *Proc. IEEE Congr. on Evolutionary Comp*. 2001. Vol. 1. P. 27—30.

8.7. Eberhart R.C., Shi Y. Tracking and Optimizing Dynamic Systems with Particle Swarms. *Proc. IEEE Congr. on Evolutionary Comp*. 2001. Vol. 1. P. 94—100.

8.8. Eshelman L.J., Schaffer J.D. Real-Coded Genetic Algorithms and Interval Schemata. *Found. of Genet. Algorithms*. 1993. Vol. 2. P. 187—202.

8.9. Engelbrecht A. Computational Intelligence. An Introduction / Sec. Edition. John Wiley & Sons, Ltd., 2007. 630 p.

8.10. Deneubourg J.-L., Aron S., Goss S., Pasteels J.-M. The Self-Organizing Exploratory Pattern of the Argentine Ant. *Insect Behavior*. 1990. Vol. 3. P. 159—168.

8.11. Dorigo M., Bonabeau E., Theraulaz G. Ant Algorithms and Stigmergy. *Future Generation Comp. Syst*. 2000. 16, No. 9. P. 851—871.

8.12. Dorigo M., Stützle T. Ant Colony Optimization. MIT Press, 2004. 328 p.

8.13. Dorigo M., Di Caro G. The Ant Colony Optimization Meta-Heuristic. *New Ideas Optim*. 1999. P. 11—32.

8.14. Dorigo M., Gambardella L.M. Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. *IEEE Trans. Evolutionary Comp*. 1997. Vol. 1, No. 1. P. 53—66

8.15. Dorigo M., Maniezzo V., Colorni A. Ant System: Optimization by a Colony of Cooperating Agents. *IEEE Trans. Syst. Man and Cybernet*. 1996. Part B. Vol. 26, No. 1. P. 29—41.

8.16. Dorigo M., Stützle T. An Experimental Study of the Simple Ant Colony Optimization Algorithm. *WSES Intern. Conf. on Evolutionary Comp*. 2001. P. 253—258.

8.17. Stützle T., Hoos H. MAX-MINo. Ant System. *Future Generation Comp. Syst*. 2000. Vol. 16, No. 8. P. 889—914.

8.18. Taillard E.D. FANT: Fast Ant System. *Techn. Rep. IDSIA 46—98*. IDSIA, Lugano: Switzerland, 1998.

РОЗДІЛ 9

ГЛИБОКІ НЕЙРОННІ МЕРЕЖІ. АЛГОРИТМИ ГЛИБОКОГО НАВЧАННЯ

9.1. Автоасоціатори. Автокодувальники

9.2. Машини Больцмана (БМ)

9.3. Алгоритм навчання «контрастна дивергенція» (КД–k)

9.4. Стекові мережі автоасоціаторів

9.5. Глибоке мережеве навчання

9.6. Алгоритми оптимізації функції вартості
в машинному навчанні для НМ

9.7. Регуляризація глибокого навчання

Висновки

Список літератури

Одним із сучасних і ефективних інструментів для аналізу великого обсягу даних є глибокі нейронні мережі [9.1—9.3]. Сьогодні теорія і практика машинного навчання зазнають глибокої трансформації завдяки значному поширенню мереж глибокого навчання, які становлять третє покоління нейронних мереж. На відміну від класичних нейронних мереж (другого покоління) нові парадигми навчання 80—90-х років XX ст. сприяють усуненню деяких проблем, що перешкоджають успішному застосуванню традиційних нейронних мереж. Нейронні мережі, навчені за допомогою алгоритмів глибокого навчання, не тільки перевершили за точністю найкращі альтернативні підходи, а й у деяких випадках продемонстрували розуміння змісту вхідної інформації (розпізнавання зображень, аналіз тексту та інші задачі).

Найуспішніші промислові системи комп'ютерного зору та розпізнавання мови побудовані на глибоких мережах, а гіганти IT-індустрії, як-от *Apple, Google* та *Microsoft*, створили великі дослідницькі команди, що займаються глибоким навчанням. Термін «глибока мережа» означає велику нейронну мережу з багатьма прихованими шарами нейронів [9.3, 9.4]. Глибоке навчання є набором методів і прийомів для навчання складних нейронних мереж (НМ) з багатьма шарами. Для таких мереж традиційні алгоритми машинного навчання, розроблені для звичайних НМ, стали недоцільними через деякі недоліки, зокрема проблему зникнення та вибуху градієнта в алгоритмі зворотного поширення [9.2, 9.5]. Тому багато які сучасні нейронні мережі, що застосовуються для розпізнавання 3D-зображень та автоматичного розпізнавання мови, потребували розроблення нових ефективних методів навчання, які дістали назву «глибоке навчання» (*deep learning*).

Однак найістотнішим недоліком мереж глибокого навчання є проблема визначення їх правильної структури та добору адекватної кількості шарів.

9.1. Автоасоціатори. Автокодувальники

Впровадження глибокого навчання призвело до розробки спеціальної структури навчання, заснованої на застосуванні так званих автоасоціаторів (АА).

Основне завдання автоасоціатора — отримати на виході максимально точне відображення вхідного вектора (шаблона).

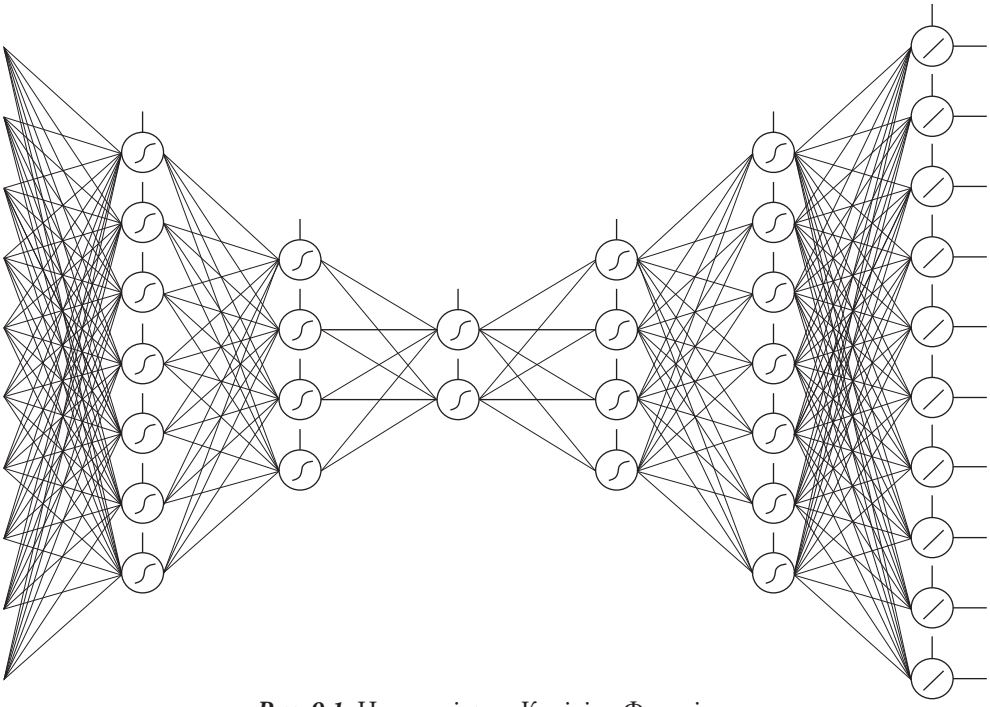


Рис. 9.1. Неокогнітрон Куніхіко Фукусіми

Першим автоасоціатором став неокогнітрон (рис. 9.1), запропонований японським ученим Куніхіко Фукусімою (Kunihiko Fukushima) [9.12].

Є два типи АА — генерувальні та синтезувальні. Для першого типу використовується обмежена машина Больцмана (ОМБ), для другого — автокодувальники (АК).

Автокодувальник. Одним із перших алгоритмів глибокого навчання є автокодувальник. Це алгоритм навчання без учителя, вихідний вектор якого дорівнює вхідному вектору [9.10]. Однією з найпоширеніших архітектур автокодувальників є нейронна мережа прямого поширення, що містить вхідний, прихований та вихідний шари.

На відміну від виходу перцептрона шар автокодувальника має таку саму кількість нейронів, як і вхідний шар. Дані вхідного шару стискаються і відновлюються, щоб отримати приховані ознаки.

Мета автокодувальника полягає в тому, щоб забезпечити максимальну близькість виходу НМ до вхідного вектора. Для нетривіального розв'язання цієї задачі на топологію мережі накладаються спеціальні обмеження:

1) кількість нейронів у прихованому шарі має бути меншою за кількість вхідних нейронів;

2) кількість неактивних нейронів у прихованому шарі має значно перевищувати кількість активних нейронів.

Перше обмеження дає можливість стискати дані під час передання вхідного сигналу на вихід мережі. Таке стискання можливе, якщо в даних є приховані взаємозв'язки та кореляція між ознаками. Друге обмеження — потреба великої кількості неактивних нейронів — дає змогу отримати нетривіальні результати навіть у разі, якщо кількість нейронів у прихованому шарі перевищує розмірність вхідних даних. Інакше кажучи, метою автокодувальника є отримання найбільш значущих ознак.

Вважатимемо, що нейрон є активним, якщо його активація близька до одиниці, і неактивним, якщо його активація близька до нуля. Ці обмеження змушують автокодувальник шукати кореляції та узагальнення у вхідних даних і виконувати їх стискання.

Отже, мережа автоматично навчається виділяти у вхідних даних загальні ознаки, які закодовано у вагах мережі. Необхідно, щоб середнє значення передавальної функції кожного прихованого нейрона набувало максимального значення, близького до заданого параметра розрідженості, приблизно $s = 0{,}05$. Для цього в кожний нейрон прихованого шару було введено параметр розрідженості ρ:

$$\hat{\rho}_j = \frac{1}{m} \sum_{i=1}^{m} [a_j^{(2)}(x^{(i)})]. \tag{9.1}$$

Потрібно, щоб середнє значення передавальної функції кожного прихованого нейрона набувало найближчого до ρ значення:

$$\hat{\rho}_j = \rho. \tag{9.2}$$

Введемо штрафну функцію:

$$S = \sum_{j=1}^{S_2} KL(\rho | \hat{\rho}_j), \tag{9.3}$$

де

$$KL(\hat{\rho}_j) = \rho \log \frac{\rho}{\hat{\rho}_j} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}_j}. \tag{9.4}$$

Важливою властивістю штрафної функції є її похідна:

$$\frac{\partial KL(\hat{\rho}_j)}{\partial \rho_j} = \frac{\rho}{\hat{\rho}_j} + \frac{1 - \rho}{1 - \hat{\rho}_j}, \tag{9.5}$$

Приклад автокодувальника наведено на рис. 9.2. Автокодувальник прагне побудувати функцію $h(x) = x$, тобто знайти таке наближення цієї функції, що вихід НМ дорівнюватиме вхідному вектору. Це робить розв'язання цієї задачі нетривіальним: кількість нейронів прихованого шару має бути меншою за розмірність вхідних даних (див. рис. 9.2), що, зі свого боку, дає можливість досягти стискання даних за допомогою передання вхідного сигналу на вихід.

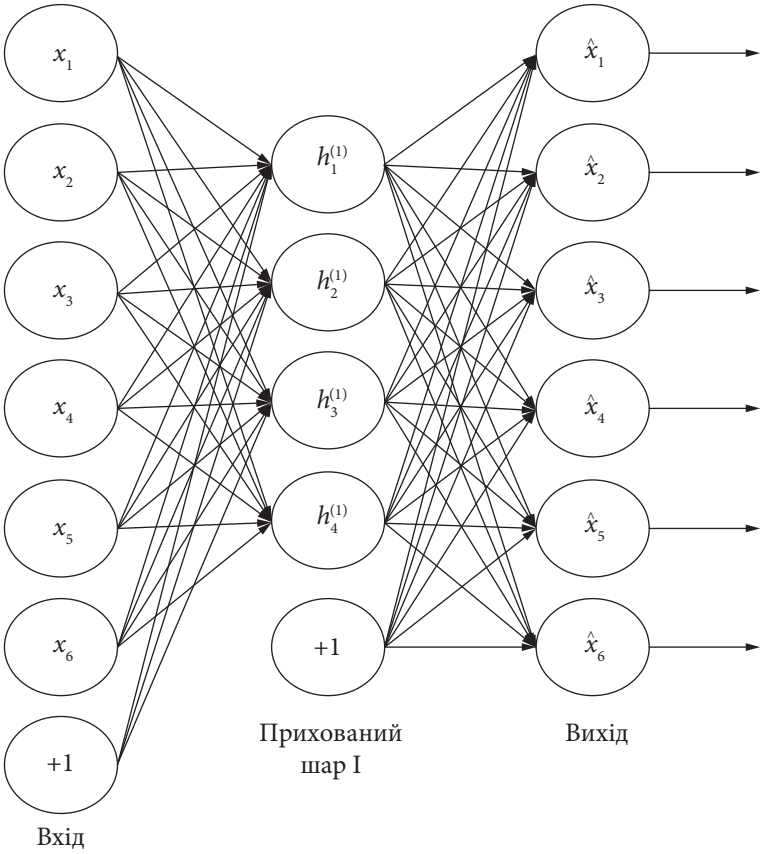


Рис. 9.2. Архітектура автокодувальника

Наприклад, якщо вхідний вектор є набором рівнів яскравості зображення розміром 10 × 10 пікселів (усього 100 ознак), то кількість нейронів прихованого шару дорівнює 50 і мережа змушена вчитися стискати зображення.

Справді, вимога $h(x) = x$ означає, що на основі рівнів активації 50 нейронів вихідний шар має відновити 100 пікселів початкового зображення. Таке стискання можливе, якщо є приховані взаємозв'язки, кореляція в ознаках і загалом певна структура в даних. Отже, функції автокодувальника дуже нагадують алгоритм головних компонент (*Principal Component Analysis, PCA*) у контексті скорочення розмірності вхідних даних.

Пізніше, під час обговорення ідеї розрідженості, з'явився т. зв. розріджений автокодувальник, що набув широкого застосування [9.10, 9.26]. Розріджений автокодувальник — це автокодувальник, кількість прихованих нейронів якого значно перевищує розмірність вхідного вектора. Розріджена активація означає, що кількість неактивних нейронів у прихованому шарі значно перевищує кількість активних. Неформально розрідженість можна описати так:

нейрон вважається активним, якщо його передавальна функція дорівнює приблизно 1. У разі використання сигмоїда як передавальної функції її значення для неактивного нейрона має бути близьким до нуля (для гіперболічного тангенса — близьким до –1).

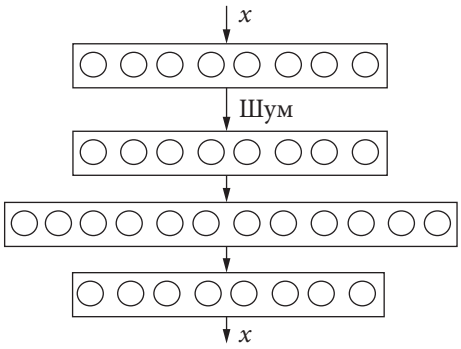


Рис. 9.3. Структура знешумлювального автокодувальника

Варіант автокодувальника, який називається «знешумлювальним» автокодувальником (*denoising autoencoder*) [9.7, 9.10, 9.26] відрізняється від автокодувальника у класичному розумінні специфікою навчання: на вхід подаються випадково спотворені дані (кілька вхідних значень змінюються на 0), неспотворені значення виводяться для порівняння. Отже, мета автокодувальників — відновлення спотворених вхідних даних (рис. 9.3).

Штучні нейронні мережі прямого поширення (ШНМ) з великою кількістю шарів важко навчаються звичайними методами, які придатні для ШНМ з невеликою кількістю прихованих шарів через проблему градієнта, що зникає. Що далі від виходу перебуває шар, то менше значення норми градієнта.

Цю проблему можна вирішити за допомогою правильно підібраних початкових ваг. У цьому випадку не потрібно суттєво змінювати їх у ході тренувального процесу.

9.2. Машини Больцмана (БМ)

Машини Больцмана (Людвіг Больцман, Ludwig Eduard Boltzmann) є особливою формою логічно лінійного поля Маркова (ЛПМ) (Андрій Марков, Andrey Markov), тобто його енергетична функція є лінійною за параметрами. Тому спочатку розглянемо енергетичні моделі (ЕМ). Енергетичні моделі пов'язують енергію з кожною конфігурацією змінних. Навчання відповідає модифікації енергетичної функції так, щоб та набувала бажаних властивостей. Наприклад, потрібно, щоб бажані конфігурації мали низьку енергію. А отже, імовірнісні моделі, які ґрунтуються на енергетичному принципі, визначають розподіл імовірностей [9.14, 9.15]:

$$p(x) = \frac{e^{-E(x)}}{z}. \tag{9.6}$$

Нормувальний множник Z називається статистичною сумою за аналогією з фізичними системами:

$$Z = \sum_x e^{-E(x)}. \tag{9.7}$$

Енергетичні моделі можна досліджувати за допомогою стохастичного градієнтного спуску за емпіричною від'ємно-логарифмічною функцією ймовірності даних.

Що стосується логістичної регресії, спочатку визначають логарифмічну функцію правдоподібності, а потім функцію втрат як від'ємну логарифмічну функцію правдоподібності:

$$L(\theta, D) = \frac{1}{N} \Sigma_{x^{(i)}} \log p(x^{(i)}); \tag{9.8}$$

$$l(\theta, D) = -L(\theta, D).$$

Обмежена машина Больцмана (ОМБ)

Історія розвитку ОМБ почалася з рекурентної нейронної мережі (РНМ). Однією з таких моделей була мережа Хопфілда, який також увів поняття енергії, порівнявши нейродинаміку з термодинамікою.

Наступним кроком були звичайні машини Больцмана, які відрізняються від мережі Хопфілда своєю стохастичною природою та поділом нейронів на дві групи, що описують приховані і видимі стани.

Обмежені машини Больцмана відрізняються від звичайних тим, що в них немає зв'язків між нейронами одного шару (подібно до прихованих марковських моделей). Структуру ОМБ зображено на рис. 9.4.

Властивість цієї моделі полягає в тому, що за заданого стану однієї групи нейронів стани іншої групи нейронів будуть незалежними один від одного. Тепер розглянемо деякі теоретичні результати, в яких ця властивість відіграє ключову роль.

Інтерпретація ОМБ. ОМБ інтерпретуються як приховані марковські моделі. Вони мають шар станів, які ми можемо спостерігати (видимі нейрони), і шар станів, які приховані, і ми не можемо їх бачити (приховані нейрони). Утім, можна зробити ймовірнісний висновок про приховані стани на основі видимих. Після навчання такої моделі отримуємо також можливість робити висновки про видимі стани, маючи інформацію про приховані (використовуючи теорему Байєсса), і в такий спосіб генерувати дані з того імовірнісного розподілу, на якому навчалася модель.

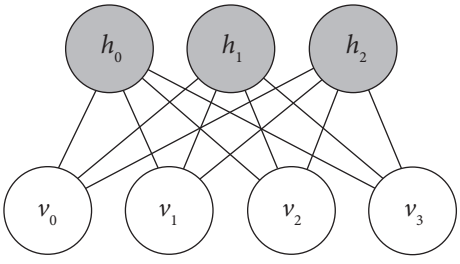


Рис. 9.4. Структура ОМБ

Отже, можна сформулювати мету навчання ОМБ: потрібно налаштувати параметри моделі так, щоб відновлений вектор був максимально наближений до оригіналу. Під відновленим вектором мається на увазі вектор, отриманий імовірнісним виведенням з видимих станів.

Алгоритм ОМБ

Часто немає можливості повного спостереження зразка X, або ж метою є введення деяких неспостережуваних змінних, які збільшують описову силу моделі. Тому розглянемо видиму частину моделі [9.15, 9.16, 9.23, 9.24] (яку позначимо X) і невидиму частину (позначимо h). Запишемо:

$$P(x) = \sum_h P(x, h) = \sum_h \frac{e^{-E(x)}}{Z}. \tag{9.9}$$

Енергетична функція $E(v, h)$ обмеженої машини Больцмана має вигляд

$$E(v, h) = b^T v + c^T h + h' W v, \tag{9.10}$$

де W — вага, що з'єднує видимі й невидимі нейрони; b і c — зсуви видимого та прихованого шарів відповідно.

Це безпосередньо переноситься в таку формулу для вільної енергії:

$$F(v) = -b' v - \sum_i \log \sum_{h_i} e^{h_i (c_i + W_i v)}. \tag{9.11}$$

Завдяки специфічній структурі ОМБ видимі та невидимі нейрони є умовно залежними одні від одних. Використовуючи цю властивість, можна записати:

$$p(h \mid v) = \prod_i p(h_i \mid v);$$
$$p(v \mid h) = \prod_j p(v_j \mid h). \tag{9.12}$$

Мережа складається зі стохастичних нейронів, які набувають значень 0 або 1 (де v_j та $h_i \in \{0, 1\}$). З формул (9.10) і (9.11) отримуємо імовірнісний варіант звичайної активації нейрона:

$$p(v, W, b_h) = \sigma(W * v + b_h), \tag{9.13}$$

де v — вхід нейрона; W — ваговий вектор (матриця); b_h — зсув; $\sigma(x)$ — сигмоїдальна функція. Це базовий варіант для бінарних входів (МНВ Бернуллі) (Йоганн Бернуллі, Johann Bernoulli), але є також модифікації для дійсних входів (МНВ Гаусса — Бернуллі та ін).

Алгоритм ОМБ виглядає так.

1. Встановлюємо початкові значення для вхідних змінних v: = x.

2. Обчислюємо ймовірності p_h зміни станів нейронів другого шару (невидимого) $p_h = \sigma(v * W + b_v)$, де W — вагова матриця; b_v — вектор зсуву першого шару; σ — функція активації (сигмоїдна).

Зберігаємо попередні значення вхідних нейронів $v' := v$.

3. Визначаємо стани нейронів другого шару h і присвоюємо нейронам стан 0 або 1 з імовірністю p_h.

4. Обчислюємо ймовірності p_v зміни станів нейронів першого шару:

$$p_v := \sigma(h * W^T + b_h),$$

де b_h — вектор зсуву другого шару; σ — функція активації (сигмоїдна), і присвоюємо нейронам стан 1 з імовірністю p_v (або 0 з імовірністю $1 - p_v$).

5. Якщо $v \neq v'$, повторюємо алгоритм з кроку 2. В іншому випадку переходимо до наступного кроку.

6. Виводимо результат v.

7. Кінець.

Метод навчання «контрастна дивергенція» (КД)

Алгоритм навчання ОМБ називається контрастною дивергенцією і є модифікованим градієнтним спуском [9.21, 9.22]. Як функцію оцінки, що оптимізується, використовують функцію правдоподібності L. Шукатимемо її максимум. Функція правдоподібності L для параметрів (W, b_v, b_h) та шаблона v визначається за заданих значень параметрів W, h як $p(v, h) = \frac{1}{z}e^{-E(v, h)}$:

$$L(v) = p(\theta);$$

$$L(v) = p(v \mid W, b_v, b_h). \tag{9.14}$$

Для простоти обчислень використаємо логарифм (9.15):

$$\ln L(v) = \ln p(v) = \ln\frac{1}{z}\sum_h \exp(-E(v, h)) =$$

$$= \ln\sum_h \exp(-E(v, h)) - \ln\sum_{v,h} \exp(-E(v, h)). \tag{9.15}$$

За великої кількості видимих і прихованих нейронів для знаходження максимуму логарифмічної функції правдоподібності зазвичай застосовують чисельні методи, такі як градієнтний спуск. Максимізація функції правдоподібності еквівалентна мінімізації ваг, оскільки ваги лінійно пов'язані з енергетичною функцією E:

$$E(v, h) = (b_v * v + b_h * h + v * h * W). \tag{9.16}$$

Як випливає з виразу (9.16), мінімізація ваг призводить до мінімізації енергетичної функції E. На кожній ітерації алгоритму градієнтного спуску параметри налаштовуються залежно від функції правдоподібності, похідні якої такі:

$$\frac{\partial \ln L(v)}{\partial \theta} = \frac{\partial}{\partial \theta}\left(\ln\sum_h e^{-E(v,h)}\right) - \frac{\partial}{\partial \theta}\left(\ln\sum_{v,h} e^{-E(v,h)}\right) = -\frac{1}{\sum_h e^{-E(v,h)}}\sum_h e^{-E(v,h)}\frac{\partial E(v, h)}{\partial \theta} +$$

$$+ \frac{1}{\sum_{h,v} e^{-E(v,h)}}\sum_{h,v} e^{-E(v,h)}\frac{\partial E(v, h)}{\partial \theta} = -\sum_h p(v)\frac{\partial E(v, h)}{\partial \theta} + \sum_{h,v} p(h, v)\frac{\partial E(v, h)}{\partial \theta}. \tag{9.17}$$

Градієнт цієї функції можна розподілити на три частини [9.22]:

$$\begin{cases} \dfrac{\partial \ln L(W, b_v, b_h \mid v)}{\partial W} = \nabla \mathrm{W} = (v * h)_{\text{data}} - (v * h)_{\text{model}}; \\ \dfrac{\partial \ln L(W, b_v, b_h \mid v)}{\partial b_v} = \nabla b_v = (v)_{\text{data}} - (v)_{\text{model}}; \\ \dfrac{\partial \ln L(W, b_v, b_h \mid v)}{\partial b_h} = \nabla b_h = (h)_{\text{data}} - (h)_{\text{model}}, \end{cases} \tag{9.18}$$

де $(\cdot)_{\text{data}}$ — значення станів шарів у початковий момент часу; $(\cdot)_{\text{model}}$ — математичне сподівання станів шарів.

Математичне сподівання станів нейронів обчислюється за допомогою так званої дискретизації, тобто $(\cdot)_{\text{model}}$ — це стан шару після деяких ітерацій (на практиці для роботи алгоритму достатньо одного кроку дискретизації (однієї ітерації)). Ваги змінюються так:

$$\begin{cases} W \coloneqq \varepsilon * (\nabla \mathrm{W} + \mu * \Delta \mathrm{W}), b_v; \\ b_h \coloneqq \varepsilon * (\nabla b_h + \mu * \Delta b_h), \end{cases} \tag{9.19}$$

де μ — так званий моментний параметр; ε — швидкість навчання; ∇W, ∇b_v, ∇b_h — параметри, змінені на попередній ітерації.

Як критерій зупинки використаємо середньоквадратичну похибку між входом і виходом ОМБ $E(v_0, v_k)$, і це значення має зменшитися до встановленого порогу $E_{\min}$.

Алгоритм навчання складається з таких кроків [9.17—9.19].

Ініціалізуємо (нулями) вагову матрицю W та вектори зміщення b_v, b_h.

Обираємо випадковий мініпакет з усієї навчальної вибірки (мініпакета) X.

1. Для всіх прикладів у мініпакеті присвоюємо першому шару початкові значення $v \coloneqq x$.

2. Виконуємо k циклів у мережі і визначаємо початковий та кінцевий стани шарів v_0, h_0, v_k, h_k, де k — параметр).

3. Обчислюємо градієнт за формулою (9.18) і коригуємо ваги за формулою (9.19).

4. Обчислюємо MSE E.

5. Якщо $E < E_{\min}$, переходимо до наступного кроку, інакше — повертаємось до кроку 2.

6. Кінець.

9.3. Алгоритм навчання «контрастна дивергенція» (КД-k)

Цей алгоритм, розроблений професором Дж. Хінтоном (Джеффрі Хінтон, Geoffrey Hinton) у 2002 р., вирізняється своєю простотою [9.18]. Основна ідея полягає в тому, що математичне сподівання замінюється певними значеннями. Вводиться поняття вибірки — вибірка Гіббса (Джозая Віллард Гіббс, Josiah Willard Gibbs).

Алгоритм КД-k працює так.

1. Стани видимих нейронів встановлюють такими, що дорівнюють вхідному шаблону.

2. Обчислюють ймовірності нейронів прихованого шару.

3. Кожному нейрону прихованого шару присвоюють стан «1» з імовірністю, що дорівнює його поточному стану.

4. Ймовірності станів видимого шару визначають на основі станів прихованого шару.

5. Якщо кількість поточних ітерацій менша за k, повертаються до кроку 2.

6. Отримано ймовірності станів нейронів прихованого шару.

Роботу відповідного алгоритму наведено на рис. 9.5.

Чим довший процес дискретизації, тим точнішим є алгоритм КД.

Приклад. Розглянемо реалізацію описаної моделі. На початку в пам'яті зберігається декілька зображень латинських літер (рис. 9.6, *а*). Потім системі надаються інші подібні спотворені зображення, за якими потрібно відновити початкове зображення. Навчання розпочато.

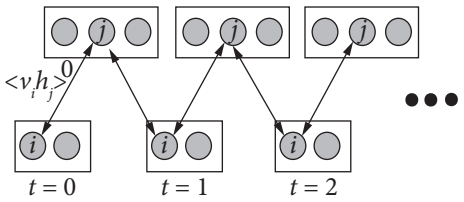


Рис. 9.5. Алгоритм навчання КД-k

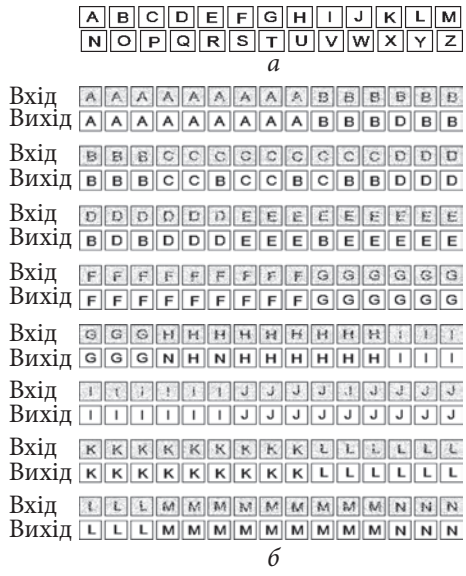


Рис. 9.6. Робота алгоритму КД-k: *а* — вихідні дані; *б* — результати роботи ►

9.4. Стекові мережі автоасоціаторів

Стековий автокодувальник (САК)

Для отримання високорівневих абстракцій з вхідної множини автоасоціатори об'єднують у мережу. На рис. 9.7 показано структурну схему стекового автокодувальника, яка в цілому становить собою мережу глибокого навчання з вагами, ініціалізованими стековим автокодувальником [9.17, 9.19].

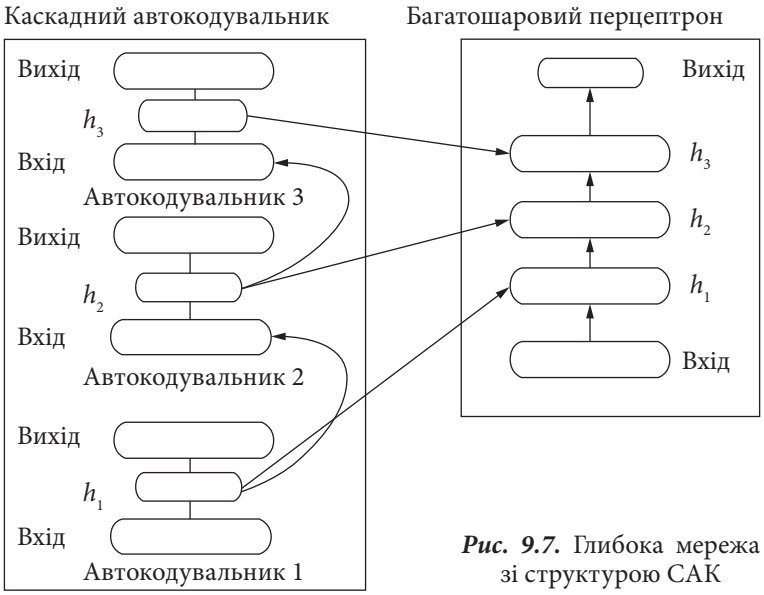


Рис. 9.7. Глибока мережа зі структурою САК

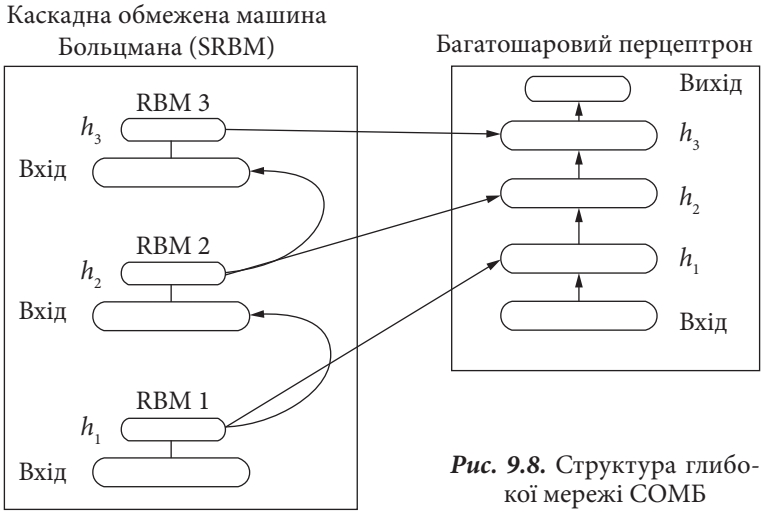


Рис. 9.8. Структура глибокої мережі СОМБ

Стекова ОМБ

Структурну схему стекових обмежених мереж Больцмана (СОМБ) і нейронних мереж, що є глибокою нейронною мережею з вагами, ініціалізованими СОМБ, зображено на рис. 9.8.

Структури глибоких мереж показано саме так, підкреслюючи, що інформація передається догори (знизу вгору).

9.5. Глибоке мережеве навчання

Процес навчання глибоких мереж поділяється на два етапи [9.1—9.3]:

1) попередня підготовка;

2) тонке налаштування ваг.

Попередня підготовка до роботи з глибокою мережею

На першому етапі автоасоціативна мережа (САК або СОМБ) навчається на масиві немаркованих даних, після чого нейрони прихованого шару багатошарового перцептрона (БП) ініціалізуються вагами, отриманими після навчання. Цей процес навчання та перенесення показано на рис. 9.8. Після навчання першого АК/ОМБ ваги нейронів прихованого шару стають входами другого шару і так далі. Отже, з даних дістається все більше загальної інформації про структуру даних (лінія, контур тощо).

Розглянемо процедуру підготовки детальніше. Попереднє навчання полягає в таких діях: потрібно взяти пари сусідніх шарів мережі глибокого навчання, починаючи з першого шару, і побудувати з цих пар автокодувальник, додаючи вихідний шар, ідентичний до вхідного. Цю процедуру повторюють послідовно для всіх шарів мережі. Її можна описати так.

1. Завантажуємо навчальний набір даних X_0.

2. Визначаємо параметри мережі — кількість (N) та розмір шарів.

3. Встановлюємо номер поточного шару $i = 1$.

4. Будуємо автокодувальник для шарів i, $i + 1$.

5. Навчаємо автокодувальник на заданому значенні X_i.

6. Прибираємо допоміжний (вихідний) шар автокодувальника.

7. Зберігаємо вагу з'єднання шарів i, $i + 1$.

8. Якщо залишилося обробити ще пару шарів ($i < N - 2$), переходимо до наступного кроку, інакше — до кроку 10.

9. Генеруємо набір даних X_{i+1} для наступного автокодувальника; для цього пропускаємо через пару шарів i, $i + 1$, набір даних X_i, переходимо до кроку 3.

10. Завершуємо роботу.

Після цієї процедури мережа навчається одним із градієнтних методів.

Крім того, Дж. Хінтон запропонував також виконувати тонке налаштування в два етапи для глибоких мереж з більш ніж трьома прихованими шарами. На першому етапі тренувати лише два верхніх шари, а на другому — всю мережу.

Варто зазначити, що у разі неконтрольованого навчання СОМБ дає менш стабільні результати, ніж САК.

Тонке налаштування (*Fine Tuning*)

На другому етапі виконується тонке налаштування ваг мережі *DL* (навчання з учителем) відомими методами. Практично доведено, що така ініціалізація встановлює ваги нейронів прихованих шарів мережі *DL* в області глобального мінімуму і наступне тонке налаштування виконується за дуже короткий час.

Тонке налаштування — це процес зважування невеликих змін для поліпшення або оптимізації результатів. Як правило, воно спрямоване на підвищення ефективності процесу. Тонке налаштування може бути виконано за допомогою методів, що залежать від оптимізованих процесів і до яких входять градієнтні методи першого порядку і градієнтні методи другого порядку — ньютонівські, квазіньютонівські та інші методи.

9.6. Алгоритми оптимізації функції вартості в машинному навчанні для НМ

Наразі навчальні пакети згорткових нейронних мереж (ЗНМ) мають великий набір оптимізаторів функції вартості. Робота більшості оптимізаторів ґрунтується на обчисленні градієнта, але є й спеціальні оптимізатори, призначені для розв'язання задачі локальних мінімумів [9.1, 9.2]. У цьому розділі розглянуто найвідоміші алгоритми оптимізації функції вартості.

Градієнтний спуск

Градієнтний спуск є основним методом пошуку локального мінімуму функції [9.30, 9.23]. Правило оновлення ваг θ_i виражається формулою

$$\theta_i^{(t+1)} = \theta_i^{(t)} - \lambda \frac{\partial C^{(t)}}{\partial \theta_i}, \tag{9.20}$$

де λ — швидкість навчання; $\theta_i^{(t)}$, $\theta_i^{(t+1)}$ — значення параметра i на ітераціях t та $t + 1$ відповідно; $C^{(t)}$ — функція вартості.

Адаптивний градієнтний спуск (*Adagrad*)

Адаптивний градієнтний спуск (*Adagrad*) нормалізує швидкість навчання для кожного виміру, від якого залежить функція вартості. На кожній ітерації глобальна швидкість навчання ділиться на l^2 — норму попередніх градієнтів для кожного виміру [9.25, 9.33]. Правило оновлення параметрів виражається формулою

$$\theta_i^{(t+1)} = \theta_i^{(t)} - \frac{\lambda}{\sqrt{\sum_{\tau=1}^{t} \theta_i^{(\tau)2} + \varepsilon}} \frac{\partial C^{(t)}}{\partial \theta_i}, \tag{9.21}$$

де λ — швидкість навчання; $\theta^{(t)}$, $\theta^{(t+1)}$ — вектори параметрів на ітераціях t та $t+1$ відповідно; $C^{(t)}$ — функція вартості; $\sqrt{\sum_{\tau=1}^{t} \theta_i^{(\tau)2} + \varepsilon}$ — l^2 — норма параметрів усіх попередніх ітерацій для ознаки i.

В алгоритмі *Adagrad* параметри оновлюються з різною швидкістю навчання. Що частіше трапляється ознака, то менша швидкість навчання, і що рідше трапляється ознака, то більша швидкість навчання. Це пов'язано з тим, що для розсіяних ознак значення норми L_2 буде меншим, тому загальна швидкість навчання буде вищою. Цей алгоритм добре працює у сферах обробки природної мови та обробки зображень, для яких характерні розсіяні дані.

Алгоритм *RMSprop*

Алгоритм *RMSprop* реалізує версію методу оптимізації стійкого зворотного поширення (*Rprop*) з міні пакетами (мінібатчами), який найбільш придатний для повнопакетного навчання. Алгоритм *Rprop* розв'язує проблему, коли вектори градієнта не орієнтовані на мінімум в еліптичних контурах функції цінності. Характерною особливістю алгоритму *Rprop* є те, що для кожної ваги в разі оновлення використовуються тільки знаки градієнта.

На першому кроці алгоритм *Rprop* починає з однакової швидкості оновлення для всіх ваг:

$$\Delta_{ij}^{(t=0)} = \Delta_{ij}^{(0)} = \Delta, \tag{9.22}$$

де $\Delta_{ij}^{(0)}$ — коефіцієнт оновлення ваги w_{ij} на першій ітерації. Також задаються мінімальні та максимальні значення вагових коефіцієнтів $\Delta_{\min}$ та $\Delta_{\max}$ відповідно. На кожній ітерації алгоритм перевіряє знаки попередньої та поточної компонент градієнта, тобто частинних похідних функції витрат за різних ваг.

Якщо знаки поточної та попередньої ітерацій збігаються, тобто $\operatorname{sign}\left(\frac{\partial C^{(t)}}{\partial w_{ij}} \times \frac{\partial C^{(t-1)}}{\partial w_{ij}}\right) > 0$, то швидкість навчання збільшується з коефіцієнтом $\lambda_+ = 1{,}2$.

Коефіцієнт оновлення розраховується за формулою

$$\Delta_{ij}^{(t+1)} = \min(\lambda_+ \Delta_{ij}^{(t)}, \Delta_{\max}), \tag{9.23}$$

де λ_+ — коефіцієнт підвищеної швидкості навчання; $\Delta_{ij}^{(t)}$ — коефіцієнт оновлення w_{ij} на ітерації t; $\Delta_{\max}$ — максимальне значення коефіцієнта оновлення.

Якщо знаки поточної та попередньої складових градієнта відрізняються, тобто $\text{sign}\left(\frac{\partial C^{(t)}}{\partial w_{ij}} \frac{\partial C^{(t-1)}}{\partial w_{ij}}\right) > 0$, то швидкість навчання зменшується з коефіцієнтом $\lambda_- = 0{,}5$.

Після цього правило оновлення набуває вигляду

$$\Delta_{ij}^{(t+1)} = \max(\lambda_- \Delta_{ij}^{(t)}, \Delta_{\min}), \tag{9.24}$$

де λ_- — коефіцієнт зменшення швидкості навчання; $\Delta_{ij}^{(t)}$ — коефіцієнт оновлення w_{ij} на ітерації t; $\Delta_{\min}$ — мінімальне значення коефіцієнта оновлення.

Якщо $\text{sign}\left(\frac{\partial C^{(t)}}{\partial w_{ij}} \frac{\partial C^{(t-1)}}{\partial w_{ij}}\right)$ дорівнює нулю, то коефіцієнт оновлення не змінюється:

$$\Delta_{ij}^{(t+1)} = \Delta_{ij}^{(t)}.$$

Правило оновлення параметрів виражається формулою

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} - \text{sign}\left(\frac{\partial C^{(t)}}{\partial w_{ij}}\right)\Delta_{ij}^{(t+1)}. \tag{9.25}$$

Оптимізатор *RMSprop* поєднує можливості адаптивного навчання кожної ваги алгоритму *Rprop* з ефективністю стохастичного градієнтного спуску. Алгоритм *Rprop* не використовує знак градієнта для кожної ваги.

Проблемою стохастичного градієнтного спуску є зміна функції вартості та градієнтів з кожним новим міні-пакетом. Основна ідея *RMSprop* полягає в тому, щоб вибрати таке значення градієнта, яке не буде суттєво коливатися в межах декількох сусідніх міні-пакетів. Для цього необхідно використовувати середньоквадратичні значення градієнтів для кожної ваги на останніх кількох міні-пакетах [9.24, 9.25]

$$g_{ij}^{(t)} = \alpha g_{ij}^{(t+1)} + (1 - \alpha)\left(\frac{\partial C^{(t)}}{\partial w_{ij}}\right)^2; \tag{9.26}$$

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} - \frac{\lambda}{\sqrt{g_{ij}^{(t)} + \varepsilon}} \frac{\partial C^{(t)}}{\partial w_{ij}}, \tag{9.27}$$

де $g_{ij}^{(t)}$ — середнє значення квадрата градієнта для ваги w_{ij} на ітерації t; α — коефіцієнт зменшення квадрата градієнта; $\frac{\partial C^{(t)}}{\partial w_{ij}}$ показує значення компонент градієнта.

Алгоритм *Adam*

Алгоритм *Adam* (адаптивний оцінювач моменту) підтримує не тільки ковзне середнє квадратів градієнтів, а й ковзне середнє минулих градієнтів.

Алгоритм *Adam* є ще одним алгоритмом, який має адаптивну швидкість навчання для кожного з параметрів [9.24, 9.25].

Нехай β_1 та β_2 — середні темпи зменшення середнього $m_{ij}^{(t)}$ і середньоквадратичного значення градієнтів $v_{ij}^{(t)}$ для всіх ваг w_{ij}. Тоді імпульс можна розглядати як кульку, що котиться по схилу, а алгоритм *Adam* моделює поведінку важкої кульки з тертям, віддаючи перевагу пласким мінімумам на поверхні функції похибки. Тоді правила оновлення для алгоритму *Adam* записують як

$$m_{ij}^{(t)} = \beta_1 m_{ij}^{(t-1)} + (1 - \beta_1)\left(\frac{\partial C^{(t)}}{\partial w_{ij}}\right); \tag{9.28}$$

$$v_{ij}^{(t)} = \beta_2 v_{ij}^{(t-1)} + (1 - \beta_2)\left(\frac{\partial C^{(t)}}{\partial w_{ij}}\right)^2, \tag{9.29}$$

де $\dfrac{\partial C^{(t)}}{\partial w_{ij}}$ — значення компонент градієнта.

Нормовані значення середнього значення градієнтів $\hat{m}_{ij}^{(t)}$ та середнє квадратів градієнтів $\hat{v}_{ij}^{(t)}$ обчислюються за формулами

$$\begin{aligned} \hat{m}_{ij}^{(t)} &= \frac{m_{ij}^{(t)}}{(1 - \beta_1^t)}; \\ \hat{v}_{ij}^{(t)} &= \frac{v_{ij}^{(t)}}{(1 - \beta_2^t)}. \end{aligned} \tag{9.30}$$

Правило оновлення для кожної ваги має такий вигляд:

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} - \frac{\lambda}{\sqrt{\hat{v}_{ij}^{(t)} + \varepsilon}}\, \hat{m}_{ij}^{(t)}. \tag{9.31}$$

9.7. Регуляризація глибокого навчання

Під час навчання нейронних мереж трапляється два типи похибок: похибка навчання ε_{tr} і похибка узагальнення ε_{gen}. Похибку навчання знаходять за навчальною вибіркою, а похибку узагальнення — за перевірною. Ці дві похибки є функціями кількості ітерацій навчання n і мають різні властивості: ε_{tr} монотонно зменшуються зі збільшенням n, тоді як похибка узагальнення ε_{gen} спочатку зменшується, потім досягає мінімуму, а потім починає зростати зі збільшенням n (це явище називається *перенавчанням*). Мета навчання полягає в мінімізації похибки узагальнення.

Регуляризація — це будь-яка модифікація алгоритму навчання, спрямована на зменшення похибки узагальнення за рахунок певного збільшення похибки навчання. Регуляризація є однією з центральних проблем машинного навчання, яка за своєю значущістю конкурує з проблемою оптимізації.

Відповідно до вислову «про безкоштовний сніданок», немає найкращого алгоритму для машинного навчання. Зокрема, немає єдиного найкращого методу регуляризації. Натомість потрібно вибрати форму регуляризації, найбільш придатну для задачі, яку маємо розв'язати. У філософії глибокого навчання широке коло проблем (наприклад, всі інтелектуальні проблеми) можуть бути ефективно вирішені із застосуванням загальних форм (методів) регуляризації. Розглянемо найпопулярніші методи та форми регуляризації [9.25].

L_p-Регуляризація лінійної регресії

Розглянемо класичну лінійну регресійну модель:

$$t = \sum_{j=1}^{d} w_j x(j) + \varepsilon,\ \varepsilon \sim N(0, \sigma^2).$$

Пошук ваг w шляхом максимізації функції правдоподібності вибірки в цій моделі еквівалентний методу МНК [9.1, 9.2]:

$$\sum_{n=1}^{N} [t_n - \sum_{j=1}^{d} w_j x_j]^2 = ||t - w_1 x_1 - \cdots - w_d x_d||^2 = ||t - Xw||^2 \to \min, \quad (9.32)$$

де $x_i \in R^N$ значення i-ї ознаки для всіх об'єктів у вибірці $X = [x_1, \ldots, x]$.

Зауважимо, що введене тут позначення x_i відрізняється від стандартного, коли під x_i мається на увазі i-й об'єкт вибірки. Тут і далі вибірка вважається нормалізованою.

Ця задача має просту геометричну інтерпретацію — пошук проєкції вектора t на гіперплощину з векторами напрямків $x_1, \ldots, x_d$ (див. рис. 9.9).

Задачу (9.32) можна розв'язати аналітично: $w = (X^T X)^{-1} X^T t$, $t_{\text{pr}} = Xw = (X^T X)^{-1} X^T t$.

Розв'язок для w відповідає псевдорозв'язку системи лінійних рівнянь $Xw = t$.

Щоб запобігти переналаштуванню лінійної регресії, необхідно встановити обмеження на варіабельність рішення. Це можна зробити, ввівши обмеження на норму вагового вектора w:

$$||t - Xw||^2 \to \min; \quad (9.33)$$

$$||w||_{L_p}^p \le b. \quad (9.34)$$

Традиційно замість розв'язання задачі (9.33) розглядається задача оптимізації такого регуляризованого функціонала:

$$||t - Xw||^2 + \lambda ||w||_{L_p}^p \to \min, \lambda \ge 0. \quad (9.35)$$

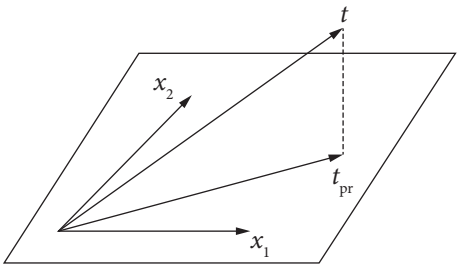


Рис. 9.9. Графічна ілюстрація лінійної регресії

Легко показати, що оптимізаційні задачі (9.33), (9.34) та (9.35) еквівалентні за умови $p \geq 1$, тобто коли всі розглянуті функції опуклі.

Введемо лагранжіан

$$L(w; \lambda) = ||t - Xw||^2 + \lambda(||w||_{L_p}^p - b). \quad (9.36)$$

Тоді за варіантом теореми Куна — Такера для опуклих функцій необхідною і достатньою умовою існування розв'язку $\hat{w}$ в задачі (9.33) є наявність $\lambda \geq 0$, для якої будуть виконуватись:

1) *принцип мінімуму* $L(\hat{w}; \lambda) = \min L(w; \lambda)$, тобто $\nabla L(w, \lambda) = 0$; (9.37)

2) *умова доповнювальної нежорсткості* $\lambda(||w||_{L_p}^p - b) = 0$. (9.38)

Зауважимо, що для достатності умов 1 і 2 потрібно також виконання умови Слейтера, тобто наявності такого розв'язку w, для якого $||w||_{L_p}^p < b$. Зрозуміло, що ця умова виконується за $b > 0$.

Задача оптимізації (9.35) еквівалентна умові 1. Розглянемо умову 2. Ця умова еквівалентна настанню однієї з двох подій: умова $\lambda = 0$ або $||w||_{L_p}^p = b$. Якщо $\lambda = 0$, то оптимальна точка лежить у області $||w||_{L_p}^p < b$. Отже, обмеження $||w||_{L_p}^p \leq b$ стає очевидним і задача оптимізації (9.33) перетворюється на задачу оптимізації без обмежень, яка еквівалентна задачі (9.35) за умови $\lambda = 0$. $\nabla L = 0$.

Нехай $\lambda > 0$; $||w||_{L_p}^p = b$. У задачі (9.35) це обмеження виконати легко. Просто позначимо через b значення норми вектора w, що оптимально для задачі (9.35).

Розглянемо оптимальний розв'язок задачі (9.35) за різних значень p. Можна показати, що у разі $p \leq 1$ оптимальний розв'язок має властивість розрідженості, тобто частина ваг точно дорівнює нулю. У випадку $p > 1$ строго нульові ваги в оптимальному розв'язку практично неможливі. Зауважимо, що ситуація $p = 1$ є особливою, оскільки в цьому випадку функціонал (9.35), що оптимізується, є опуклим, а оптимальний розв'язок — розрідженим. Метод підбору ваг у лінійній регресії за допомогою розв'язання задачі (9.33) або (9.35) з нормою L_1 дістав назву *LASSO* (абревіатура від англ. *Least absolutes shrinkage and selection operator* — оператор найменшого абсолютного скорочення та вибору).

Алгоритм «рання зупинка»

Алгоритм «рання зупинка» передбачає поділ процесу навчання на етапи навчання та валідації. Замість того, щоб навчання мережі відбувалося на обмеженій кількості ітерацій, мережа тренується доти, доки її ефективність не почне падати. Фактично це запобігає простому запам'ятовуванню шаблонів. Дві можливі точки зупинки показано на рис. 9.10. На рис. 9.11 наведено ефективність і ступінь перенавчання після зупинок у цих точках (*а* і *б*).

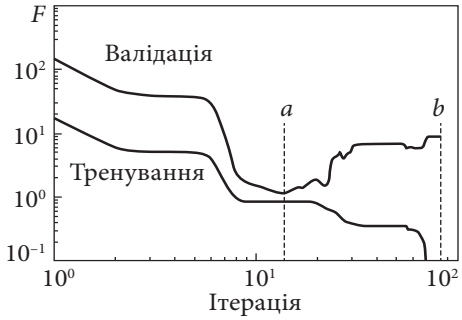


Рис. 9.10. Можливі точки зупинки

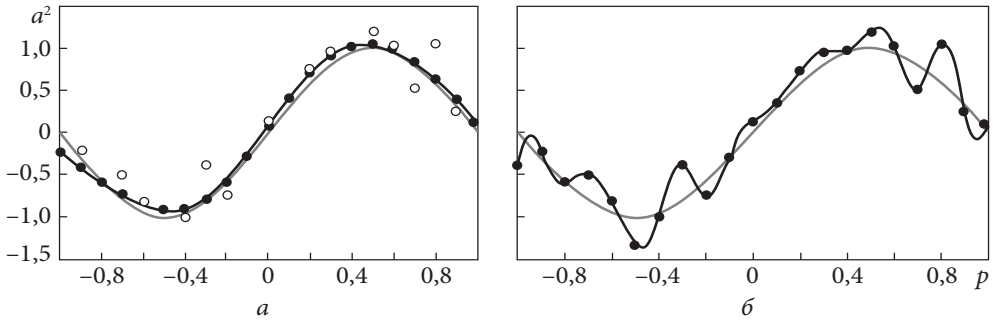


Рис. 9.11. Можливі варіанти поведінки мережі: *а* — рання зупинка (early stop); *б* — зупинка по мінімуму похибки на навчальній вибірці

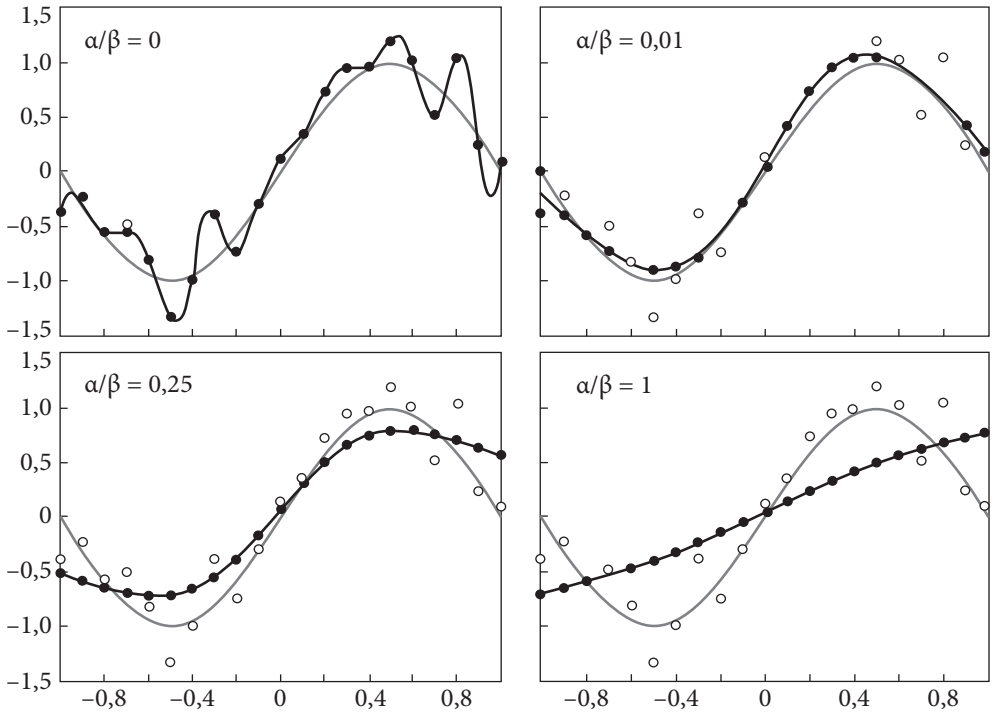


Рис. 9.12. Можливі варіанти для параметрів α та β

Регуляризація «штрафує» мережу за використання складної структури. Складність у цьому випадку вимірюється розміром та вагами мережі і встановлюється додаванням інтервалу до функції втрат, яка прив'язана до розміру та ваг [9.1]:

$$\epsilon = \beta\Sigma_{p=1}^{P_t}(t_p - O_p)^2 + \alpha\Sigma_{j=1}^{n}\nu_j^2,$$

де n — кількість ваг у нейронній мережі. Параметри α і β контролюють рівень, після якого відбувається недо- або перенавчання. Їх відповідні значення можна знайти за допомогою оптимізації або байєсового аналізу (рис. 9.12).

Видалення (*Dropout*)

Основна ідея видалення полягає в тому, щоб навчати не одну глибоку мережу, а ансамбль з декількох глибоких мереж, усереднюючи потім отримані результати [9.1, 9.2].

Мережі для навчання отримують видаленням з мережі (*Dropout*) нейронів з імовірністю p так, що ймовірність того, що нейрон залишиться в мережі, дорівнює $q = 1 - p$. «Видалення» нейрона означає, що за будь-яких вхідних даних він повертає значення «0».

Видалені нейрони не беруть участі в процесі навчання на всіх етапах роботи алгоритму, тому видалення навіть одного нейрона еквівалентне навчанню нової нейронної мережі.

Ймовірності видалення кожного нейрона рівні. Це означає, що:

- $h(x) = xW + b$ — лінійна проєкція вхідного вектора X розмірністю d_i;
- d_h — вимірний простір вихідних змінних;
- $a(h)$ — функція активації.

Застосування видалення до цієї проєкції на етапі навчання можна подати у вигляді модифікованої функції активації:

$$f(h) = D \odot a(h),$$

де $D = (X_1, \ldots, X_{dh})$ — d_h-вимірний вектор випадкових величин X_i, розподілений за законом Бернуллі. Тоді X_i має такий розподіл імовірностей:

$$f(k; p) = \begin{cases} p, & \text{if } k = 1; \\ 1 - p, & \text{if } k = 0, \end{cases}$$

де k — усі можливі вихідні значення.

Очевидно, що ця випадкова величина ідеально відповідає процедурі видалення, застосованій до одного нейрона. Дійсно, нейрон видаляється з імовірністю $p = P(k = 1)$, в іншому випадку він залишається невидаленим.

Розглянемо застосування видалення до i-го нейрона [9.1]:

$$O_i = X_i a(\sum_{k=1}^{d_i} w_k x_k + b) = \begin{cases} a(\sum_{k=1}^{d_i} w_k x_k + b), \text{ якщо } X_i = 1; \\ 0, \text{ якщо } X_i = 0, \end{cases} \quad (9.40)$$

де $P(X_i = 0) = p$.

Оскільки на етапі навчання нейрон залишається невидаленим з імовірністю q, на етапі тестування нам потрібно імітувати поведінку ансамблю нейронів, який використовувався на етапі навчання. Для цього було запропоновано помножити функцію активації на коефіцієнт q на етапі тестування. Отже, маємо

$$\text{на етапі навчання} — O_i = X_i a(\sum_{k=1}^{d_i} w_k x_k + b); \quad (9.41)$$

$$\text{на етапі тестування} — O_i = qa(\sum_{k=1}^{d_i} w_k x_k + b).$$

Можна використати й інший підхід — т. зв. *зворотне видалення* [9.1, 9.2]. У цьому випадку ми множимо функцію активації на відповідний коефіцієнт не на етапі тестування, а на етапі навчання. Цей коефіцієнт дорівнює оберненій величині ймовірності того, що нейрон залишиться у мережі:

$$\frac{1}{1 - \rho} = \frac{1}{q}.$$

У цьому разі вихід i-го прихованого нейрона дорівнює нулю:

$$\text{на етапі навчання} — O_i = \frac{1}{q} X_i a(\sum_{k=1}^{d_i} w_k x_k + b);$$

$$\text{на етапі тестування} — O_i = a(\sum_{k=1}^{d_i} w_k x_k + b).$$

У випадку прямого видалення виникає проблема вимушеної зміни нейронної мережі для тестування, оскільки без множення на нейрон повертатиме сигнал, який є вищим за очікуваний наступними нейронами, тому частіше використовується реалізація зворотного видалення.

Ансамблевий метод «бегінг»

Одним із способів створення ансамблів мереж є навчання програми на різних навчальних вибірках, отриманих у результаті випадкового процесу, який називається бегінгом («*bagging*», скорочення від «*bootstrap aggregating*»). Бегінг — це метод зменшення похибки узагальнення завдяки об'єднанню декількох моделей [9.1]. Ідея полягає в тому, щоб навчити кілька різних моделей окремо, а потім усі моделі голосують на виході тестової вибірки.

Методи, що використовують цю стратегію, відомі як ансамблеві методи. Причина використання усереднення полягає в тому, що різні моделі зазвичай дають різні похибки у тестовій вибірці. Розглянемо, наприклад, набір з k регресійних моделей. Припустимо, що кожна модель робить по-

милку ϵ_i для кожного шаблона i, отриманого з багатовимірного нормального розподілу з дисперсіями $E[\epsilon_i^2] = v$ і коваріацією $E[\epsilon_i \epsilon_j] = c$.

Тоді усереднена похибка передбачення, зроблена всім ансамблем, дорівнює $\frac{1}{k}\sum_i c_i$. А середньоквадратична похибка ансамблю становить

$$E\left[\left(\frac{1}{k}\sum_i c_i\right)^2\right] = \frac{1}{k^2}E[\sum_{i=1}^{k}(\epsilon_i^2 + \sum_{j\neq i(j\pm i)}\epsilon_i\epsilon_j)] = \frac{1}{k}v + \frac{k-1}{k}c. \quad (9.42)$$

У разі, якщо всі похибки повністю корельовані і $c = v$, *MSE* дорівнює v, і тому усереднення моделей не допомагає взагалі. Однак, якщо похибки різних моделей некорельовані і $c = 0$, *MSE* ансамблю дорівнює $\frac{1}{k}v$ [9.1].

Це означає, що *MSE* ансамблю лінійно зменшується зі збільшенням розміру ансамблю. Інакше кажучи, в середньому ансамбль поводитиметься принаймні не гірше ніж будь-який з його членів, а якщо всі його члени роблять незалежні похибки, то ансамбль поводитиметься краще, ніж усі його члени, і тому середня похибка прогнозування після використання бегінгу буде значно менша, ніж результати прогнозування окремих моделей.

Висновки

Нейронні мережі глибокого навчання (*Deep Learning, DL*) є фундаментальною складовою сучасних методів машинного навчання, що забезпечують вирішення складних завдань у різних сферах науки і техніки. Основою цих мереж є спеціальні структурні блоки, зокрема автоенкодери та обмежені машини Больцмана, що дають можливість ефективно навчати глибокі архітектури.

Автоенкодери є одним із ключових елементів глибоких мереж, що використовуються для вилучення суттєвих ознак із даних та їх ефективного стиснення. Їхня математична модель передбачає побудову симетричних нейронних мереж, які складаються з енкодера, що перетворює вхідні дані у приховане подання, та декодера, що відновлює вихідні дані з цього подання. Обмежені машини Больцмана, своєю чергою, реалізують стохастичні енергетичні моделі, які дають змогу будувати ймовірнісні подання складних багатовимірних розподілів.

Для ефективного навчання блоків глибоких мереж використовується алгоритм контрастної дивергенції (КД-k), який є основою для обмежених машин Больцмана. За допомогою цього алгоритму можна швидко знаходити параметри моделі, використовуючи процедуру наближеного байєсівського оновлення. Аналіз властивостей КД-k показує, що він дає змогу зменшити обчислювальну складність навчання, однак може мати проблеми з точністю апроксимації розподілів у випадках складних багатовимірних даних.

Методи оптимізації відіграють ключову роль у навчанні нейронних мереж, забезпечуючи корекцію вагових коефіцієнтів та пришвидшуючи збіжність моделей. Градієнтний спуск є базовим підходом до навчання, проте його варіації, такі як *Adagrad, Adam* і *RMSProp*, уможливлюють адаптивну зміну швидкості навчання для покращення збіжності та уникнення локальних мінімумів. *Adagrad* використовує змінний коефіцієнт навчання для кожного параметра, що допомагає у випадках неоднорідних даних, тоді як *Adam* поєднує адаптивність *Adagrad* з інерцією моменту, забезпечуючи стабільність оновлень ваг. *RMSProp* дає змогу згладжувати градієнти для ефективного навчання моделей із нерівномірним розподілом параметрів.

Одним із підходів до побудови архітектур глибокого навчання є використання стеків автоенкодерів та обмежених машин Больцмана. Такі стеки дають змогу виконувати пошарове навчання, що зменшує проблему згасання або вибуху градієнта, характерну для методу *Backpropagation*. Однак цей підхід не вирішує проблеми вибору оптимальної структури мережі, що потребує додаткових методів, зокрема, алгоритмів самоорганізації, які можуть визначати структуру мережі на основі вхідних даних.

Регуляризація є однією з основних проблем у контексті навчання глибоких нейронних мереж, оскільки може запобігати перенавчанню та покращувати узагальнювальні властивості моделей. Серед поширених методів регуляризації можна виділити *Lp*-регуляризацію, що накладає штраф на ваги нейронної мережі, ранню зупинку (*Early Stopping*), яка контролює момент припинення навчання на основі помилки на тестовій вибірці, а також методи *Dropout* і ансамблеве навчання (*Bagging*). Застосовуючи метод *Dropout*, можна випадковим чином вилучати нейрони під час навчання, що запобігає їхній надмірній спеціалізації, тоді як *Bagging* покращує узагальнювальні можливості моделі шляхом навчання кількох незалежних моделей та їхнього об'єднання. Отже, застосовуючи ці методи, можна зменшити похибку на тестовій вибірці за рахунок її збільшення на навчальній вибірці, що сприяє кращому узагальненню та підвищенню стійкості моделі до нових даних.

СПИСОК ЛІТЕРАТУРИ

9.1. Gooodfellow I., Bengio Y., Courville A. Deep Learning. 2nd ed. MIT Press, 2016. 772 p. URL: https://mitpress.mit.edu/9780262035613/deep-learning/

9.2. LeCun Y., Bengio Y., Hinton G.E. Deep learning. 2015. *Nature.* Vol. 521. No. 7553. P. 436—444. URL: www.cs.toronto.edu/~hinton/absps/NatureDeepReview.pdf

9.3. Bengio Y. Learning deep architectures for AI. *Foundations and trends in Machine Learning.* 2009. Vol. 2. No. 1. P. 1—127. URL: www.nowpublishers.com/article/Details/MAL-006.

9.4. Bodyanskiy Y., Zaychenko Y., Hamidov G. Hybrid Deep Learning Networks Based on Self-Organization and Their Applications. Cambridge Scholars Publishing, UK, 2024. URL: www.cambridgescholars.com/product/978-1-5275-XXXX-X.

9.5. Haykin S. Neural Networks and Learning Machines. 3rd ed. Upper Saddle River, NJ: Pearson Education, 2009. 936 p. URL: www.pearson.com/store/p/neural-networks-and-learning-machines/P100000028858.

9.6. Adeli H., Hung S.-L. Machine Learning: Neural Networks, Genetic Algorithms, and Fuzzy Systems. New York: John Wiley & Sons, 1995. 211 p.

9.7. Paterson J., Gibson A. Deep Learning: A Practitioner's Approach. 1st ed. Kindle Edition, 2017. 538 p.

9.8. Vincent P. A connection between score matching and denoising autoencoders. *Neural computation*. 2011. Vol. 23. No. 7. P. 1661—1674. URL: https://www.iro.umontreal.ca/~vincentp/Publications/smdae_techreport.pdf

9.9. Vincent P., Larochelle H., Bengio Y., Manzagol P.A. Extracting and composing robust features with denoising autoencoders. *Proceedings of the 25th international conference on Machine learning*. 2008. P. 1096—1103. URL: https://www.iro.umontreal.ca/~vincentp/Publications/denoising_autoencoders_tr1316.pdf

9.10. Kingma D.P., Welling M. (2013). Auto-encoding variational bayes. arXiv preprint arXiv: https://arxiv.org/abs/1312.6114

9.11. Hubel D.H., Wiesel T.N. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*. 1962. Vol. 160. No. 1. P. 106—154. URL: https://www.gatsby.ucl.ac.uk/teaching/courses/tn1-2025/additional/systems/JPhysiol-1962-Hubel-106-54.pdf

9.12. Fukushima K. Neocognitron: A self-organizing neural network model for amechanism of pattern recognition unaffected by shift inposition. *Biological cybernetics*. 1980. Vol. 36. No. 4. P. 193—202. URL: https://link.springer.com/article/10.1007/BF00344251.

9.13. Ackley D.H., Hinton G.E., Sejnowski T.J. «A Learning Algorithm for Boltzmann Machines». 1985. Vol. 9. No. 1. P. 147—169. URL: https://direct.mit.edu/books/edited-volume/5431/chapter-abstract/3958544/1985-David-H-Ackley-Geoffrey-E-Hinton-and-Terrence?redirectedFrom=fulltext.

9.14. Meyder A., Kiderlen C. Fundamental properties of Hopfield Networks and Boltzmann Machines for Associative Memories. 2008. P. 1—16. URL: http://uppsala.kiderlen.de/ml/mlproject.pdf.

9.15. Apolloni B., Falco D. Learning by parallel boltzmann machines. 1991. P. 1—8. URL: https://www.researchgate.net/publication/224181407_Parallel_Tempering_is_Efficient_for_Learning_Restricted_Boltzmann_Machines.

9.16. Hinton G. A Practical Guide to Training Restricted Boltzmann Machines. 2010. P. 1—20. URL: https://www.cs.toronto.edu/~hinton/absps/guideTR.pdf

9.17. Sutskever I., Tieleman T. On the convergence properties of contrastive divergence. 2010. P. 789—795. URL: http://proceedings.mlr.press/v9/sutskever10a/sutskever10a.pdf

9.18. Carreira-Perpignan M.A., Hinton G.E. On contrastive divergence learning. 2005. P. 1—8. URL: https://www.cs.toronto.edu/~hinton/absps/cdmiguel.pdf

9.19. Yoon S., Kwon D., Hwang H., Noh Y-K., Park Frank C. Generalized Contrastive Divergence: Joint Training of Energy-Based Model and Diffusion Model through Inverse Reinforcement Learning. 2023. URL: https://arxiv.org/abs/2312.03397.

9.20. Tieleman T. Training restricted Boltzmann machines using approximations to the likelihood gradient. 2008. P. 1—9. URL: https://www.cs.toronto.edu/~tijmen/pcd/pcd.pdf

9.21. Berglund M. Stochastic Gradient Estimate Variance in Contrastive Divergence and Persistent Contrastive Divergence. 2016. P. 521—526. URL: https://www.esann.org/sites/default/files/proceedings/legacy/es2016-6.pdf

9.22. Cho K., Raiko T., Ilin A. Enhanced Gradient for Training Restricted Boltzmann Machines. 2013. P. 805—831. URL: http://users.ics.aalto.fi/alexilin/papers/nc2013.pdf

9.23. Larochelle H. Online Lectures: Restricted Boltzmann machine — example. 2015. URL: https://www.youtube.com/watch?v=n26NdEtma8U.

9.24. Larochelle H., Bengio Y. Classification using Discriminative Restricted Boltzmann Machine. P. 1—8. 2008. URL: https://icml.cc/Conferences/2008/papers/601.pdf

9.25. Dahl G., Adams R., Larochelle H. Training Restricted Boltzmann Machines on Word Observation. 2012. P. 1—8. arXiv: https://arxiv.org/pdf/1202.5695.pdf

9.26. Goodfellow I., Bengio Y., Courville A. Autoencoders. MIT Press, 2016. P. 500—523. URL: https://www.deeplearningbook.org/contents/autoencoders.html#pf2

9.27. Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre at al. Learning phrase representations using RNN encoder-decoder for statistical machine translation. 2014. arXiv: https://arxiv.org/abs/1406.1078

9.28. Hinton G. Deep belief networks. *Scholarpedia* 4(5):5947. 2009. URL: http://www.scholarpedia.org/article/Deep_belief_networks.

9.29. Erhan D., Bengio Y., Courville A. at al. Why Does Unsupervised Pre-training Help Deep Learning? *Journal of Machine Learning Research*. N. 11. 2010. P. 625—660. arXiv: http://www.jmlr.org/papers/volume11/erhan10a/erhan10a.pdf

9.30. Hinton G., Osindero S., Teh Y. A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation*. 2006. 18. P. 1527—1554. URL: https://www.cs.toronto.edu/~hinton/absps/ncfast.pdf

9.31. Kingma D.P., Ba J. Adam: a method for stochastic optimization. *Conference paper at ICLR 2015*. P. 1—15. arXiv: https://arxiv.org/pdf/1412.6980

9.32. Ruder S. (2016). An overview of gradient descent optimization algorithms. 15 Jun 2017. Vol. 2. arXiv: https://arxiv.org/abs/1609.04747

9.33. Villarraga D. SYSEN 6800 Fall 2021. AdaGrad. URL: https://optimization.cbe.cornell.edu/index.php?title=AdaGrad.

9.34. Brownlee J. Gentle Introduction to the Adam Optimization Algorithm for Deep Learning. January 13, 2021. URL: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

РОЗДІЛ 10

ГІБРИДНІ НЕЧІТКІ МЕРЕЖІ ГЛИБОКОГО НАВЧАННЯ, АЛГОРИТМИ НАВЧАННЯ ТА ОПТИМІЗАЦІЇ

10.1. Архітектура нейронечіткої системи МГУА, що еволюціонує

10.2. Нейронечітка мережа з малою кількістю параметрів налаштування як вузол МГУА-системи

10.3. Експериментальні дослідження гібридних нейронечітких мереж

10.4. Дослідження гібридних неонечітких НМ на основі МГУА та оптимізація структури в задачах прогнозування

10.5. Оптимізація структури та дослідження гібридних неонечітких нейронних мереж МГУА в задачах прогнозування

10.6. Неонечіткі системи МГУА з розширеними неонечіткими нейронами

10.7. Гібридна система обчислювального інтелекту на основі пакетного та групового оброблення даних

Висновки

Список літератури

Останнім часом мережі глибокого навчання становлять значний інтерес для вчених у галузі нейронних мереж (НМ), їх широко застосовують для вирішення різноманітних завдань у сфері обчислювального інтелекту, наприклад прогнозування, розпізнавання образів, обробки медичних зображень та експрес-діагностики. Було розроблено і досліджено багато алгоритмів навчання, виявлено недоліки глибинного навчання та запропоновано можливі способи їх усунення.

Однак традиційні алгоритми глибокого навчання коригують лише ваги зв'язків у НМ і не змінюють структуру мережі. Разом з тим ефективність навчання можна суттєво підвищити, якщо в процесі навчання адаптувати не тільки значення ваг, а й структуру власне нейронної мережі. Для цього можна з успіхом застосувати метод самоорганізації — т. зв. метод групового урахування аргументів (МГУА, *Group Method of Data Handling, GMDH,* див. розд. 4). Основними перевагами МГУА є: можливість автоматичної побудови структури моделі в процесі роботи алгоритму; робота з невеликою вибіркою; висока швидкість навчання через невелику кількість вхідних даних [10.1—10.4]. Завдяки цим властивостям застосування МГУА для глибокої оптимізації структури глибоких НМ видається дуже привабливим.

У попередні роки було розроблено МГУА-нейронні мережі з активними нейронами [10.3, 10.4], *R*-нейронами [10.5], *Q*-нейронами [10.6], МГУА-вейвлет-нейронечіткі системи [10.7, 10.8] та МГУА-неонечіткі системи [10.7—10.10].

В останні два роки було запропоновано і досліджено новий клас гібридних нейронних мереж з використанням цього підходу — гібридні глибокі МГУА-нейронечіткі мережі [10.11] на основі самоорганізації, які підтвердили високу ефективність у задачах прогнозування в економіці. Як вузли гібридних нейронечітких мереж використовують нейрони Дж. Менделя (Jerry M. Mendel) та Л. Ванга (Laurent Wang) з двома входами [10.12, 10.13]. Але недоліком МГУА-нейронечітких мереж є необхідність адаптувати в цих мережах не лише вихідні ваги, а й параметри функції належності. Ефективність навчання можна суттєво підвищити, якщо навчати лише ваги правил. Для цього дуже перспективним видається застосування неонечітких нейронів як вузлів гібридних мереж.

Метою подальших досліджень було розроблення нового класу гібридних мереж глибокого навчання на основі неонечітких нейронів, знаходження їх оптимальних параметрів, оптимізація структури та дослідження їх ефективності в задачах прогнозування в економіці та фінансовій сфері [10.14].

У цьому розділі розглянуто, проаналізовано і досліджено архітектуру, властивості, алгоритми навчання та застосування запропонованих гібридних нейронечітких та неонечітких мереж.

10.1. Архітектура нейронечіткої системи МГУА, що еволюціонує

Архітектуру гібридної нечіткої мережі МГУА, що еволюціонує, показано на рис. 10.1 [10.14, 10.15].

На вхідний шар системи подається $(n \times 1)$-вимірний вектор вхідних сигналів $x = (x_1, x_2, \ldots, x_n)^T$. Далі цей сигнал подається на перший прихований шар, який містить $n_1 = c_n^2$ вузлів-нейронів, кожен з яких має лише два входи. На виходах вузла $N^{[1]}$ першого прихованого шару формуються вихідні сигнали $y_l^{[1]}$, $l = 1, 2, \ldots, 0{,}5n(n-1) = c_n^2$. Потім ці сигнали подаються на блок селекції першого прихованого шару $SB^{[1]}$, який вибирає серед вихідних сигналів $\hat{y}_l^{[1]} \cdot n_1^{\star}$ найточніші за прийнятим критерієм (найчастіше за середньоквадратичною похибкою $\sigma^2_{y_{[1]}}$).

З цих $n_1^{\star}$ найкращих виходів першого прихованого шару $\hat{y}_l^{[1]\star} n_2$ формуються парні комбінації $\hat{y}_l^{[1]\star}\hat{y}_p^{[1]}$, які подаються на другий прихований шар, утворений нейронами $N^{[2]}$.

Серед сигналів цього шару $\hat{y}_l^{[2]}$ блок відбору $SB^{[2]}$ відбирає F найкращих нейронів за точністю (наприклад, за значенням $\sigma^2_{y_l^{[2]}}$, якщо найкращий сигнал другого шару кращий за найкращий сигнал першого прихованого шару $\hat{y}_l^{[1]\star}$. Інші приховані шари формують сигнали аналогічно другому.

Процес еволюції системи триває доти, доки найкращий сигнал блока селекції $SB^{[s+1]}$ не стане гіршим за найкращий сигнал попереднього шару, тобто $\sigma^2_{y_l^{[s+1]}} > \sigma^2_{y_l^{[s]}}$.

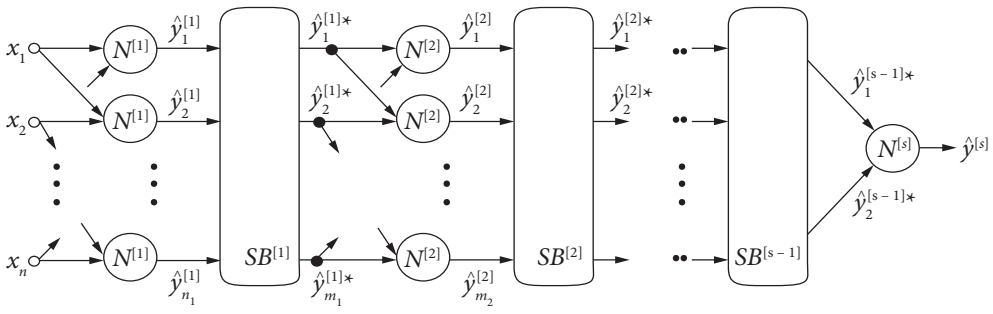


Рис. 10.1. Еволюція МГУА-системи

Тоді повертаємося до попереднього шару і вибираємо його найкращий вузловий нейрон $N_i^{[s]}$ для формування вихідного сигналу системи. І, рухаючись від цього нейрона (вузла) по його зв'язках назад і послідовно проходячи всі попередні шари, визначаємо структуру нейронечіткої мережі МГУА. Слід підкреслити, що завдяки алгоритму МГУА отримуємо не тільки оптимальну структуру мережі, а й добре навчену мережу. Крім того, оскільки навчання виконується послідовно шар за шаром, проблем високої розмірності, а також градієнта, який зникає або вибухає, немає. Це дуже важливо для мереж глибокого навчання.

10.2. Нейронечітка мережа з малою кількістю параметрів налаштування як вузол МГУА-системи

Розглянемо архітектуру вузла, зображену на рис. 10.2 і запропоновану як нейрон МГУА-системи. Ця архітектура фактично є нейронечіткою системою Ванга — Менделя з двома входами — x_i та x_j і одним виходом [10.14, 10.15].

На вхід вузла подається двовимірний вектор сигналів $x(k) = (x_i(k), x_j(k))^T$, де $k = 1, 2, \ldots, N$ — порядок спостереження у навчальній вибірці або поточний дискретний час.

Перший шар вузла містить $2h$ функцій належності $\mu_{pi}((x_i(k))$, $\mu_{pj}((x_i(k))$, $p = 1, 2, \ldots, h$ і забезпечує фазифікацію вхідних змінних. Як функції належності зазвичай використовують дзвоноподібні конструкції з нечітко локальною рецептивною підтримкою. Це дає змогу уникнути появи «розривів» у фазифікованому просторі під час використання розбиття вхідного

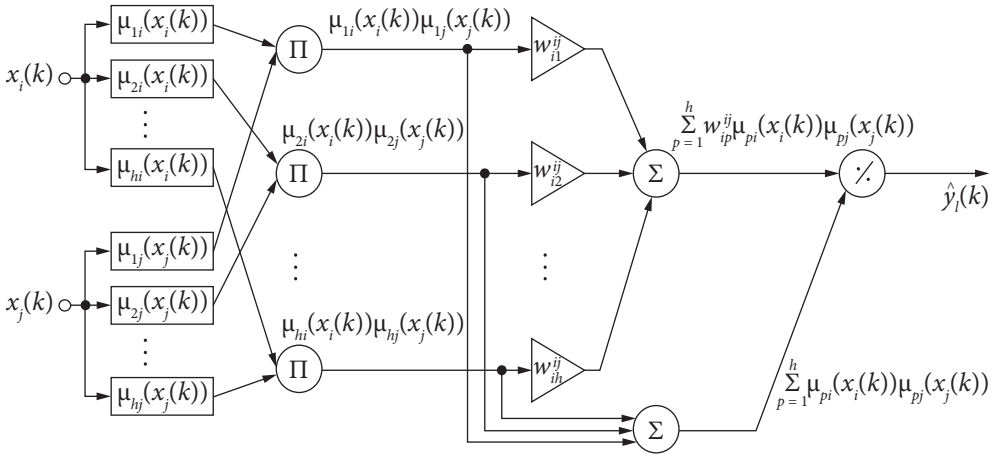


Рис. 10.2. Вузол нейронечіткої системи МГУА

простору [10.15]. Як функції належності першого шару зазвичай використовують гауссіани

$$\begin{cases} \mu_{pi}(x_i(k)) = \exp\left(-\dfrac{(x_i(k) - c_{pi})^2}{2\sigma_i^2}\right), \\ \mu_{pj}(x_j(k)) = \exp\left(-\dfrac{(x_j(k) - c_{pj})^2}{2\sigma_j^2}\right), \end{cases} \tag{10.1}$$

де c_{pi}, c_{pj} — параметри, що визначають центри функцій належності; σ_i^2, σ_j^2 — параметри ширини цих функцій. Другий шар забезпечує агрегування рівнів належності. Він складається з h одиниць множення і формує двовимірні радіально-базисні функції активації

$$\tilde{x}_p(k) = \mu_{pi}(x_i(k))\mu_{pj}(x_j(k)), \tag{10.2}$$

і для гауссіанів з однаковими значеннями $\sigma_i^2 = \sigma_j^2 = \sigma^2$ можемо записати

$$\tilde{x}_p(k) = \exp\left(-\frac{||x(k) - c_p||^2}{2\sigma^2}\right); \tag{10.3}$$

тут $c_p = (c_{pi}, c_{pj})^T$, тобто елементи першого та другого шарів обробляють вхідний сигнал аналогічно до R-нейронів нейронних мереж з радіально-базисними функціями.

Третій шар — це синаптичні ваги, які налаштовуються в процесі навчання. Виходами цього шару є значення

$$w_{lp}^{ij}\,\mu_{pi}(x_i(k))\,\mu_{pj}(x_j(k)) = w_{lp}^{ij}\,\tilde{x}_p(k). \tag{10.4}$$

Четвертий шар утворений двома підсумовувальними блоками і обчислює суми вихідних сигналів другого і третього прихованих шарів:

$$\begin{cases} \sum_{p=1}^{h} w_{lp}^{ij}\,\mu_{pi}(x_i(k))\,\mu_{pj}(x_j(k)) = \sum_{p=1}^{h} w_{lp}^{ij}\,\tilde{x}_p(k), \\ \sum_{p=1}^{h} \mu_{pi}(x_i(k))\,\mu_{pj}(x_j(k)) = \sum_{p=1}^{h} \tilde{x}_p(k). \end{cases} \tag{10.5}$$

Нарешті, в п'ятому шарі здійснюється нормалізація, в результаті чого формується вихідний сигнал вузла [10.15]:

$$\hat{y}_l(k) = \frac{\sum_{p=1}^{h} w_{lp}^{ij}\,\mu_{pi}(x_i(k))\,\mu_{pj}(x_j(k))}{\sum_{p=1}^{h} \mu_{pi}(x_i(k))\,\mu_{pj}(x_j(k))} = \frac{\sum_{p=1}^{h} w_{lp}^{ij}\,\tilde{x}_p(k)}{\sum_{p=1}^{h} \tilde{x}_p(k)} =$$

$$= \sum_{p=1}^{h} w_{lp}^{ij}\,\varphi_{pi}(x(k)) = (w_l^{ij})^T\,\varphi^{ij}(x(k)), \tag{10.6}$$

де

$$\varphi^{ij}(x(k)) = (\varphi_1^{ij}(x(k)), \ldots, \varphi_p^{ij}(x(k)), \ldots, \varphi_h^{ij}(x(k)))^T;$$

$$\varphi^{ij}(x(k)) = \mu_{pi}(x_i(k))\,\mu_{pj}(x_j(k))\,(\textstyle\sum_{p=1}^{h} \mu_{pi}(x_i(k))\,\mu_{pj}(x_j(k)))^{-1};$$

$$w_l^{ij} = (w_{l1}^{ij}, \ldots, w_{lp}^{ij}, \ldots, w_{lh}^{ij})^T.$$

Неважко помітити, що вузол реалізує нелінійне відображення вхідних сигналів у вихідний сигнал подібно до нейронної мережі з нормалізованою радіальною базисною функцією, однак нейронечітка система (*NFS*) містить значно меншу кількість h параметрів, що коригуються, порівняно з нейронною мережею.

Використовуючи введені позначення і вибравши перетворення в кожному вузлі стандартного МГУА у вигляді

$$\hat{y}_1(k) = w_{l0}^{ij} + w_{l1}^{ij} x_i(k) + w_{l2}^{ij} x_j(k), \tag{10.7}$$

що містить три невідомі параметри, легко побачити, що з трьома функціями належності на кожному вході запропонованого вузла отримаємо ті самі три синаптичні ваги, які необхідно скоригувати.

У найпростішому випадку оцінку цих синаптичних ваг можна виконати за допомогою звичайного методу найменших квадратів (МНК), який традиційно використовують у МГУА. Якщо подано всю навчальну вибірку, можна використовувати МНК у його пакетній формі [10.15]:

$$w_l^{ij}(N) = (\Sigma_{k=1}^{N} \varphi^{ij}(x(k))\varphi^{ij}(x(k))^T)^+ \Sigma_{k=1}^{N} \varphi^{ij}(x(k))y(k), \tag{10.8}$$

де $y(k)$ — зовнішній опорний сигнал. Якщо точки навчальної вибірки подаються послідовно в режимі онлайн, то використовується рекурентна форма МНК у вигляді

$$\begin{cases} w_l^{ij}(k) = w_l^{ij}(k-1) + \dfrac{p^{ij}(k-1)(y(k) - (w_l^{ij}(k-1))^T \varphi^{ij}(x(k)))\varphi^{ij}(x(k))}{1 + (\varphi^{ij}(x(k)))^T p^{ij}(k-1)\varphi^{ij}(x(k))}; \\ P^{ij}(k) = P^{ij}(k-1) - \dfrac{p^{ij}(k-1)\varphi^{ij}(x(k))\varphi^{ij}(x(k))^T p^{ij}(k-1)}{1 + (\varphi^{ij}(x(k)))^T p^{ij}(k-1)\varphi^{ij}(x(k))}. \end{cases} \tag{10.9}$$

10.3. Експериментальні дослідження гібридних нейронечітких мереж

У першому експерименті було розглянуто задачу прогнозування цін на акції корпорації *Microsoft.* Як вхідну вибірку було використано ціни на акції корпорації *Microsoft* з 01.11.14 по 29.12.14 [10.15]. Обсяг вибірки становив 64 пункти. Навчальна вибірка містила 62 точки, тестова вибірка — чотири точки. Інтервал прогнозування — чотири кроки вперед, перші два кроки перевіряються за наявними даними. Побудована МГУА-нейронечітка мережа мала шість нечітких входів. Результати експерименту наведено в табл. 10.1.

Точність прогнозування гібридної нейронечіткої мережі порівнювали з методом МГУА та каскадною нейронечіткою мережею. Результати про-

гнозування цими методами наведено в табл. 10.2. Як випливає з результатів експерименту, МГУА-нейронечітка мережа продемонструвала кращу точність прогнозування, ніж каскадна нейронечітка мережа. Її значення *MAPE* (*Mean Absolute Percentage Error*) не перевищує 0,4 %.

У наступному експерименті порівнювали час навчання нейронечіткої мережі МГУА та каскадної нечіткої мережі [10.15, 10.16]. У табл. 10.3 наведено час навчання (в секундах) для гібридної нейронечіткої мережі МГУА та повної каскадної нейронечіткої мережі. Як початкову вибірку було використано ціни на акції компанії *Microsoft* за той самий період — з 01.11.14 по 29.12.14.

Таблиця 10.1

Результати прогнозування для гібридних мереж МГУА

Дата	Реальне значення	Прогнозоване значення	Абсолютна похибка	Відносна похибка, %
26.12.14	18030,2	17971,63	58,577	0,325
24.12.14	18053,7	17991,94	61,772	0,342

Таблиця 10.2

Результати прогнозування (*MAPE*) для різних нейронечітких мереж та МГУА

Реальне значення	Значення MAPE, %, для мереж		
	нейронечіткої мережі МГУА	МГУА	каскадної нейронечіткої мережі
48,14	0,623	1,20	3,40
47,88	2,13	1,94	2,54
Середнє	1,377	1,57	2,97

Таблиця 10.3

Тривалість навчання гібридної мережі МГУА та каскадної мережі

Кількість входів	Час навчання, с, для мереж	
	гібридної мережі МГУА	каскадної мережі
2	0,004	0,015
4	0,009	0,021
6	0,013	0,037
8	0,021	0,048
10	0,030	0,053

У наступних експериментах досліджували ефективність гібридної нейронечіткої мережі у прогнозуванні індексу *NASDAQ.* Дані взято за період з 13.11.17 по 29.11.19. Розмір вибірки становив 510 точок. Як вихідну змінну взято ціну закриття індексу *NASDAQ* на наступний день.

У першому експерименті було досліджено залежність точності від кількості входів для гібридної неонечіткої мережі. У табл. 10.4 наведено результати прогнозування з різною кількістю входів, 8 функціями належності на змінну (параметр h) та співвідношенням навчальної / перевірної вибірок 70/30.

Аналізуючи результати, можна дійти висновку, що зі збільшенням кількості входів значення *MAPE* спочатку зменшується, потім досягає мінімуму і після цього починає зростати. Це повністю відповідає принципу самоорганізації методу МГУА [10.15]. Найкраще значення було отримано за таких значень параметрів: кількість входів $n = 5$, $h = 8$, кількість шарів — 4, а значення *MAPE* становить 3,91.

Для оцінки ефективності прогнозування гібридної мережі її було порівняно з каскадною нечіткою мережею [10.14, 10.15] та МГУА за однакових даних. У каскадній нечіткій мережі використовували такі значення параметрів: кількість входів $n = 9$, кількість правил — 9, кількість каскадів — 3. Порівняльні результати прогнозування наведено в табл. 10.5, навчальна вибірка — 70 %.

Аналізуючи ці результати, можна легко зробити висновок, що найкращу точність має запропонована гібридна нейронечітка мережа, на другому місці — метод МГУА, на третьому — каскадна нейронечітка мережа.

Таблиця 10.4

***MAPE*-прогнозування залежно від кількості входів для гібридних нейронечітких мереж**

Кількість входів	2	3	4	5	6	7	8	9	10
MAPE	5,2	4,7	4,33	3,91	4,22	4,72	5,24	5,53	5,85

Таблиця 10.5

Значення *MAPE* для різних методів прогнозування

Кількість входів	Значення MAPE, %, для мереж		
	гібридної нейронечіткої	МГУА	каскадної неонечіткої
4	4,30	4,19	6,04
5	3,93	4,11	6,09
6	4,35	5,53	8,01
7	4.80	6,26	8,68

Висновок. Розглянуто та досліджено гібридну нейронечітку мережу на основі самоорганізації. Як вузли цієї мережі використовуються елементарні нечіткі нейронні мережі Менделя — Ванга з двома входами.

Розроблено алгоритм синтезу структури гібридної нейронної мережі на основі методу самоорганізації МГУА онлайн. Перевагою цього алгоритму є синхронність синтезу структури мережі з її навчанням, шар за шаром, до виконання умови зупинки.

Проведено експериментальне дослідження запропонованої гібридної МГУА-нейронечіткої мережі в завданнях прогнозування цін на акції корпорації *Microsoft,* у результаті якого знайдено оптимальні значення параметрів мережі: кількість входів, кількість фазифікаторів, а також оптимальну структуру мережі для цієї задачі. Також було проведено порівняльні експерименти з прогнозування методом МГУА та повної каскадної неонечіткої мережі, які показали, що гібридна нейронна мережа має вищу точність прогнозування цін на акції та фондові індекси порівняно з альтернативними методами прогнозування.

У результаті проведених експериментів було зроблено такі висновки.

1. Варіація кількості входів у гібридній нейронечіткій мережі впливає на якість прогнозу: зі збільшенням кількості входів критерій *MAPE* спочатку знижується, досягає мінімуму, а потім починає зростати.

2. Застосування запропонованого методу синтезу структури та навчання гібридної мережі дає змогу скоротити час навчання і підвищити точність прогнозування порівняно з каскадною нейронною мережею.

3. У цілому, запропонований метод навчання гібридної мережі на основі МГУА має такі переваги: немає проблем з алгоритмами глибокого навчання — зникнення або вибуху градієнта через алгоритм навчання, який працює послідовно.

10.4. Дослідження гібридних неонечітких НМ на основі МГУА та оптимізація структури в задачах прогнозування

У попередньому підрозділі розглянуто і досліджено новий клас мереж глибокого навчання — гібридні глибокі МГУА-нейронечіткі мережі [10.14, 10.15], які підтвердили високу ефективність у задачах прогнозування в економіці.

Однак недоліком МГУА-нейронечітких мереж є те, що в цих мережах потрібно адаптувати не лише вихідні ваги, а й параметри функції належності. Ефективність навчання можна суттєво підвищити, якщо навчати лише ваги правил. Для цього дуже перспективним видається застосування неонечітких нейронів як вузлів гібридних мереж.

Неонечіткий нейрон. Його структура та алгоритм навчання

Архітектура неонечіткого нейрона (*neo-fuzzy neuron, NFN*) була запропонована Такеші Ямакава (Takeshi Yamakawa) та співавторами в роботі [10.16]. Серед найважливіших переваг *NFN* автори називають високу швидкість навчання, обчислювальну простоту, можливість знаходження глобального мінімуму критерію навчання в реальному часі, а також те, що він характеризується нечіткими лінгвістичними правилами «якщо, то». Нечіткий нейрон (*neo-fuzzy*) — це нелінійна система з багатьма входами і одним виходом (рис. 10.3).

Він реалізує відображення

$$\hat{y} = \Sigma_{i=1}^{2} f_i(x_i), \tag{10.10}$$

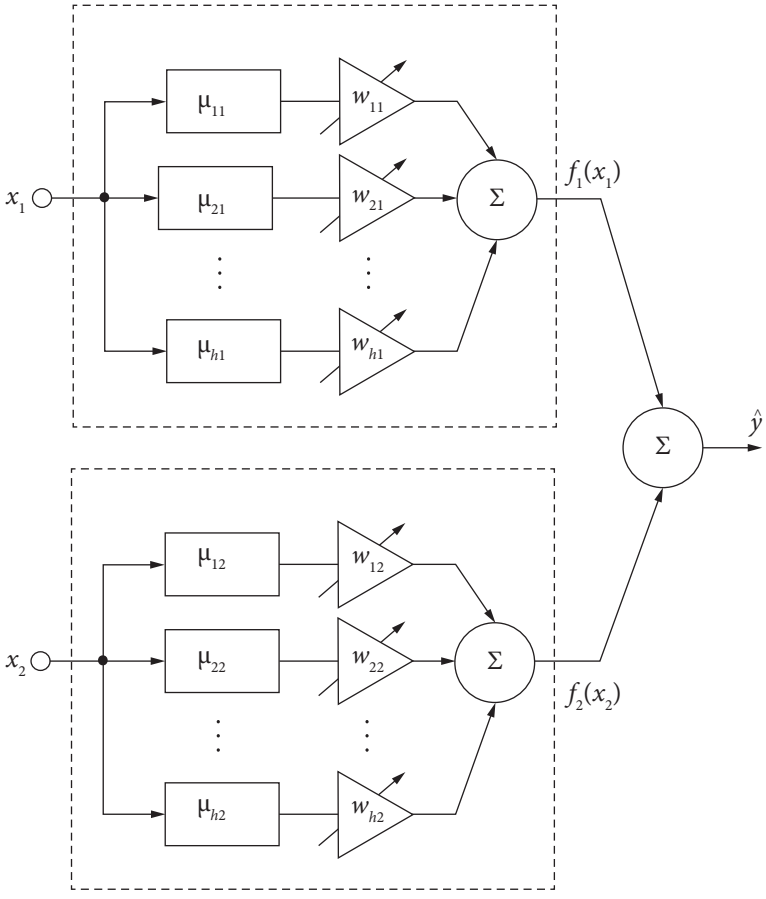


Рис. 10.3. Неонечіткий нейрон

де x_i — i-й вхід ($i = 1, 2, \ldots, n$), $\hat{y}$ — система виходу. Структурними блоками неонечіткого нейрона є нелінійні синапси *NSi*, які виконують перетворення i-го вхідного сигналу на вихідний

$$f_i(x_i) = \sum_{j=1}^{h} w_{ji}\, \mu_{ji}(x_i)$$

і реалізують нечіткий висновок: якщо x_i дорівнює x_{ji}, то вихід дорівнює w_{ji}, де x_{ji} — нечітка множина з функцією належності μ_{ji}, w_{ji} є синглетоном (синаптичною вагою) в послідовності [10.16]. Як легко побачити, нелінійний синапс фактично реалізує нечіткий висновок Тоширо — Мінору (Takagi Toshiro, Sugeno Minoru) нульового порядку.

Умовно функції належності $\mu_{ji}(x_i)$ в антецеденті є комплементарними трикутними функціями.

Неонечітка нейронна мережа та оптимізація її архітектури з використанням методу групового урахування аргументів

Неонечітка нейронна мережа (*Neo-Fuzzy Neural Network*) — це багатошарова архітектура з прямим зв'язком, що складається з нечітких нейронів. Структуру неонечіткої нейронної мережі [10.16] з n входами та m виходами зображено на рис. 10.4.

Така архітектура повністю збігається зі структурою тришарового перцептрона, за винятком того, що як елементарні вузли тут використовують не нейрони Розенблатта, а неонечіткі нейрони. Тому для налаштування вагових коефіцієнтів такої архітектури необхідно використовувати алгоритми навчання, розглянуті раніше. Як відомо, ці алгоритми досить складні з обчислювальної точки зору і працюють повільно.

Якщо ж використовувати неонечіткі нейрони, які мають лише два входи, то МГУА можна застосувати для синтезу неонечіткої нейронної мережі з оптимальною архітектурою.

Основна ідея алгоритму МГУА полягає в послідовному синтезі шарів нейронів доти, доки зовнішній критерій не почне зростати.

Алгоритм полягає у таких діях [10.17, 10.18].

1. Сформувати пари з виходів нечітких нейронів поточного шару (на першій ітерації використати набір вхідних сигналів). Кожна пара подається на відповідний нечіткий нейрон.

2. Використовуючи навчальну підвибірку, налаштувати синаптичні вагові коефіцієнти кожного нечіткого нейрона за допомогою алгоритму навчання.

3. На тестовій підвибірці обчислити значення зовнішнього критерію (регулярність, МНК) для кожного нечіткого нейрона:

$$\varepsilon_p^{[s]} = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} (y(i) - \hat{y}_p^{[s]}(i))^2, \qquad (10.11)$$

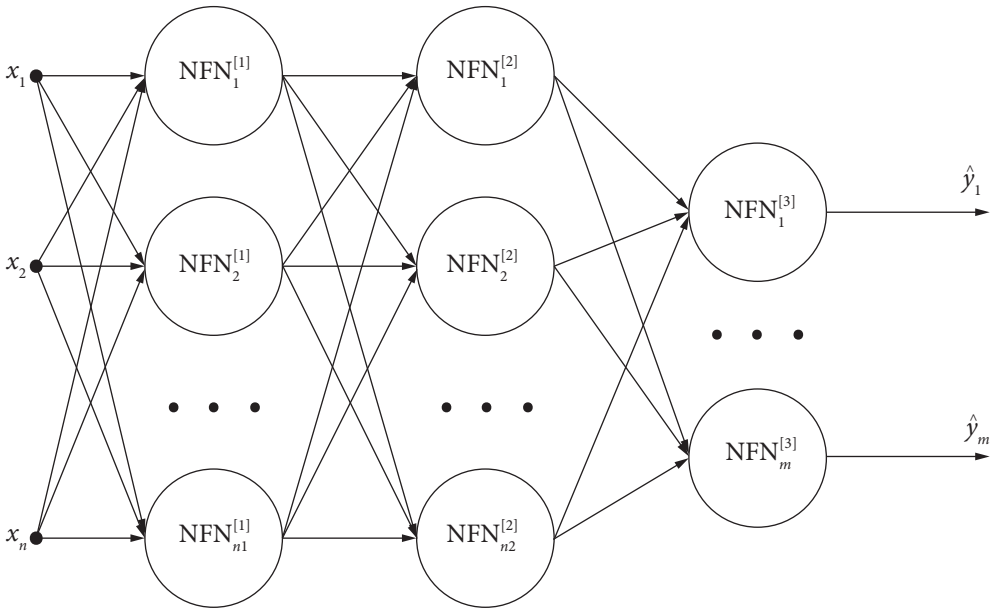


Рис. 10.4. Структура неонечіткої нейронної мережі

де N_{test} — розмір тестової підвибірки; s — номер шару; p — номер нейрона у поточному шарі $p = \overline{1, n_s}$; $\hat{y}_p^{[s]}(i)$ — сигнал відгуку p-го нейрона s-го шару на i-й вхідний вектор. Знайти мінімальне значення зовнішнього критерію для всіх нечітких нейронів поточного шару:

$$\varepsilon^{[s]} = \min_p \varepsilon_p^{[s]}. \tag{10.12}$$

Перевірити умову

$$\varepsilon^{[s]} > \varepsilon^{[s-1]}, \tag{10.13}$$

де $\varepsilon^{[s]}$, $\varepsilon^{[s-1]}$ — значення критерію для найкращих нейронів s-го та $(s-1)$-го шарів відповідно. Якщо умова (10.13) виконується, то повернутися до попереднього шару і знайти найкращий нейрон, який має мінімальне значення критерію (10.11). В іншому випадку вибрати F найкращих нейронів за значенням критерію (10.11) і перейти до кроку 1 для побудови наступного шару нейронів.

4. Визначити остаточну структуру мережі. Рухаючись назад від найкращого нейрона $(m-1)$-го шару вздовж вхідних зв'язків і послідовно проходячи всі шари нейронів, зберегти в кінцевій структурі тільки ті нейрони, які використовуються в наступному шарі.

Після завершення роботи МГУА можна стверджувати, що синтезовано остаточну оптимальну структуру неонечіткої нейронної мережі. Як бачимо, отримано не лише оптимальну структуру, а й навчену нейронну мережу, яка готова до оброблення нових даних. Однією з найважливіших

переваг використання МГУА для синтезу нечіткої архітектури є можливість використання простих, але дуже швидких процедур навчання для налаштування ваг нечітких нейронів, оскільки мережа навчається пошарово. Крім того, завдяки таким методам навчання виключаються типові недоліки глибокого навчання — зникнення градієнта та перенавчання.

10.5. Оптимізація структури та дослідження гібридних неонечітких нейронних мереж МГУА в задачах прогнозування

Метою цього підрозділу є побудова нової гібридної неонечіткої мережі глибокого навчання, знаходження її оптимальних параметрів, оптимізація структури та дослідження її ефективності в задачі прогнозування в економіці та фінансовій сфері.

Для експериментальних досліджень було розглянуто три задачі прогнозування [10.18].

1. **Задача прогнозування промислових цін.** Для цієї задачі як вихідні дані було взято індекс промислових цін виробників (ІЦВ) в Україні за період 2017—2018 рр. Як вхідні змінні було взято індекс оптових цін (ІОЦ), грошовий агрегат M_2, грошовий агрегат M_0, імпорт, експорт.

1.1. *Дослідження чутливості до співвідношення вибірок навчальна / перевірна.* У першому експерименті було досліджено чутливість співвідношення навчальна / тестова вибірка до точності прогнозування. З результатів прогнозування випливає, що найкращу точність було отримано за співвідношення вибірок навчальна / тестова 60/40. Дані для цього співвідношення вибірок наведено на рис. 10.5 та в табл. 10.6.

1.2. *Дослідження чутливості до кількості входів.* Цей експеримент проводився за співвідношень вибірок 50/50 і 60/40 та кількості входів $n = 3, 4, 5$. Сумарні результати за критерієм *MSE* для різних співвідношень наведено на рис. 10.6

Результати точності за критерієм *MAPE* наведено в табл. 10.7.

Аналіз даних експериментів свідчить, що найвищу точність було отримано для трьох входів за співвідношення вибірок навчальна / перевірна 60/40 за обома критеріями — *MSE* та *MAPE* [10.18].

1.3. *Дослідження чутливості до кількості вхідних даних.* Цей експеримент проводили за частки навчальної вибірки 50 % і 60 %, кількості входів $n = 3, 4, 5$.

В інших експериментах для прогнозування індексу цін виробників (ІЦВ) в Україні на період 2013—2020 рр. (поквартально) вхідними змінними слугували: індекс промислового виробництва, курс долара, зовнішні інвестиції, грошовий агрегат M_2.

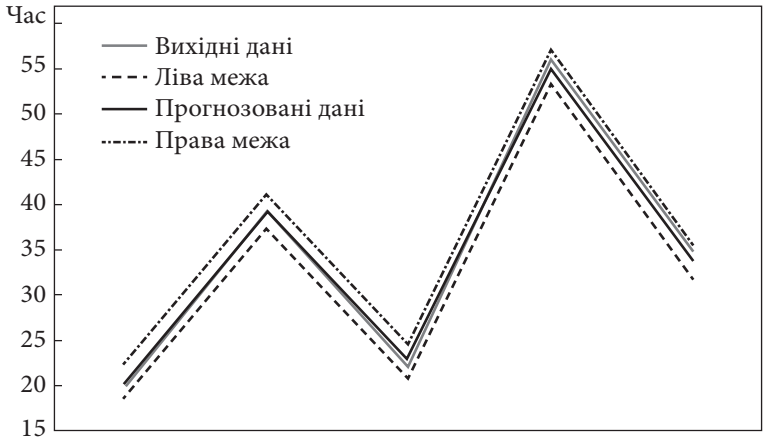


Рис. 10.5. Результати прогнозування для співвідношення вибірок навчальна / перевірна 60/40: *MSE* = 0,55, *MAPE* = 1,053 %

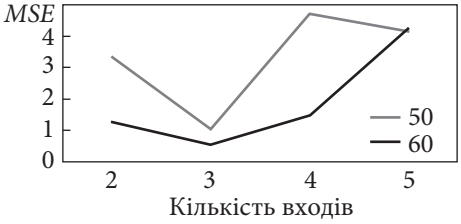


Рис. 10.6. Значення *MSE* для різної кількості входів за співвідношень вибірок навчальна / тестова 50/50 та 60/40

Таблиця 10.6

Значення *MSE* та *MAPE* за різних співвідношень вибірок навчальна / перевірна

Критерій	Співвідношення вибірок навчальна / перевірна			
	50/50	60/40	70/30	80/20
MSE	1,3237	0,6237	0,791	0,6352
MAPE, %	1,542	0,7212	1,67	0,7342

Таблиця 10.7

Критерій *MAPE* за відсотком навчальної вибірки для різної кількості входів

Кількість входів	*MAPE*, %, за частки навчальної вибірки, %	
	50	60
2	4,765	3,176
3	2,57	1,053
4	3,24	3,761
5	3,75	4,006

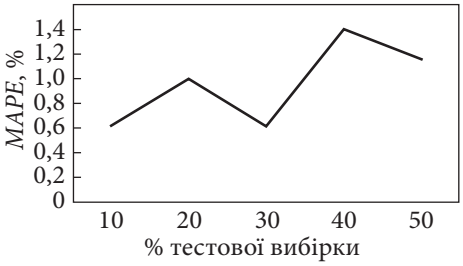


Рис. 10.7. MAPE для різного співвідношення вибірок навчальна / перевірна

1.4. *Дослідження чутливості до співвідношення до інших вхідних змінних.* Для експерименту було використано вхідні дані № 4. Остаточні результати експерименту наведено в табл. 10.8.

Відповідні результати наведено на рис. 10.7.

1.5. *Дослідження чутливості до інших вхідних змінних.* Цей експеримент проводили за частки навчальної вибірки 60 % та 80 %, кількості входів $n = 3, 4, 5, 6$. Результати точності за критерієм *MAPE* наведено в табл. 10.9.

Як випливає з аналізу даних цього експерименту, найкращу точність отримано для шести входів за співвідношення вибірок навчальна / перевірна 60/40 та 80/20 як за критерієм *MSE*, так і за критерієм *MAPE.*

2. **Задача прогнозування цін акцій.** У цій задачі як вихідні змінні було взято ціни закриття акцій *Google* з серпня по грудень 2019 р. Як вхідні змінні було взято *Open* (ціни відкриття), *Low* (мінімальна денна ціна), *High* (максимальна денна ціна) та *Volume* (кількість змін ціни між спостереженнями).

Кількість вхідних змінних $n = 3$. У першому експерименті було досліджено чутливість точності до співвідношення вибірок навчальна / перевірна. Результати наведено в табл. 10.10.

З поданих результатів випливає, що найкращим співвідношенням навчальна / тестова вибірка за обома критеріями є 60/40.

Для оцінки ефективності прогнозування гібридної МГУА-нечіткої мережі було проведено її порівняння з методом МГУА. Відповідні результати наведено в табл. 10.11.

За результатами експериментів, найкращу точність прогнозування було отримано для співвідношення вибірок навчальна / перевірна, що становить 60/40.

Результати порівняння з класичною МГУА підтвердили, що гібридна МГУА-неонечітка мережа має кращу точність, ніж МГУА [10.17, 10.18].

У наступному експерименті було досліджено чутливість точності до кількості входів *n,* яку змінювали від 2 до 5. Результати наведено в табл. 10.12.

У цьому експерименті найкращу точність прогнозування отримано для кількості входів $n = 4$ за співвідношення 60/40.

Таблиця 10.8

Значення *MSE* та *MAPE* за різних співвідношень вибірок навчальна / перевірна

Критерій	Співвідношення вибірок навчальна / перевірна			
	50/50	60/40	70/30	80/20
MSE	3,16	3,81	0,72	1,34
MAPE, %	1,16	1,41	0,63	1,00

Таблиця 10.9

Критерій *MAPE* за відсотком навчальної вибірки за різної кількості входів

Кількість входів	*MAPE*, %, за частки навчальної вибірки у співвідношенні, %	
	60	80
3	1,98	1,31
4	1,41	1,00
5	0,90	1,01
6	0,84	0,94

Таблиця 10.10

Точність прогнозування за різних співвідношень вибірок навчальна / перевірна

Критерій	Співвідношення вибірок навчальна / перевірна			
	50/50	60/40	70/30	80/20
MSE	70,3	45,6	60,5	47,51
MAPE, %	0,56	0,46	0,5	0,49

Таблиця 10.11

Значення *MSE* для неонечіткої мережі МГУА та МГУА за різних співвідношень вибірок навчальна / перевірна

Частка навчальної вибірки у співвідношенні, навчальна / перевірна, %	Значення *MSE* для мережі	
	гібридної неонечіткої	МГУА
50	70,31	72,34
60	45,67	51,46
70	60,51	62,2
80	47,51	52,79

У наступних експериментах досліджували чутливість точності до типу функцій належності (ФН). Під час експериментів було досліджено такі ФН: трикутну, гауссову та дзвоноподібну. Результати наведено в табл. 10.13. Кількість входів становила $n = 4$ за співвідношення 60/40.

Як бачимо, найкращу точність було отримано для гауссової ФН.

Оптимізація структури гібридних мереж. У наступному експерименті досліджували властивість МГУА будувати оптимальну структуру гібридної мережі. Експеримент виконували за таких параметрів: співвідношення вибірок навчальна / перевірна 60/40, кількість входів $n = 4$ та гауссова ФН.

Процес генерації структури та зміну похибки на тестовій вибірці, отримані в результаті роботи МГУА-алгоритму синтезу структури гібридної неонечіткої мережі, зображено на рис. 10.8 [10.17].

Аналізуючи отримані результати, можна спостерігати процес побудови структури, після чого було завершено оптимальну структуру глибокої гібридної нечіткої мережі.

3. **Прогнозування індексу *NASDAQ (National Association of Securities Dealers Automated Quotations).*** У наступних експериментах досліджували ефективність гібридної неонечіткої мережі у прогнозуванні індексу *NASDAQ* [10.17]. Як вихідну змінну було взято ціну закриття індексу на наступний день.

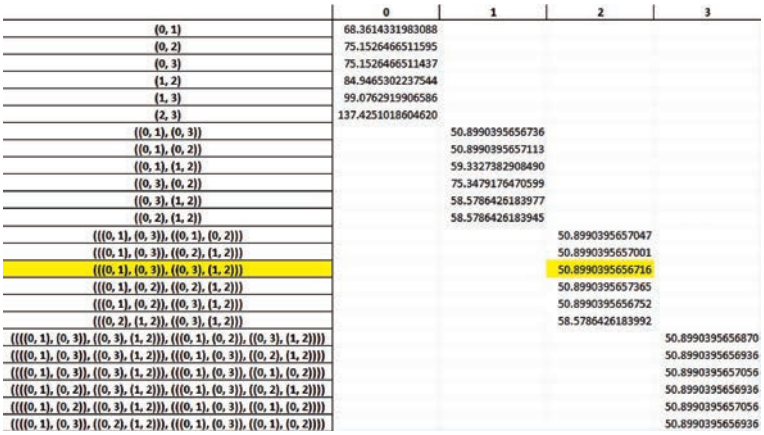
У першому експерименті було досліджено залежність точності від кількості вхідних даних. Результати прогнозування за різної кількості вхідних даних з 8 функціями належності на кожну змінну (параметр h) та за співвідношення вибірок навчальна / перевірна 70/30 наведено в табл. 10.14.

У наступному експерименті досліджували залежність похибки від кількості ФН на змінну (параметр h). Кількість входів $n = 5$, співвідношення вибірок навчальна / перевірна 70/30. Результати наведено на рис. 10.9.

Аналізуючи ці результати, можна дійти висновку, що зі збільшенням кількості ФН *MAPE* спочатку знижується, потім досягає мінімуму, а після цього починає зростати. Це повністю узгоджується з принципом самоорганізації МГУА. Найкраще значення отримано за таких значень параметрів: кількість входів $n = 5$, $h = 5$, кількість шарів — 4, а значення *MAPE* становить 3,91.

У наступному експерименті було здійснено оптимізацію структури гібридної мережі. Оптимальна структура, згенерована МГУА, була такою: шість нейронів на першому шарі, чотири нейрони на другому, два нейрони на третьому і один нейрон на останньому шарі. У структурі використовували всі п'ять входів. Свободу вибору було взято $F = 5$. Згенеровану оптимальну структуру мережі зображено на рис. 10.10.

Для оцінки ефективності прогнозування гібридної мережі її було порівняно з каскадною нечіткою мережею та МГУА з тими самими даними [10.17]. У каскадній нечіткій мережі використовували такі значення параметрів: кількість входів $n = \overline{4,7}$, кількість правил — 9, кількість каскадів — 3.



	0	1	2	3
(0, 1)	68.3614331983088			
(0, 2)	75.1526466511595			
(0, 3)	75.1526466511437			
(1, 2)	84.9465302237544			
(1, 3)	99.0762919906586			
(2, 3)	137.4251018604620			
((0, 1), (0, 3))		50.8990395656736		
((0, 1), (0, 2))		50.8990395657113		
((0, 1), (1, 2))		59.3327382908490		
((0, 3), (0, 2))		75.3479176470599		
((0, 3), (1, 2))		58.5786426183977		
((0, 2), (1, 2))		58.5786426183945		
(((0, 1), (0, 3)), ((0, 1), (0, 2)))			50.8990395657047	
(((0, 1), (0, 3)), ((0, 2), (1, 2)))			50.8990395657001	
(((0, 1), (0, 3)), ((0, 3), (1, 2)))			50.8990395656716	
(((0, 1), (0, 2)), ((0, 2), (1, 2)))			50.8990395657365	
(((0, 1), (0, 2)), ((0, 3), (1, 2)))			50.8990395656752	
(((0, 2), (1, 2)), ((0, 3), (1, 2)))			58.5786426183992	
((((0, 1), (0, 3)), ((0, 3), (1, 2))), (((0, 1), (0, 2)), ((0, 3), (1, 2))))				50.8990395656870
((((0, 1), (0, 3)), ((0, 3), (1, 2))), (((0, 1), (0, 3)), ((0, 2), (1, 2))))				50.8990395656936
((((0, 1), (0, 3)), ((0, 3), (1, 2))), (((0, 1), (0, 3)), ((0, 1), (0, 2))))				50.8990395657056
((((0, 1), (0, 2)), ((0, 3), (1, 2))), (((0, 1), (0, 3)), ((0, 2), (1, 2))))				50.8990395656936
((((0, 1), (0, 2)), ((0, 3), (1, 2))), (((0, 1), (0, 3)), ((0, 1), (0, 2))))				50.8990395657056
((((0, 1), (0, 3)), ((0, 2), (1, 2))), (((0, 1), (0, 3)), ((0, 1), (0, 2))))				50.8990395656936

Рис. 10.8. Значення *MSE* для кожного каскаду гібридної неонечіткої мережі

Таблиця 10.12

Значення *MSE* та *MAPE* залежно від кількості входів

Критерій	Кількість входів			
	2	3	4	5
MSE	58,62	61,71	42,52	45,39
MAPE, %	0,5	0,48	0,38	0,41

Таблиця 10.13

Значення *MSE* та *MAPE* за різних функцій належності

Критерій	Функція належності		
	трикутна	гауссова	дзвоноподібна
MSE	48,13	35,21	61,03
MAPE, %	0,417	0,38	0,504

Таблиця 10.14

Значення *MAPE* прогнозу за різної кількості вхідних даних

Кількість входів	2	3	4	5	6	7	8	9	10
MAPE, %	5,24	4,71	4,3	3,9	4,2	4,7	5,2	5,53	5,85

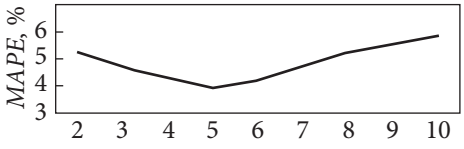


Рис. 10.9. Значення *MAPE* для різної кількості функції належності *h* на кожну змінну

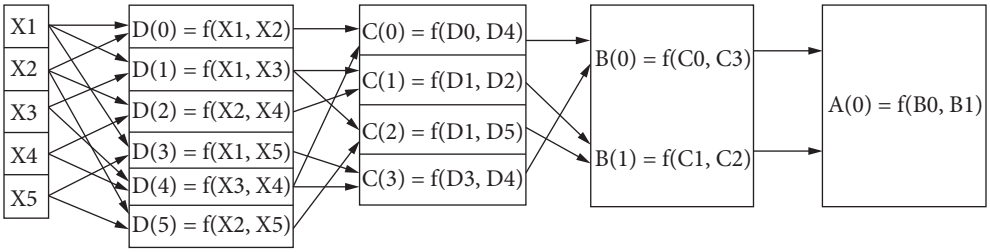


Рис. 10.10. Оптимальна структура гібридної неонечіткої мережі

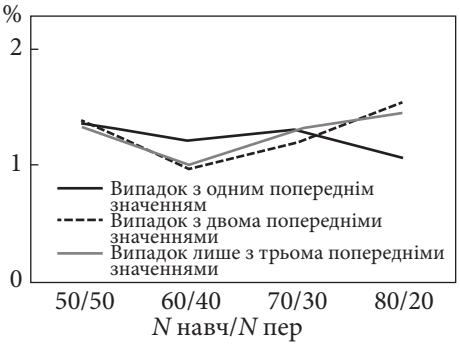


Рис. 10.11. Порівняння значень *MAPE* для результатів прогнозування індексу *NASDAQ*

Таблиця 10.15.

Значення *MAPE* для різних методів прогнозування

Кількість входів	Значення *MAPE*, %, для мереж		
	гібридної неонечіткої МГУА	МГУА	каскадної неонечіткої МГУА
4	4,31	4,19	6,04
5	3,91	4,11	6,09
6	4,36	5,53	8,01
7	4,77	6,26	8,68

Порівняльні результати прогнозування наведено в табл. 10.15, навчальна вибірка становить 70 %.

Аналізуючи результати, можна легко дійти висновку, що найкращу точність має запропонована гібридна нечітка мережа, на другому місці — метод МГУА, а найгіршу — каскадна нечітка мережа.

Інші експерименти, проведені для задачі прогнозування індексу *NASDAQ,* були пов'язані з дослідженням чутливості до співвідношення вибірок навчальна / перевірна. У цій задачі як вихідні змінні було взято значення індексу *NASDAQ* за три місяці. Вхідними змінними слугували акції *Google, Microsoft, Intel* та дані передісторії.

Для того щоб оцінити вплив додавання передісторії, було виконано такі експерименти:

1) до наявних вхідних змінних було додано одне попереднє значення;

2) до наявних вхідних змінних було додано два попередніх значення;

3) як вхідні змінні використовували лише три вихідні значення з передісторії.

Порівняння результатів прогнозування для різних вхідних змінних наведено на рис. 10.11. У результаті експериментів було встановлено, що найвищу точність прогнозування отримано у разі додавання двох попередніх значень у співвідношенні 60/40. Для випадку з трьома попередніми значеннями, а також для випадку без передісторії найвищої точності досягнуто за співвідношення 60/40.

10.6. Неонечіткі системи МГУА з розширеними неонечіткими нейронами

Архітектура розширеного неонечіткого нейрона

У попередньому підрозділі розглянуто гібридну мережу глибокого навчання з вузлами — неонечіткими нейронами, запропоновану Такеші Ямакава та співавторами [10.16].

Вона виконує відображення

$$f_i(x_i) = \Sigma_{j=1}^{h} w_{ji}\, \mu_{ji}(x_i) \qquad (10.14)$$

і реалізує нечіткий висновок: якщо x_i дорівнює x_{ji}, то вихід дорівнює w_{ji}, де x_{ji} — нечітка множина з функцією належності $\mu_{ji}(x_i)$, w_{ji} — синглетна синаптична вага у висновку. Як бачимо, нелінійний синапс фактично актуалізує нечіткий висновок Тоширо — Мінору нульового порядку. Функції належності $\mu_{ji}(x_i)$ в антецеденті можуть бути *B*-сплайнами або трикутними функціями. Вхідний сигнал x_i активує тільки дві сусідні функції належності одночасно, а сума оцінок дорівнює одиниці, забезпечуючи розбиття Руспі-

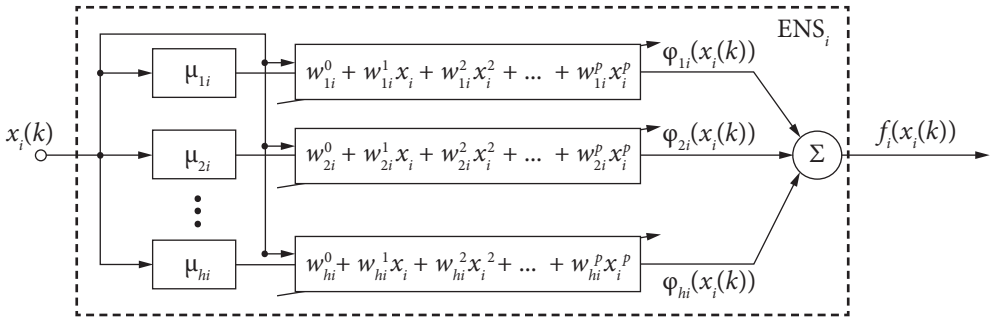


Рис. 10.12. Розширений нелінійний синапс

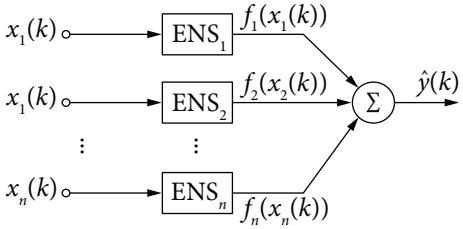


Рис. 10.13. Розширений неонечіткий нейрон

ні (Енріко Руспіні, Enrico Ruspini):

$$\mu_{j-1,i}(x_i) + \mu_{ji}(x_i) = \mu_{ji}(x_i) + \mu_{j+1,i}(x_i) = 1. \qquad (10.15)$$

Результат виведення можна отримати довільним методом дефазифікації. Центроїдний метод надає висновок у найпростішому вигляді:

$$f_i(x_i) = w_{ji}\,\mu_{ji}(x_i) + w_{j+1,i}\,\mu_{j+1,i}(x_i).$$

Поліпшити апроксимаційні властивості такої системи можна завдяки використанню структурної одиниці, названої авторами «розширений нелінійний синапс» (*ENSi*) (рис. 10.12), та синтезованого на її основі «розширеного неонечіткого нейрона» (*ENFN)* [10.19]. *ENFN* містить *ENSi* як елементи замість звичайних нелінійних синапсів *NSi.*

Увівши додаткові змінні

$$\begin{aligned} y_{li}(x_i) &= \mu_{li}(x_i)(w_{li}^0 + w_{li}^1 x_i + w_{li}^2 x_i^2 + ... + w_{li}^p x_i^p); \\ f_i(x_i) &= \textstyle\sum_{l=1}^{h} \mu_{li}(x_i)(w_{li}^0 + w_{li}^1 x_i + w_{li}^2 x_i^2 + ... + w_{li}^p x_i^p) = \\ &= w_{1i}^0\,\mu_{1i}(x_i) + w_{1i}^1 x_i\,\mu_{1i}(x_i) + ... + w_{1i}^p x_i^p\,\mu_{1i}(x_i) + \\ &+ w_{2i}^0\,\mu_{2i}(x_i) + ... + w_{2i}^p x_i^p\,\mu_{2i}(x_i) + ... + w_{hi}^p x_i^p\,\mu_{hi}(x_i), \end{aligned} \qquad (10.16)$$

де

$$w_i = (w_{1i}^0, w_{1i}^1, ..., w_{1i}^p, w_{2i}^0, ..., w_{2i}^p, ..., w_{hi}^p)^T;$$

$$\tilde{\mu}_i(x_i) = (\mu_{1i}(x_i), x_i\,\mu_{1i}(x_i), ..., x_i^p\,\mu_{1i}(x_i),$$
$$\mu_{2i}(x_i), ..., x_i^p\,\mu_{2i}(x_i), ..., x_i^p\,\mu_{hi}(x_i))^T,$$

можемо записати так:

$$f_i(x_i) = w_i^T\,\tilde{\mu}_i(x_i); \tag{10.17}$$

$$\hat{y} = \sum_{i=1}^n f_i(x_i) = \sum_{i=1}^n w_i^T\,\tilde{\mu}(x_i) = \tilde{w}^T\,\tilde{\mu}(x), \tag{10.18}$$

де

$$\tilde{w}^T = (w_1^T, ..., w_i^T, ..., w_n^T)^T;$$

$$\tilde{\mu}(x) = \tilde{\mu}_1^T(x_1), ..., \tilde{\mu}_i^T(x_i), ..., \tilde{\mu}_n^T(x_n))^T.$$

Легко побачити, що *ENFN* містить $(p + 1)hn$ регулювальних синаптичних ваг, а нечіткий вихід, реалізований кожною *ENSi*, має вигляд [10.19]: якщо x_i дорівнює x_{li}, то вихід становить

$$w_{li}^0 + w_{li}^1 x_i + ... + w_{li}^p x_i^p, \quad l = 1, 2, \ldots, h, \tag{10.19}$$

тобто збігається з p-порядком виведення Тоширо — Мінору.

Архітектуру розширеного неонечіткого нейрона показано на рис. 10.13.

Алгоритм навчання неонечітких нейронів

Як цільову функцію для навчання ННН використовують локальну квадратичну функцію похибки:

$$E(k) = \frac{1}{2}(y(k) - \hat{y}(k))^2 = \frac{1}{2}\,e^2(k) = \frac{1}{2}\,(y(k) - \sum_{i=1}^n \sum_{j=1}^h w_{ji}\,\mu_{ji}\,(x_i(k)))^2,$$

де $y(k)$ — зовнішній опорний сигнал. Він мінімізується градієнтним покроковим алгоритмом.

Для збільшення швидкості процесу навчання можна використати однокроковий алгоритм навчання Качмаржа — Відроу — Хоффа (Стефан Качмарж, Бернард Відроу, Марселлус Хофф; Stefan Kaczmarz, Bernard Widrow, Marcian Edward Hoff) [10.19]:

$$w(k) = w(k-1) + \frac{y(k) - w^T(k-1)\mu(x(k))}{||\mu(x(k))||^2}\,\mu(x(k)), \tag{10.20}$$

де

$$\mu(x(k)) = (\mu_{11}(x_1(k)), \ldots, \mu_{h1}(x_1(k)), \ldots, \mu_{h2}\,(x_2(k)), \ldots, \mu_{ji}(x_i(k)), \ldots, \mu_{hn}\,(x_n(k)))^T;$$

$$w(k-1) = (w_{11}(k-1), \ldots, w_{h1}(k-1), \ldots, w_{h2}(k-1), ..., w_{ji}(k-1), ..., w_{hn}(k-1))^T —$$

$(nh \times 1)$-вектори, згенеровані вхідними змінними.

Алгоритм навчання має експоненційно зважену форму:

$$\begin{cases} w(k) = w(k-1) + r^{-1}(k)(y(k) - w^T(k-1)\mu(x(k)))\mu(x(k)); \\ r(k) = \alpha r(k-1) + ||\mu(x(k))||^2, \quad 0 \le \alpha \le 1. \end{cases}$$

Він має властивості фільтрації та відстеження, може бути ефективно використаний у стохастичних і нестаціонарних ситуаціях.

Експериментальні дослідження

Гібридну нечітку мережу МГУА з розширеними нечіткими нейронами було застосовано для розпізнавання емоцій на обличчі людини [10.19]. Задача розпізнавання виразу обличчя людини є складною й багатоетапною. Вона полягає у попередньому обробленні зображення та пошуку ділянки обличчя на зображенні. Після того як ділянку обличчя виділено, можна розпізнати емоцію за набором ознак обличчя. У практиці розпізнавання міміки використовують кілька принципів дескрипторів. Найпоширенішими є адаптивні моделі зовнішності, які використовують описи на основі характерних точок і контурів зображення обличчя. Встановлено, що такі описи передають повну інформацію про емоційний стан людини, навіть якщо він виражений слабо.

За основу вектора ознак пропонується використовувати набір з 35 характерних точок, які можна локалізувати в ділянці обличчя за допомогою контурних детекторів. Нечіткий нейрон має один вихід як розмірність вектора вихідних даних. Для розпізнавання було обрано сім основних емоцій: гнів, відраза, страх, здивування, щастя, сум, нейтральний вираз обличчя. Отже, вихідними значеннями є {1; 2; 3; 4; 5; 6; 7}. Вектор ознак персонажа містить двовимірні координати положення точок ознак (рис. 10.14). Отже, на вхід системи подається вектор $\{x_i\}_{1\times 70}$. Порядок полінома в нелінійній функції належності синапсів було обрано таким, що дорівнює 4, кількість синапсів у неонечіткому нейроні — 5.

Здатність запропонованої архітектури розпізнавати індивідуальні емоції досліджували на фотографіях з двох відкритих баз — *Psychological Image Collection at Stirling (PICS)*, частково з бази *Extended Cohn-Kanade (CK+)* [10.19]. Деякі зображення є у відкритому доступі як об'єкти для розпізнавання.

З цього набору фотографій було відібрано ті, що відрізняються за ступенем вираження емоційного стану людини — від слабко помітного до дуже помітного (див. рис. 10.14).

У цьому завданні особливу увагу було приділено навчальному набору даних невеликого розміру. Для перевірки того, як запропонована архітектура та алгоритм навчання розпізнаватимуть вираз обличчя, використано невеликі набори фотографій (табл. 10.16).

Рис. 10.14. Приклад навчальних зображень та положення точок ознак

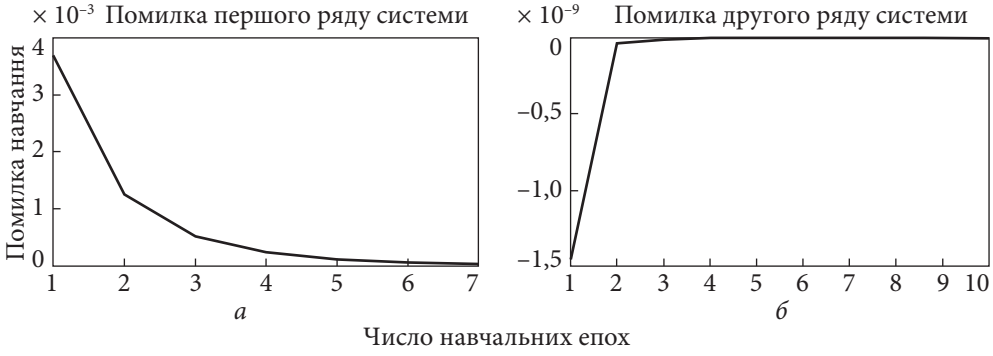


Рис. 10.15. Помилка навчання нейронечіткої системи МГУА на основі розширених неонечітких нейронів: *а* — помилка першого ряду нейронів, *б* — помилка другого ряду нейронів

Таблиця 10.16

Розміри навчальних вибірок із зображеннями емоцій особи

Емоція	Злість	Відраза	Страх	Щастя	Смуток	Здивування	Нейтральна
Розмір набору даних, шт.	49	66	35	45	19	50	80

Таблиця 10.17

Кількість нерозпізнаних емоцій

Первинна емоція	Злість	Відраза	Страх	Щастя	Смуток	Здивування	Нейтральна
Частка нерозпізнаних зображень, %	2,04	0	5,71	4,44	0	0	2,5

Потім було досліджено здатність архітектури навчатися на змішаному наборі, загальний розмір якого становив 344 фотографії. Кількість нерозпізнаних емоцій наведено в табл. 10.17 [10.19].

Нейронечітка система МГУА на основі розширених неонечітких нейронів конструює модель задачі з двох рядів. Зміну похибки навчання показано на рис. 10.15 [10.19].

10.7. Гібридна система обчислювального інтелекту на основі пакетного та групового оброблення даних

Сьогодні глибокі нейронні мережі (ГНМ) широко використовують для розв'язання великого класу задач *Data Mining* і, насамперед, класифікації (розпізнавання образів, оброблення зображень різної природи) та екстраполяції (прогнозування часових рядів, оброблення текстів природною мовою, виявлення несправностей), завдяки своїм універсальним апроксимаційним властивостям, що ґрунтуються головно на теоремі Цибенка (Георгій Цибенко; *Cybenko's Theorem*) [10.22]. Водночас ШНМ є досить громіздкими конструкціями, які містять забагато конфігурованих синаптичних ваг, що, у свою чергу, потребує занадто великого обсягу навчальних даних, які не завжди доступні для вирішення конкретних реальних задач. Крім того, процес навчання ШНМ є досить тривалим, тому під час розв'язування задач *Data Stream Mining*, особливо в разі оброблення нестаціонарних потоків даних, використання цих систем стикається з певними труднощами.

Подолати ці труднощі можна за допомогою ансамблевих методів, при цьому ансамбль може містити різноманітні системи обчислювального інтелекту — від найпростішого типу елементарного перцептрона Розенблатта до найскладніших ШНМ.

Основною проблемою у використанні ансамблевого підходу є об'єднання результатів окремих членів цього ансамблю з метою отримання оптимального початкового результату в сенсі прийнятого критерію навчання. Для цього можуть бути використані процедури бегінгу (*bagging*) [10.22], що мінімізують середньоквадратичну похибку на навчальній вибірці, модифіковані для роботи в режимі онлайн.

Ансамблевий підхід і система пакування можуть бути занадто складними з обчислювальної точки зору. Для спрощення її слід декомпонувати на низку простіших підсистем, при цьому така декомпозиція може бути реалізована досить просто за допомогою методу групового урахування аргументів (*Group Method of Data Handling*, МГУА) для оброблення даних.

Цікаво зазначити, що Й. Шмідхубер вважає, що саме на основі МГУА було побудовано перші системи глибокого навчання. Згодом на основі

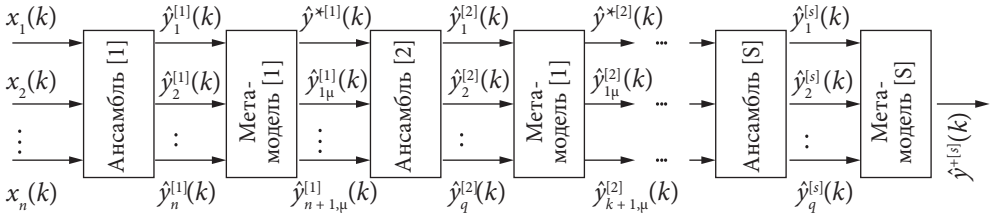


Рис. 10.16. Гібридна система обчислювального інтелекту на основі бегінгу та МГУА

МГУА було запропоновано нейронні мережі та нейронечіткі системи, які продемонстрували свою точність і швидкість у вирішенні низки завдань [10.23]. На нашу думку, доцільно внести до розгляду гібридну систему обчислювального інтелекту (ГСОІ), побудовану на основі ансамблевого підходу і пакетування, яка б збільшила свою архітектуру в процесі навчання на основі ідей МГУА, не потребувала б значних обсягів навчальних вибірок і була б досить простою з обчислювальної точки зору.

Архітектура ГСОІ на основі бегінгу та МГУА. Архітектуру запропонованої системи зображено на рис. 10.16 [10.22].

Архітектура системи містить 2*S* послідовно з'єднаних стеків, при цьому непарні стеки утворені ансамблями паралельно з'єднаних підсистем, які розв'язують одну й ту саму задачу (розпізнавання, прогнозування тощо), а парні є по суті метамоделями, що навчаються, які узагальнюють вихідні сигнали ансамблів і формують оптимальні результати в сенсі прийнятого критерію. Вихідним сигналом першої метамоделі є узагальнений оптимальний сигнал $y^{*[1]}(k)$ та $(n-1)$ вихідних сигналів $\hat{y}_{i\mu}^{[1]}(k)$, $i = 1, 2, \ldots n-1$ «кращих членів ансамблю». За своєю суттю метамоделі функціонують як блоки вибору в традиційних системах МГУА, але не тільки вибирають найкращі результати з попереднього стеку, а й формують оптимальний розв'язок на основі цих результатів.

Далі вихідні сигнали першої метамоделі подаються на входи другого ансамблю, який повністю аналогічний першому. Виходи другого ансамблю $\hat{y}_1^{[2]}(k)$, $\hat{y}_2^{[2]}(k)$, ..., $\hat{y}_q^{[2]}(k)$ надходять на другу метамодель, яка обчислює оптимальний сигнал $y^{*[2]}(k)$ та $(n-1)$ «найближчих» до нього $\hat{y}_{i\mu}^{[2]}(k)$. Останній *S*-й ансамбль аналогічний першим двом, а виходом останньої *S*-ї метамоделі є $y^{*[s]}(k)$, що точно відповідає апріорно встановленим вимогам до якості розв'язання задачі, яка розглядається.

Кожен з ансамблів містить q різних систем обчислювального інтелекту, які розв'язують одну й ту саму задачу. Це можуть бути прості нейронні мережі, такі як одношаровий перцептрон, радіально-базисна нейронна мережа (*RBFN*), нейронна мережа зворотного поширення тощо, які не використовують для навчання процедуру зворотного поширення похибки, нейронечіткі системи типу *ANFIS (Adaptive Neuro-Fuzzy Inference System)*

Ванга — Менделя або Такагі Тоширо та Мінору — Канга, вейвлет-нейросистеми, неонечіткі нейрони та інші, вихідний сигнал яких лінійно залежить від параметрів, які адаптуються, що дає змогу використовувати алгоритми оптимального за швидкістю навчання.

Навчання ГСОІ на основі бегінгу та МГУА

Вхідною інформацією, на основі якої будується система, є навчальна вибірка вхідних сигналів $x(1)$, $x(2)$, ..., $x(k)$, ..., $x(N)$; $x(k) = (x_1(k), x_2(k), \ldots, x_n(k))^T \in R^n$ та відповідних їм скалярних опорних сигналів $y(1)$, ..., $y(\mathrm{k})$, ..., $y(N)$. На основі цих спостережень незалежно один від одного налаштовуються елементи першого ансамблю, на виходах яких формуються q скалярних сигналів $\hat{y}_p^{[2]}(k)$, $p = 1, 2, \ldots, q$, які зручно зображувати у вигляді вектора $\hat{y}^{[1]}(k) = (\hat{y}_1^{[1]}(k), \ldots, \hat{y}_p^{[1]}(k), \ldots, \hat{y}_q^{[1]}(k))^T$. Ці сигнали подаються на входи першої метамоделі, на виходах якої — n послідовностей $\hat{y}^{*[1]}(k)$, $\hat{y}_{1\mu}^{[1]}(k)$, …, $\hat{y}_{i\mu}^{[1]}(k)$, …, $\hat{y}_{n-1,\mu}^{[1]}(k)$, основною з яких є $y^{*[1]}(k)$, а інші є допоміжними. Основним сигналом метамоделі $y^{*[1]}(k)$ є об'єднання виходів усіх членів ансамблю у вигляді

$$y^{*[1]}(k) = \sum_{p=1}^{q} w_p^{*[1]} \hat{y}_p^{[1]}(k) = \hat{y}^{[1]T}(k) w^{*[1]}, \tag{10.21}$$

де $w^{*[1]} = (w^{*[1]}, ..., w_p^{*[1]} ... w_q^{*[1]})^T$ — вектор адаптованих параметрів — синаптичних ваг, на які додатково накладаються обмеження на незміщеність

$$\sum_{p=1}^{q} w_p^{*[1]} = I_q^T w^{*[1]} = 1, \tag{10.22}$$

де I_q — $(q \times 1)$-одиничний вектор.

Задача навчання першої метамоделі зводиться до мінімізації стандартного квадратичного критерію за наявності додаткових обмежень [10.22].

Отже, задачу навчання першої метамоделі можна розв'язати за допомогою стандартного методу штрафних функцій, який в цьому випадку зводиться до мінімізації виразу

$$J(w^{*[1]}, \delta) = (Y(N) - \underline{Y}^{[1]}(N) w^{*[1]})^T (Y(N) - \underline{Y}^{[1]}(N) w^{*[1]}) \; Y(N) + \delta^{-2}(I_q^T w^{*[1]} - 1), \tag{10.23}$$

де $Y(N) = (y(1), \ldots, y(k), \ldots, y(N))^T$ — $(N \times 1)$-вектор; $\underline{Y}^1(N) = (\hat{y}^{[1]}(1), \ldots, \hat{y}^{[1]}(k), \ldots, \hat{y}^{[1]}(N))^T$ — $(N \times q)$-матриця; δ — штрафний коефіцієнт.

Мінімізація (10.23) за $w^{*[1]}$ приводить до такого результату:

$$w^{LS[1]} = (\underline{Y}^{[1]T}(N) \underline{Y}^{[1]}(N))^{+} \underline{Y}^{[1]T}(N) Y(N) = P^{[1]}(N) \underline{Y}^{[1]T}(N) Y(N). \tag{10.24}$$

Той самий результат можна отримати, використовуючи невизначені множники Лагранжа.

Врахуємо похибки метамоделі

$$e^{[1]}(k) = y(k) - y^{*[1]}(k) = y(k) - \hat{y}^{[1]T}(k)w^{*[1]} =$$
$$= w^{*[1]T} I_q\, y(k) - w^{*[1]T}\, \hat{y}^{[1]T}(k) = w^{*[1]T} I_q\, y(k) - \hat{y}^{[1]}(k)) = w^{*[1]T}\, E^{[1]}(k),$$

запишемо функцію Лагранжа у вигляді

$$L(w^{*[1]T}, \lambda) = \sum_{k=1}^{N} w^{*[1]T}\, E^{[1]}(k)\, E^{[1]T}(k)\, w^{*[1]} + \lambda(I_q^T\, w^{*[1]} - 1) =$$
$$= w^{*[1]T}\, R(N)\, w^{*[1]} + \lambda(I_q^T\, w^{*[1]} - 1), \qquad (10.25)$$

де λ — невизначений множник Лагранжа), та розв'яжемо системи рівнянь Куна — Такера:

$$\{\underset{w^{*[1]}}{\underline{V}}\, L(w^{*[1]}, \lambda) = 2R(N)w^{*[1]} + \lambda I_q = \vec{0},\ \frac{\partial L(w^{*[1]}, \lambda)}{\partial \lambda} = I_q^T\, w^{*[1]} - 1 = 0;$$

отримаємо остаточний результат:

$$\{w^{*[1]} = R^{-1}(N)I_q\, (I_q^T\, R^{-1}(N)I_q)^{-1} = -2\, I_q^T\, R^{-1}(N)I_q. \qquad (10.26)$$

Було доведено [10.22], що використання оцінки (10.21) приводить до результатів, які не поступаються точністю найкращому з членів першого ансамблю.

Якщо спостереження з навчальної вибірки обробляють послідовно в режимі онлайн, то доцільно використовувати рекурентний метод найменших квадратів у вигляді

$$P^{[1]}(k+1) = P^{[1]}(k) - \frac{P^{[1]}(k)\, \hat{y}^{[1]}(k+1)\, \hat{y}^{[1]T}(k+1)\, P^{[1]}(k)}{1 + \hat{y}^{[1]T}(k+1)\, P^{[1]}(k)\, \hat{y}^{[1]}(k+1)}; \qquad (10.27)$$

$$w^{LS[1]}(k+1) = w^{LS[1]}(k) + P^{[1]}(k+1)\, y(k+1) - \hat{y}^{[1]T}(k+1)\, w^{LS[1]}(k)\, \hat{y}^{[1]}(k+1);$$

$$w^{*[1]}(k+1) = w^{LS[1]}(k+1) + P^{[1]}(k+1)(I_q^T\, P^{[1]}(k+1)\, I_q)^{-1} \times$$
$$\times (1 - I_q^T\, w^{LS[1]}(k+1)\, I_q),\quad w_p^*(0) = q^{-1}, p = 1, 2, ..., q,$$

або, якщо навчальна вибірка є нестаціонарною, можна використати експоненційно зважений рекурентний метод МНК:

$$P^{[1]}(k+1) = \frac{1}{\alpha}\left(P^{[1]}(k) - \frac{P^{[1]}(k)\, \hat{y}^{[1]}(k+1)\, \hat{y}^{[1]T}(k+1)\, P^{[1]}(k)}{\alpha + \hat{y}^{T}(k+1)\, P^{[1]}(k)\, \hat{y}^{[1]}(k+1)}\right); \qquad (10.28)$$

$$w^{LS[1]}(k+1) = w^{LS[1]}(k) + \frac{P^{[1]}(k)(y(k+1) - \hat{y}^{[1]T}(k+1)\, w^{LS[1]}(k)}{\alpha + \hat{y}^{[']T}(k+1)\, P^{[']}(k)\, \hat{y}^{[']}(k+1)}\, \hat{y}^{[1]}(k+1);$$

$$w^{*[1]}(k+1) = w^{LS[1]}(k+1) + P^{[1]}(k+1)(I_q^T\, P^{[1]}(k+1)I_q)^{-1}\, (1 - I_q^T\, w^{LS[1]}(k+1)I_q,$$
$$w_p^*(0) = q^{-1},\quad p = 1, 2, ..., q,$$

де $0 < \alpha \le 1$ — фактор забування.

Параметрам метамоделі можна надати значення рівнів нечіткої належності до оптимального вихідного сигналу, ввівши додаткові обмеження на невід'ємні значення цих параметрів, тобто крім параметрів, що налаштовуються, $w^{*[1]}$, можна також обчислити рівні цієї належності $\mu_p^{[1]} \geq 0$, $p = 1, 2, \ldots, q$.

Для цього розглянемо розширену функцію Лагранжа

$$L(\mu^{[1]}, \lambda, \rho) = (Y(N) - \underline{Y}^{[1]}(N)\mu^{[1]})^T (Y(N) - \underline{Y}^{[1]}(N)\mu^{[1]}) + \lambda(I_q^T \mu^{[1]} - 1) - \rho^T\mu^{[1]},$$

де ρ — $(q \times 1)$-вектор невід'ємних невизначених множників Лагранжа.

Використовуючи систему рівнянь Куна — Такера

$$\begin{cases} \nabla_{\mu^{[1]}} L(\mu^{[1]}, \lambda, \rho) = \vec{0}, \\ \partial L(\mu^{[1]}, \lambda, \rho)/\partial\lambda = 0, \end{cases} \tag{10.30}$$

нескладно отримати розв'язок у вигляді [10.22]

$$\begin{cases} \mu^{[1]} = p^{[1]}(N)(\hat{y}^T(N)Y(N) - 0{,}5\lambda I_q + 0{,}5\rho), \\ \lambda = \dfrac{I_q^T p^{[1]}(N)\hat{y}^T(N)Y(N) - 1 + 0{,}5I_q^T p^{[1]}(N)\rho}{0{,}5I_q^T p^{[1]}(N)I_q}. \end{cases} \tag{10.31}$$

Для знаходження вектора невід'ємних множників Лагранжа доцільно застосувати процедуру Ерроу — Гурвіца — Узави:

$$\begin{cases} \mu^{[1]}(k+1) = w^{LS[1]}(k+1) - P^{[1]}(k+1) \times \\ \times \dfrac{I_q^T w^{LS[1]}(k+1) - 1 + 0{,}5I_q^T p^{[1]}(k+1)\rho(k)}{I_q^T P^{[1]}(k+1)I_q} + 0{,}5P^{[1]}(k+1)\rho(k), \\ \rho(k+1) = P_{r+}(\rho(k) - \eta_\rho(k)\,\mu^{[1]}(k+1)) \end{cases} \tag{10.32}$$

де $P_{r+}(\cdot)$ — проєктор на додатний ортант; $\rho_{(k)}$ — параметр швидкості навчання.

Перший вираз (10.32) після нескладних перетворень можна подати у більш компактному вигляді

$$\mu^{[1]}(k+1) = w^{*[1]}(k+1) - P^{[1]}(k+1)\frac{0{,}5I_q^T p^{[1]}(k+1)\rho(k)}{I_q^T P^{[]}(k+1)I_q} +$$

$$+ 0{,}5P^{[1]}(k+1)\rho(\mathrm{k}) = w^{*[1]}(k+1) + 0{,}5\left(I_{qq} - \frac{P^{[1]}(k+1)\rho(k)I_q I_q^T}{I_q^T P^{[1]}(k+1)I_q}\right)P^{[1]}(k+1)\rho(\mathrm{k}),$$

де замість оцінок за методом найменших квадратів $w^{LS[1]}(k+1)$ використовуються параметри метамоделі $w^{*[1]}(k+1)$, що спрощує процес його налаштування.

У результаті навчання першої метамоделі на її виході формується оптимальний сигнал $y^{*[1]}(k)$, а також q сигналів $\hat{y}^{[1]}_{p,\,\mu}(k)$, з яких вибираємо $n-1$ (якщо $q \geq n$) з найвищими рівнями нечіткої належності $\mu^{[1]}_p$, які в подальшому у вигляді $(n \times 1)$-векторів подаються на вхід другого ансамблю, виходи якого йдуть на входи другої метамоделі, і так далі. Процес збільшення кількості ансамблів та метамоделей триває доти, доки не буде досягнуто необхідної точності останньої метамоделі з виходом $y^{*[s]}(k)$ або не почне зростати значення критерію, що мінімізується для моделі бегінгу, тобто

$$\varepsilon^2(y^{*[s+1]}(k)) \geq \varepsilon^2(y^{*[s]}(k)).$$

Експериментальні дослідження

Експериментальні дослідження бегінгу на основі МГУА було проведено на задачах коротко- та середньострокового прогнозування промислового індексу Доу-Джонса [10.22]. Динаміку *DJIA (Dow Jones Industrial Average)* наведено на рис. 10.17 (дані взято з 05.07.22 по 03.07.23).

Розраховано кореляційну функцію динаміки індексу Доу-Джонса (корелограму), зображену на рис. 10.18. Проаналізувавши корелограму, можна побачити значну кореляцію між значеннями індексу Доу-Джонса.

Найпершою в ансамблі бегінгу застосовують модель *ARIMA (AutoRegressive Integrated Moving Average)*. Це авторегресійна інтегрована модель ковзного середнього, яка використовується для прогнозування часових рядів, зокрема фондових ринків. Графік її автокореляційної функції (АКФ) подано на рис. 10.19.

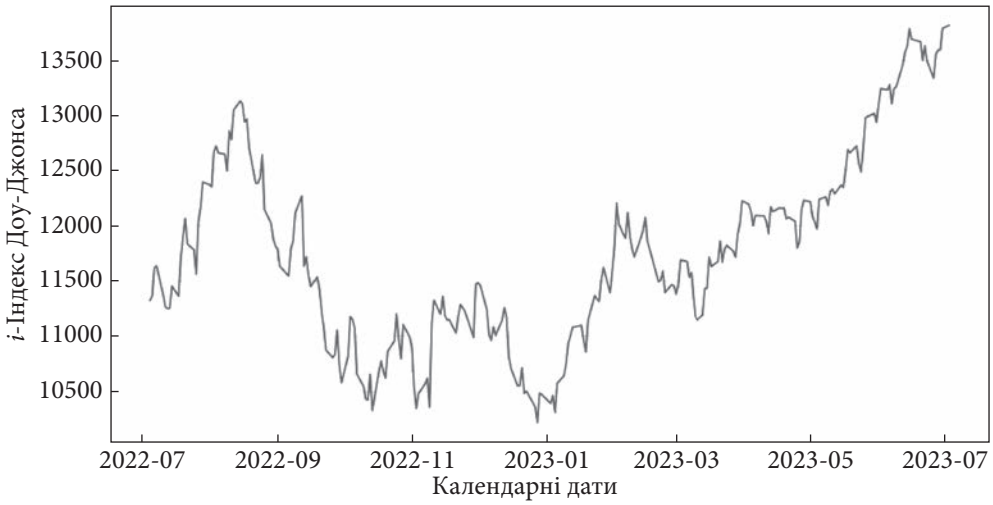


Рис. 10.17. Динаміка *DJIA*

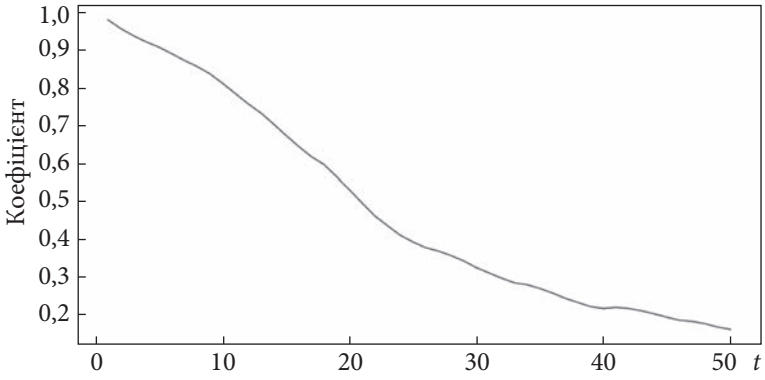


Рис. 10.18. Корелограма

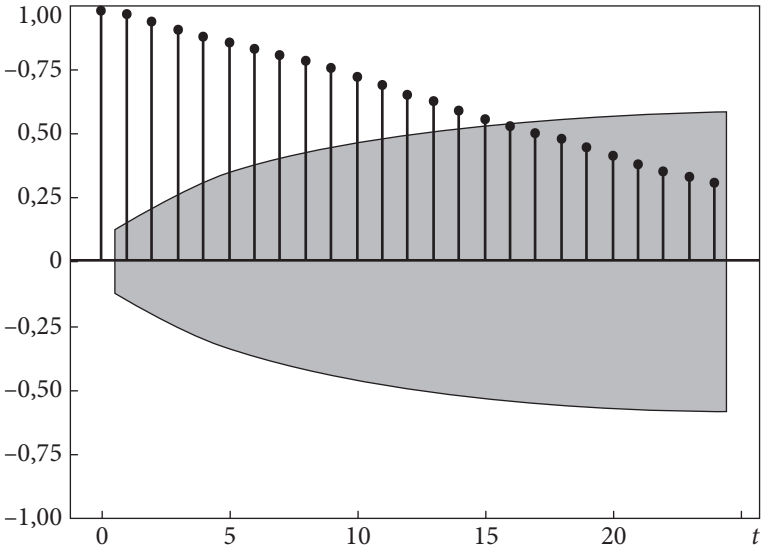


Рис. 10.19. Графік АКФ

За допомогою диференціації процес *DJIA* було перетворено на стаціонарний.

1. Проведено тест Дікі — Фуллера (статистичний тест, що використовується у фондових ринках для визначення стаціонарності часових рядів, зокрема цін акцій, фондових індексів, валютних курсів та інших фінансових показників). *P*-значення: $1{,}3051439086544856 \cdot 10^{-28} \ll 0{,}05$.

2. Експериментальні дослідження проводили на різних інтервалах прогнозування: 1, 3, 5, 7 днів (короткострокове прогнозування) та 10, 20 днів (середньострокове прогнозування).

3. Графік прогнозу з інтервалом в один день, виконаного за допомогою *ARIMA*, наведено на рис. 10.20.

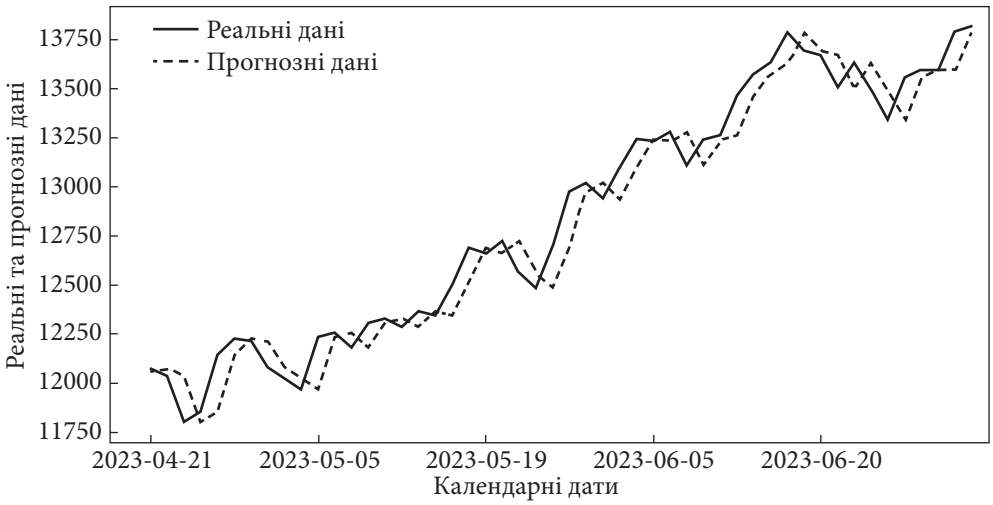


Рис. 10.20. Графік прогнозування за *ARIMA* з інтервалом в 1 день

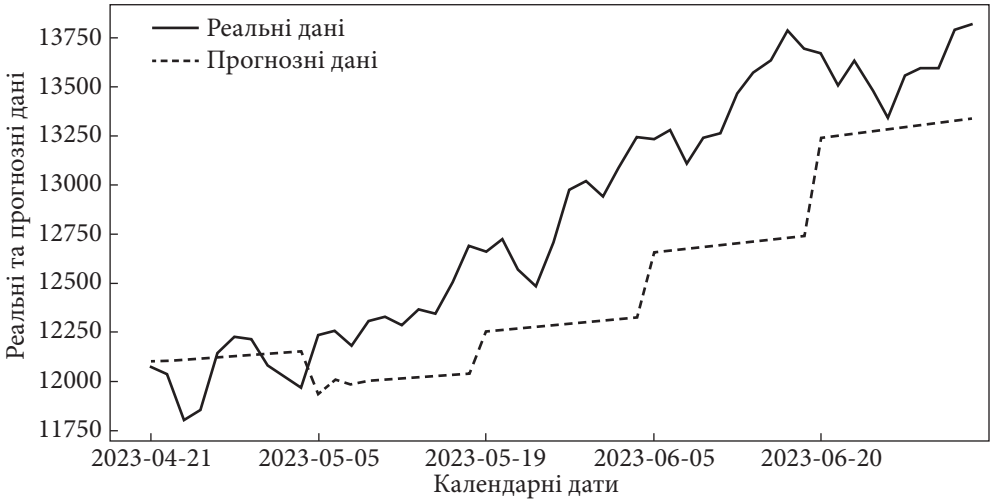


Рис. 10.21. Графік прогнозування за *ARIMA* з інтервалом у 7 днів

Графік прогнозування за *ARIMA* для інтервалу 7 днів зображено на рис. 10.21.

Наступною моделлю, що використовується в ансамблі, є гібридна нечітка мережа МГУА. Її було оптимізовано за параметрами. Після цього було проведено експерименти з різними інтервалами прогнозування: 1, 3, 5, 7, 10, 20 днів. Деякі з результатів наведено нижче. Графік прогнозу на 3 дні зображено на рис. 10.22, на 10 днів — на рис. 10.23.

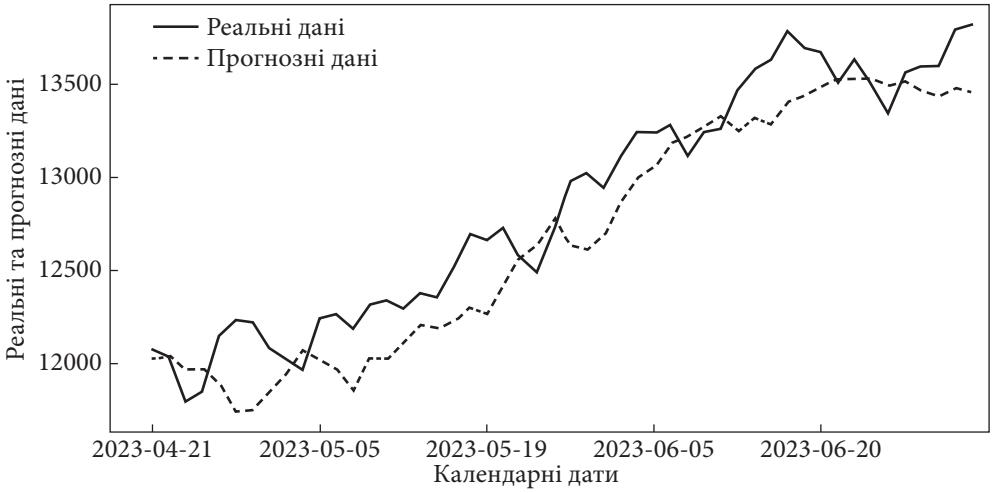


Рис. 10.22. Графік прогнозування за неонечіткою мережею МГУА з інтервалом у 3 дні

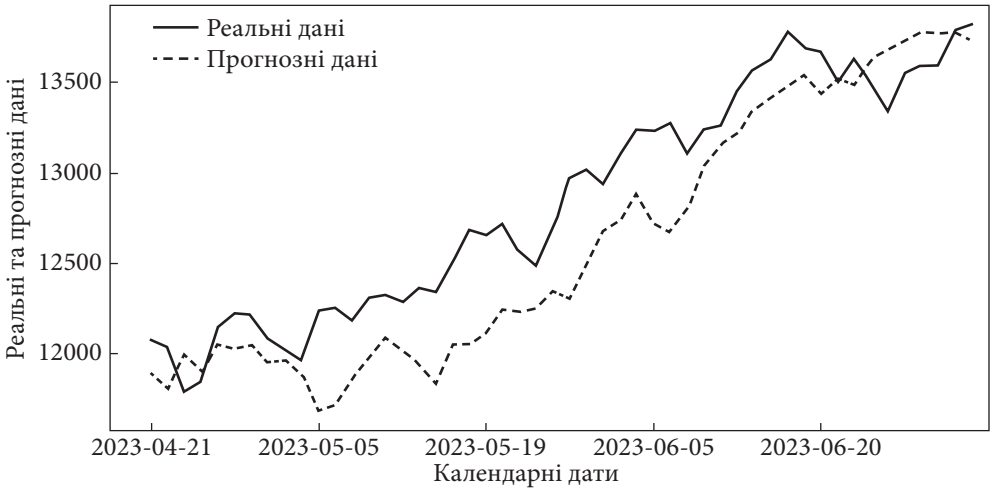


Рис. 10.23. Графік прогнозування за неонечіткою мережею МГУА з інтервалом у 10 днів

Таблиця 10.18

Середні значення *MSE* за різних інтервалів прогнозування

Інтервал, дні	*ARIMA*	Неонечітка МГУА	*HSCI*-МГУА бегінг
1	17422,752	27420,394	17382,425
3	61945,0	59202,99	49332,635
5	66993,011	78330,284	58153,202
10	235427,989	108358,616	104324,0
20	696291,974	241508,146	241508,146

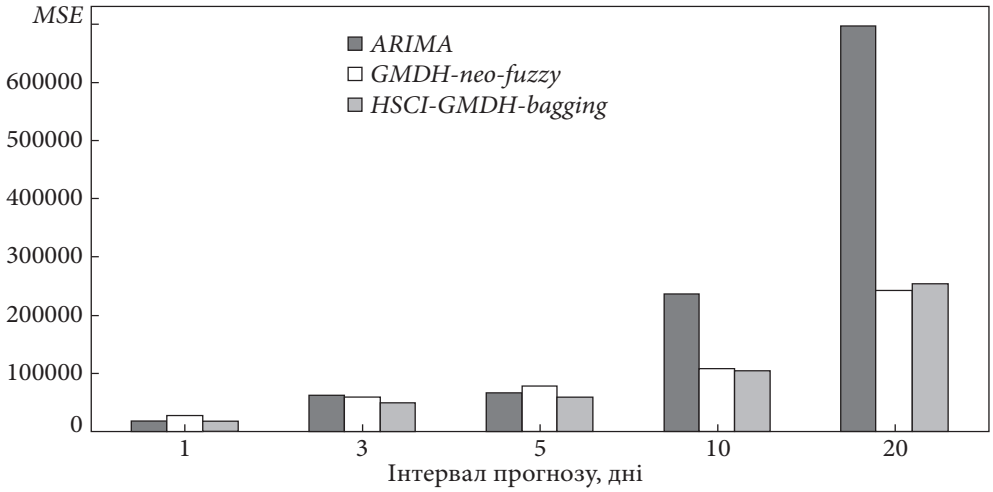


Рис. 10.24. Середні значення *MSE* для різних інтервалів прогнозування

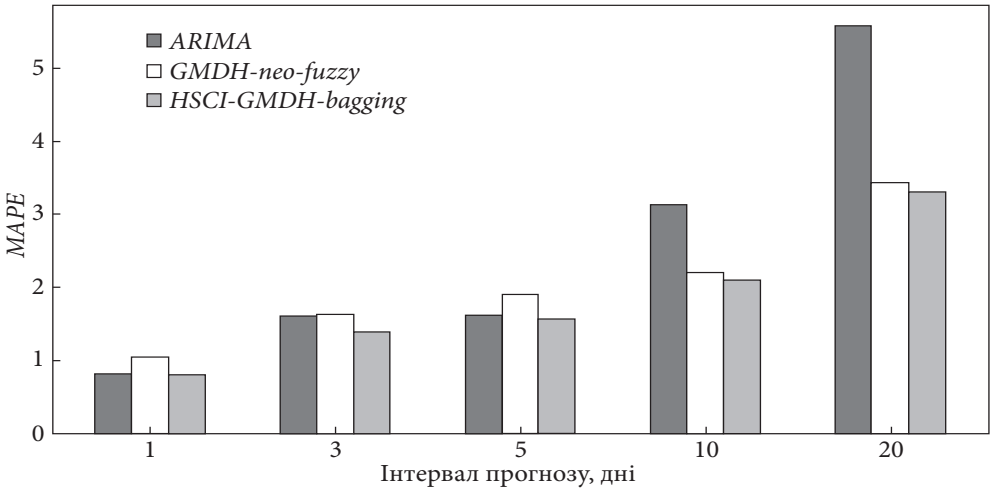


Рис. 10.25. Середні значення *MAPE* для різних інтервалів прогнозування

Таблиця 10.19

Середні значення *MAPE*, %, за різних інтервалів прогнозування

Інтервал, дні	*ARIMA*	Неонечітка МГУА	*HSCI*-МГУА бегінг
1	0,83	1,049	0,828
3	1,612	1,63	1,398
5	1,616	1,903	1,577
10	3,134	2,208	2,108
20	5,579	3,433	3,308

Після цього було реалізовано алгоритм бегінгу на основі МГУА і проведено експерименти з коротко- та середньострокового прогнозування. Значення критеріїв *MSE* та *MAPE* наведено в табл. 10.18 та 10.19 [10.22].

Середні значення *MSE* за різних інтервалів прогнозування зображено на рис. 10.24, *MAPE* — на рис. 10.25

Аналізуючи наведені результати, можна дійти висновку, що процедура бегінгу на основі МГУА має найкращі результати порівняно з окремими моделями в ансамблі, друге місце посідає гібридна МГУА-нечітка мережа для всіх інтервалів, за винятком 1 та 3 днів. А для короткострокового прогнозування на інтервалі 1 день *ARIMA* виявляється кращою за гібридну МГУА-нечітку мережу.

Загалом отримані результати добре узгоджуються з теоретичними положеннями щодо властивостей процедури пакетування, а запропонований бегінг на основі МГУА в системі *HSCI (Hybrid Systems of Computational Intelligence*, гібридні системи комп'ютерного інтелекту) виявився дуже ефективною процедурою, яка потребує мінімальних обчислень завдяки використанню МГУА.

Висновки

Розглянуто і досліджено гібридну нейронечітку мережу глибокого навчання на основі самоорганізації. Як вузли цієї мережі використовують елементарні нечіткі нейронні мережі Ванга — Менделя з двома входами.

Розроблено алгоритм синтезу структури гібридної нейронної мережі на основі методу самоорганізації МГУА онлайн. Перевагою цього алгоритму є те, що синтез структури мережі відбувається одночасно з її навчанням, шар за шаром, до виконання умови зупинки.

Проведено експериментальне дослідження запропонованої гібридної МГУА-нейронечіткої мережі глибокого навчання в задачах прогнозування цін на акції корпорації *Microsoft,* у результаті якого знайдено оптимальні значення параметрів мережі: кількість входів, кількість фазифікаторів, а також оптимальну структуру мережі для цієї задачі. Також було проведено порівняльні експерименти з прогнозування методом МГУА та повної каскадної неонечіткої мережі, які показали, що гібридна нейронна мережа має вищу точність прогнозування цін на акції та фондові індекси порівняно з альтернативними методами прогнозування.

У результаті проведених експериментів було зроблено такі висновки.

1. Варіація кількості входів у гібридній нейронечіткій мережі впливає на якість прогнозу: зі збільшенням кількості входів критерій *MAPE* спочатку падає, досягає мінімуму, а потім починає зростати.

2. Застосування запропонованого методу синтезу структури та навчання гібридної мережі дає змогу скоротити час навчання і підвищити точність прогнозування порівняно з каскадною нейронною мережею.

3. У цілому, запропонований метод навчання гібридної мережі на основі МГУА має такі переваги: немає проблем з алгоритмами глибокого навчання — зникнення або вибух градієнта через алгоритм навчання, який працює послідовно.

Розроблено і досліджено гібридну мережу МГУА з розширеними нечіткими нейронами як вузлами. Архітектура системи дає можливість будувати структуру моделі в реальному часі завдяки запропонованому налаштуванню синаптичних ваг нечітких вузлів-нейронів. Основною властивістю запропонованої архітектури та алгоритму навчання є можливість роботи з невеликими навчальними вибірками.

Вперше запропоновано архітектуру та алгоритми навчання гібридної системи обчислювального інтелекту на основі МГУА та бегінгу. Система складається з послідовності стеків, причому непарні стеки, по суті, є ансамблями, утвореними паралельно з'єднаними різними підсистемами, які розв'язують одну й ту саму задачу, а парні — метамоделями, які реалізують процедуру бегінгу та обчислюють рівні нечіткої належності кожного члена ансамблю до оптимального результату.

Процес збільшення кількості стеків ґрунтується на принципах МГУА доти, доки не буде досягнуто бажаної точності кінцевих результатів або не почне зростати критерій *MAPE* (чи *MSE*) на перевірній вибірці. Запропонована система не потребує великих обсягів навчальних вибірок, забезпечує роботу в режимі онлайн та автоматично визначає кількість шарів — стеків у процесі навчання.

СПИСОК ЛІТЕРАТУРИ

10.1. Ivakhnenko A.G., Ivakhnenko G.A., Mueller J.A. Self-organization of the neural networks with active neurons. *Pattern Recognition and Image Analysis*. 1994. 4 (2). P. 177—188.

10.2. Madala H.R., Ivakhnenko A.G. Inductive Learning Algorithms for Complex System Modeling. CRC Press, 1994. 368 p. URL: https://gmdh.net/articles/theory/GMDHbook.pdf

10.3. Ivakhnenko A.G., Wuensch D., Ivakhnenko G.A. Inductive sorting-out GMDH algorithms with polynomial complexity for active neurons of neural networks. *Neural Networks*. 1999. 2. P. 1169—1173.

10.4. Ivakhnenko A.G. Polynomial theory of complex systems. *Kybernetika*. 1971. 7(6). P. 396—408.

10.5. Ohtani T. Automatic variable selection in RBF network and its application to neurofuzzy GMDH. *Proc. Fourth Int. Conf. on Knowledge-Based Intelligent Engineering Systems and Allied Technologies.* 2000. Vol. 2. P. 840—843.

10.6. Bodyanskiy Ye.V., Vynokurova O.A., Dolotov A.I. Self-learning cascade spiking neural network for fuzzy clustering based on Group Method of Data Handling. *J. of Automation and Information Sciences*. 2013. 45 (3). P. 23—33.

10.7. Bodyanskiy Ye., Vynokurova O., Dolotov A., Kharchenko O. Wavelet-neuro-fuzzy network structure optimization using GMDH for the solving forecasting tasks. *Proc. 4th Int. Conf. on Inductive Modelling ICIM*. Kyiv, 2013. P. 61—67.

10.8. Bodyanskiy Ye., Teslenko N., Grimm P. Hybrid evolving neural network using kernel activation functions. *Proc. 17th Zittau East-West Fuzzy Colloquium.* Zittau/Goerlitz, HS. 2010. P. 39—46.

10.9. Bodyanskiy Ye., Vynokurova O., Pliss I. Hybrid GMDH-neural network of computational intelligence. *Proc. 3rd Int. Workshop on Inductive Modeling.* Krynica, Poland, 2009. P. 100—107.

10.10. Bodyanskiy Ye., Vynokurova O., Teslenko N. Cascade GMDH-wavelet-neuro-fuzzy network. *Proc. 4th Int. Workshop on Inductive Modeling «IWIM 2011».* Kyiv, 2011. P. 22—30.

10.11. Bodyanskiy Ye., Zaychenko Yu., Pavlikovskaya E., Samarina M., Viktorov Ye. The neo-fuzzy neural network structure optimization using the GMDH for the solving forecasting and classification problems. *Proc. Int. Workshop on Inductive Modeling.* Krynica, Poland, 2009. P. 77—89.

10.12. Wang L.-X., Mendel J.M. Fuzzy basis functions, universal approximation, and orthogonal least-squares learning. *IEEE Trans. on Neural Networks*. 1992. 3 (5). P. 807—814.

10.13. Wang L.-X. Adaptive Fuzzy Systems and Control. Design and Statistical Analysis. Upper Saddle River: Prentice Hall, 1994.

10.14. Zaychenko Yu., Bodyanskiy Ye., Tyshchenko O., Boiko O., Hamidov G. Hybrid GMDH-neuro-fuzzy system and its training scheme. *Int. J. Information theories and Applications*. 2018. Vol. 24 (2). P. 156—172.

10.15. Zaychenko Yu., Bodyanskiy Ye., Boiko O., Hamidov G. Evolving Hybrid GMDH-Neuro-Fuzzy Network and Its Application. *Int. conf. IEEE-SAIC, 2018.* Kyiv, IASA, 8—11 October, 2018.

10.16. Yamakawa T., Uchino E., Miki T., Kusanagi H. A neo-fuzzy neuron and its applications to system identification and prediction of the system behavior. *Proc. 2nd Int. Conf. Fuzzy Logic and Neural Networks «LIZUKA-92»*. Lizuka, 1992. P. 477—483.

10.17. Bodyanskiy Ye., Zaychenko Yu., Hamidov G. Hybrid Deep Learning Networks Based on Self-Organization and their Applications. Cambridge Scholars Publishing, 2024. 125 p.

10.18. Bodyanskiy Ye., Zaychenko Yu., Boiko O., Hamidov G., Zelikman A. Structure Optimization and Investigations of Hybrid GMDH-Neo-fuzzy Neural Networks in Forecasting Problems. System Analysis & Intelligent Computing / Ed. M. Zgurovsky, N. Pankratova. *Book Studfies in Computational Intelligence, SCI.* Vol. 1022. Springer, 2022. P. 209—228.

10.19. Bodyanskiy Ye., Zaychenko Yu., Hamidov G., Kuleshova N. Multilayer GMDH-neuro-fuzzy network based on extended neo-fuzzy neurons and its application in online facial expression recognition. *Системні дослідження та інформаційні технології.* 2020. No. 3. P. 67—78.

10.20. Zaychenko Yu., Kuzmenko O., Zaichenko H. Investigation of Artificial Intelligence Methods in the Short-Term and Middle-Term Forecasting in Financial Sphere. *Proceedings of the Int. conf. Information Technology and Implementation (IT&I-2022).* November 30 — December 02, 2022, Kyiv, Ukraine.

10.21. Gooodfellow I., Bengio Yo., Courville A. Deep Learning. 2nd ed. MIT Press, 2016. 772 p.

10.22. Bodyanskiy Ye., Kuzmenko O., Zaichenko H., Zaychenko Yu. Hybrid system of computational intelligence based on bagging and group method of data handling. *Systemresearch and Information Technologies*. 2024. No. 1. https://doi.org/10.20535/SRIT.2308-8893.2024.1.06

10.23. Schmidhuber J. Deep learning in neural networks: An overview. *Neural Networks*. 2015. 61. P. 85—117.

10.24. Zgurovsky M., Zaychenko Yu. Fundamentals of computational intelligence. System approach. Springer, 2016. 275 p.

10.25. Zgurovsky M., Zaychenko Yu. Big Data: Conceptual Analysis and Applications. Springer Nature Switzerland AG, 2019. 306 p.

10.26. Hubel D.H., Wiesel T.N. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The J. of physiology*. 1962. Vol. 160 (1). P. 106—154.

10.27. Fukushima K. Neocognitron: A self-organizing neural network model for amechanism of pattern recognition unaffected by shift inposition. *Biological cybernetics*. 1980. Vol. 36 (4). P. 193—202.

10.28. Ivakhnenko A.G. Self-organization of neuronet with active neurons for effects of nuclear test explosions forecasting. *System Analysis Modeling Simulation*. 1995. 20. P. 107—116.

10.29. Jang R.J.-S. ANFIS: Adaptive-network-based fuzzy inference systems. *IEEE Trans. on Systems, Man, and Cybernetics*. 1993. 23. P. 665—685.

10.30. Jang R. J.-S., Sun C.-T. Mizutani E. Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence. Upper Saddle River: Prentice Hall, 1997.

10.31. Lughofer E. Evolving Fuzzy Systems — Methodologies, Advanced Concepts and Applications. Berlin-Heidelberg: Springer-Verlag, 2011.

10.32. Ohtani T., Ichihashi H., Miyoshi T., Nagasaka K., Kanaumi Y. Structural learning of neurofuzzy GMDH with Minkowski norm. *Proc. 1998 Second Int. Conf. on Knowledge-Based Intelligent Electronic Systems.* 1998. Vol. 2. P. 100—107.

10.33. Osowski S. Sieci neuronowe do przetwarzania informacji. Warszawa: Oficyna Wydawnicza Politechniki Warszawskiej, 2006.

10.34. Pham D.T., Liu X. Neural Networks for Identification, Prediction and Control. London: Springer-Verlag, 1995.

10.35. Sugeno M., Kang G.T. Structure identification of fuzzy model. *Fuzzy Sets and Systems*. 1998. 28. P. 15—33.

10.36. Zaychenko Yu. The fuzzy Group Method of Data Handling and its application for economical processes forecasting *Scientific Inquiry*. 2006. 7 (1). P. 83—96.

10.37. Zaychenko Yu., Hamidov G. Hybrid Fuzzy CNN Network in the Problem of Medical Images Classification and Diagnostics. *Proceedings of 15th Int. Conf. on Natural Computation. Fuzzy Systems and Knowledge Discovery FSKD — 2019.* & *5th Int. Conf. on Harmony Search, Soft Computing and Applications.* P. 1—6.

10.38. Zaychenko Yu., Zaychenko H., Hamidov G. Structure Optimization of New Generation Computer Networks. *The 2017 10th Int. Congr. on Image and Signal Processing, BioMedical Engineering and Informatics.* China, Shanghai, October 13—16, 2017.

10.39. Zaychenko Yu., Gasanov A., Hamidov G. New Generation Networks Performance Analysis and Optimization. *IEEE 10th Int.* Baku, Azerbaijan, October 2016. P. 588—594.

10.40. George S. Atsalakis, Kimon P. Valavanis. Forecasting stock market short-term trends using a neuro-fuzzy based methodology. *Expert Systems with Applications*. 2009. Vol. 36. P. 10696—10707.

10.41. Zaychenko Yu., Hamidov G., Varga I. Medical images of breast tumors diagnostics with application of hybrid CNN — HHM network. *Системні дослідження та інформаційні технології.* 2018. No. 4. С. 37—47.

10.42. Zaychenko Yu.P., Hamidov G. Inductive Modeling Method GMDH in the Problems of Data Mining. *Int. J. Information Theory and Applications.* 2017. Vol. 22 (2). P. 156—176.

10.43. Bodyanskiy Ye., Kulishova N., Zaychenko Yu., Hamidov G. Spline-Orthogonal Extended Neo-Fuzzy Neuron. *Int. conf. CISP-BMEI*, 2019.

10.44. Çınar A., Yildirim M. Detection of tumors on brain MRI images using the hybrid convolutional neural network architecture. *Medical Hypotheses*. 2020. Vol. 139. 109684. https://doi.org/10.1016/j.mehy.2020.109684

10.45. Zaychenko Yu., Hamidov G. Hybrid Convolutional Neuro-Fuzzy Networks for Diagnostics of MRI-Images of Brain Tumors. Mathematical Modeling and Simulation of Systems / Eds. S. Shkarlet, A. Morozov, A. Palagin. Springer, 2020. P. 147—155.

10.46. Zaychenko Yu., Zaychenko H., Hamidov G. Hybrid GMDH Deep Learning Networks. Analysis, Optimization and Applications in Forecasting at Financial Sphere. Системні дослідження та інформаційні технології. 2022. No. 1. P. 73—86. https://doi.org/10.20535/SRIT.2308-8893.2022.1.06

10.47. Paterson J., Gibson A. Deep Learning: A Practitioner's Approach. 1st ed. Kindle Edition, 2017. 538 p.

РОЗДІЛ 11

ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ

11.1. Поняття згортки

11.2. Загальні особливості згорткових нейронних мереж, шари ЗНМ та їх функції

11.3. Архітектури згорткових нейронних мереж

Висновки

Список літератури

Згорткові нейронні мережі є одним із видів нейронних мереж прямого поширення і використовуються для оброблення даних з мережевою топологією. Прикладами таких даних є часові ряди, які можна розглядати як одновимірну сітку, що робить вибірки через регулярні проміжки часу, а також зображення, які можна розглядати як двовимірну сітку пікселів. Згорткові нейронні мережі виявилися неймовірно ефективними у використанні на практиці. Своєю назвою вони завдячують використанню математичної операції згортки.

Згортка — це особливий вид лінійної операції. Згорткові нейронні мережі — прості нейронні мережі, які використовують згортку замість загальної операції множення на матрицю принаймні в одному зі своїх шарів [11.1].

11.1. Поняття згортки

Загалом згортка — це операція над функціями двох аргументів з дійсним значенням, або математична операція, яка полягає в проходженні сигналу через лінійну та інваріантну в часі систему або фільтр [11.2]. Нехай ми маємо сигнал $x(t)$, що проходить через систему з імпульсною характеристикою $w(t)$, тоді на виході отримаємо згортку $s(t)$ сигналу $x(t)$ з $w(t)$. Згортка — це просто інтеграл від множення двох функцій:

$$s(t) = \int x(a)w(t - a)da. \tag{11.1}$$

Цей варіант запису називається оператором згортки і позначається так:

$$s(t) = (x * w)(t).$$

У термінології згорткових мереж функція x називається входом, а функція w — ядром. Результат операції називається картою ознак.

У дискретній формі операцію згортки можна записати так:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{+\infty} x(a)w(t - a).$$

У програмах машинного навчання вхідними даними зазвичай є багатовимірний набір даних, а ядром — багатовимірний масив параметрів, який

адаптується алгоритмом навчання. Ці масиви називаються тензорами. Оскільки кожен вхідний елемент і ядро мають зберігатися окремо в явному вигляді, зазвичай вважається, що ці функції дорівнюють нулю скрізь, крім кінцевого набору точок, для яких ми зберігаємо значення. На практиці це означає, що суму від «мінус» до «плюс» нескінченності можна замінити сумою скінченної кількості елементів масиву.

Згортку можна використовувати одночасно в напрямках декількох осей. Для двовимірних зображень згортка має бути двовимірною. Для деякого зображення I і ядра K матимемо:

$$G(i, j) = (I * K)(i, j) = \sum_m \sum_n I(m, n)\, K(i - m, j - n).$$

Оскільки операція згортки є комутативною, то наступну формулу можна записати так:

$$G(i, j) = (I * K)(i, j) = \sum_m \sum_n I(i - m, j - n)\, K(m, n).$$

11.2. Загальні особливості згорткових нейронних мереж, шари ЗНМ та їх функції

Згортка має три важливі ідеї, які допомагають покращити систему машинного навчання: розріджені зв'язки (*sparse connections*), спільний доступ до параметрів, еквівалентне подання [11.1]. Опишемо ці ідеї далі. Приклад операції згортки продемонстровано на рис. 11.1.

Розріджені з'єднання. Розрідженість зв'язків досягається завдяки тому, що розмір ядра менший за розмір вхідних даних. Наприклад, зображення може містити тисячі або мільйони пікселів, але невеликі значущі ознаки, як-от межі або кути, можуть бути виявлені за допомогою ядра з десятків пікселів. Отже, потрібно зберігати менше параметрів, що зменшує необхідний обсяг пам'яті моделі і підвищує її ефективність.

Спільне використання параметрів. Під спільним використанням параметрів мається на увазі, що один і той самий параметр моделі використовується більш ніж в одній функції. У традиційній нейронній мережі кожний параметр використовується тільки один раз при множенні на відповідний вхід. У згортковій нейронній мережі кожний елемент ядра використовується для кожного пікселя вхідного зображення, за винятком крайніх пікселів. Спільне використання параметрів означає, що замість того, щоб навчати окремий набір параметрів для кожного вхідного пікселя, потрібно навчити лише один набір для ядра, а потім можна застосовувати його до кожного вхідного пікселя. Це значно зменшує обсяг пам'яті, необхідний для моделі.

Еквівалентне зображення. Згорткові нейронні мережі мають властивість еквівалентності відносно паралельного переносу. Функція є еквівалентною, якщо вихід змінюється однаково у разі зміни входу. Функ-

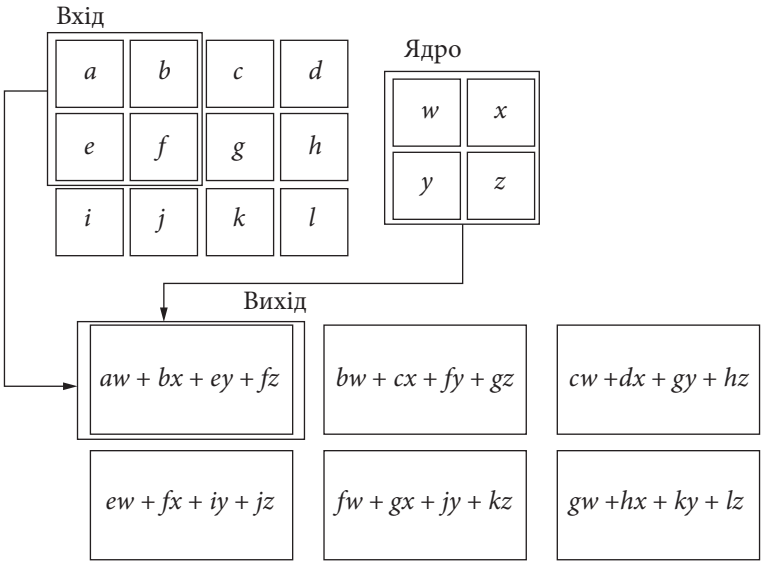


Рис. 11.1. Приклад операції згортки

ція $f(x)$ еквівалентна функції g, якщо $f(g(x)) = g(f(x))$. У випадку згортки, якщо g є паралельним перенесенням або зсувом входу, то функція згортки еквівалентна відносно g. На практиці це означає, що після виявлення ознаки в правому нижньому куті зображення мережа може розпізнати ту саму ознаку в будь-якому місці зображення. Мережа, побудована на повнозв'язних шарах, має повторно проаналізувати шаблон, щоб він з'явився в новій позиції.

Згорткові нейронні мережі є підвидом нейронних мереж прямого поширення і мають з ними багато спільного, а саме: складаються зі штучних нейронів, мають вагу, зсув і здатність до навчання. Вся мережа працює як єдина оцінна функція — отримує зображення на вході і визначає клас зображення на виході. Згорткові нейронні мережі зазвичай використовують повнозв'язні шари з функцією втрат на верхньому рівні, де виконується класифікація зображень.

Звичайні нейронні мережі отримують вектор на вході і перетворюють його через послідовність прихованих шарів. Кожний прихований шар складається з певної кількості нейронів, які повністю зв'язані з нейронами попереднього шару, а останній вихідний шар відображує результат роботи мережі у вигляді оцінки класу.

Згорткові нейронні мережі складаються з таких типів шарів: згорткові шари, об'єднувальні шари, операція нормалізації, операція відсівання, повнозв'язні шари. Згорткові нейронні мережі будуються накладанням вищезгаданих шарів один на одного.

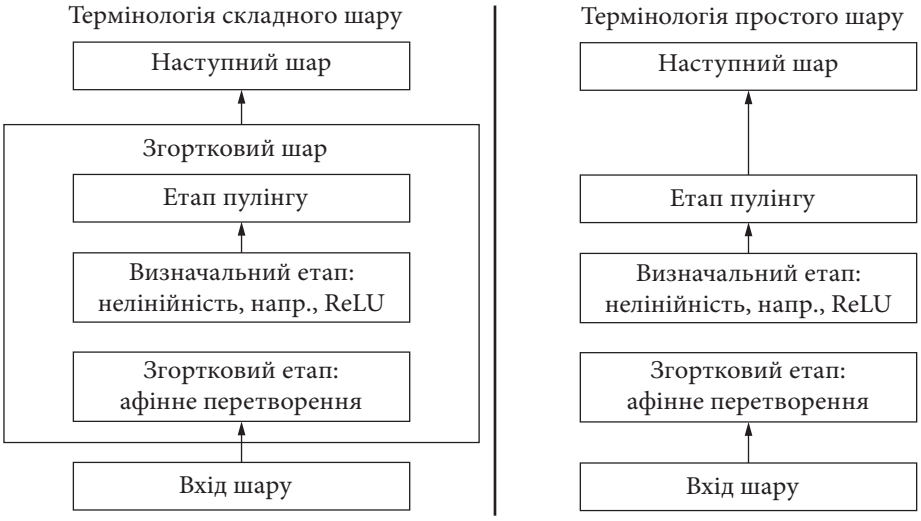


Рис. 11.2. Типові компоненти згорткових мереж

Типові компоненти згорткових нейронних мереж зображено на рис. 11.2.

Прості архітектури згорткових нейронних мереж різної складності показано на рис. 11.3.

З усіх перелічених вище шарів тільки згорткові та повнозв'язні шари можуть бути навчені під час тренування мережі. Шари активації та відсіювання (*dropout*) не вважаються дійсними шарами. Шар пулінгу (*pooling*) настільки ж важливий, як і згортковий шар, і є повністю об'єднувальним, оскільки він безпосередньо впливає на просторовий розмір зображення.

Згортковий шар

У згортковій нейронній мережі кожний шар організовано як масив карт ознак (*feature maps*), замість об'єднання всіх нейронів в одному шарі з нейронами попереднього шару, як у звичайній нейронній мережі. Масиви карт ознак є паралельними площинами фільтрації.

Розміри всіх карт згорткового шару однакові і визначаються за формулою

$$(w, h) = (mW - kW + 1, mH - kH + 1), \qquad (11.2)$$

де (w, h) — обчислюваний розмір згорткової карти; mW — ширина попередньої карти; mH — висота попередньої карти; kW — ширина ядра; kH — висота ядра.

У згортковому шарі зважені суми обчислюються у ковзному вікні і ваги є однаковими для всіх пікселів зображення, так само, як і за звичайної згортки зображення. Частину згорткової нейронної мережі *LeNet*-5 показано на рис. 11.4.

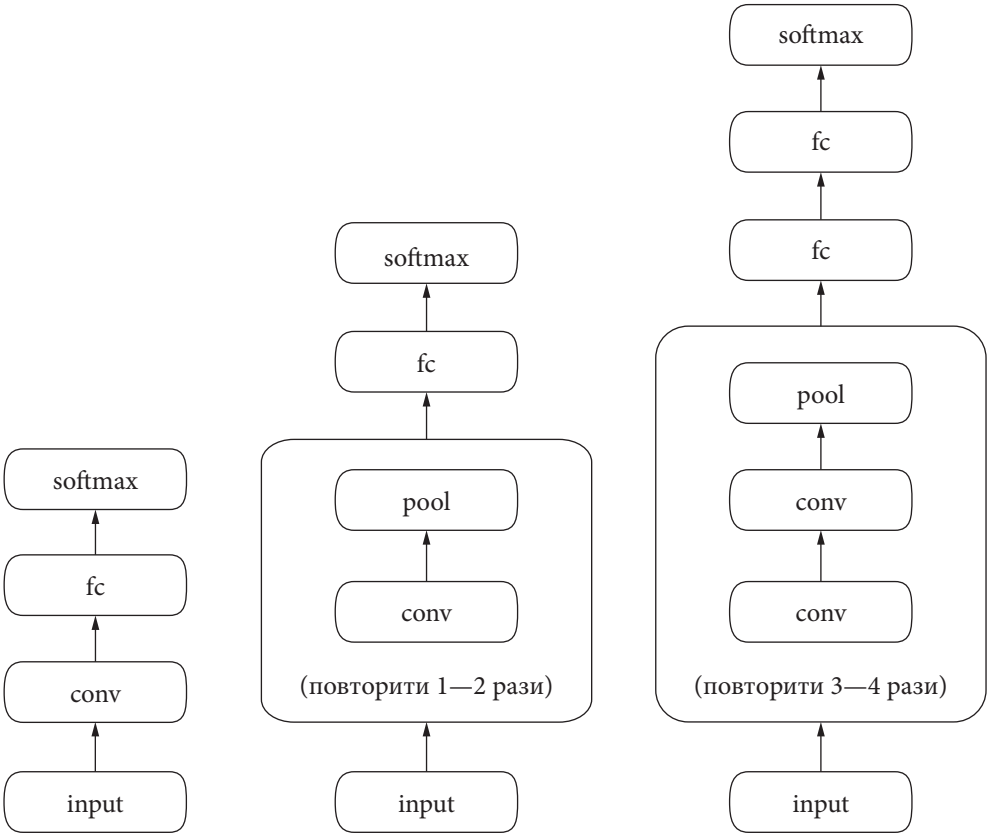


Рис. 11.3. Архітектури згорткових нейронних мереж різної складності

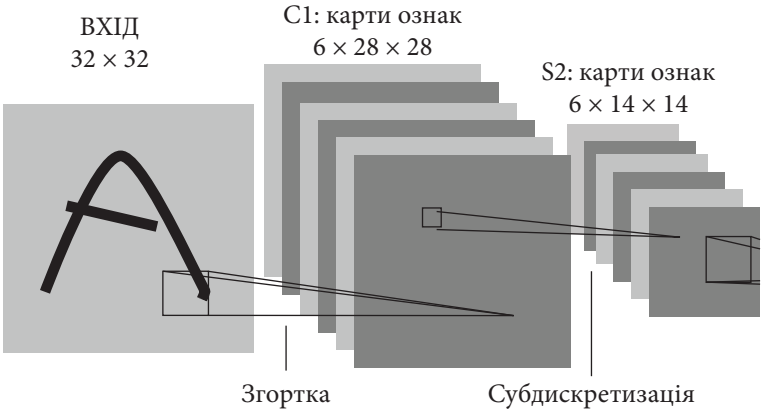


Рис. 11.4. Частина згорткової нейронної мережі *LeNet*-5

На відміну від звичайної згортки, де кожний фільтр застосовується до кожного кольорового каналу, згортка в нейронній мережі зазвичай є лінійною комбінацією результатів активації кожного з C_1 вхідних каналів попереднього шару і використовує різні ядра згортки для кожного з C_2 вихідних каналів. Суть цього підходу полягає в тому, що основним завданням згорткових шарів мережі є отримання локальних ознак, а потім їх комбінування в різних послідовностях для отримання точніших ознак.

Зважену лінійну комбінацію в згортковому шарі можна записати як

$$s(i, j, c_2) = \sum_{c_1 \in C_1} \sum_{(k,l) \in N} w(k, l, c_1, c_2)\, x(i + k, j + l, c_1) + b(c_2), \qquad (11.3)$$

де $x(i, j, c_1)$ — результат з попереднього рівня; N — зсув на два розміри ядра, їх кількість дорівнює S^2; $c_1 \in C_1$ означає, що $c_1 \in [0, C_1)$.

Слід зазначити, що оскільки зміщення (k, l) додаються (а не віднімаються) до координат пікселів (i, j), ця операція в принципі є корельованою.

Оскільки ваги в ядрах згортки однакові для всіх пікселів у кожному шарі, ці ваги фактично розподіляються так, ніби ми малюємо зв'язки між пікселями в різних шарах. Це означає, що потрібно навчати набагато менше параметрів, ніж у мережах, де всі шари повнозв'язні.

Агрегування

Шар пулінгу (*pooling layer*) використовують для зменшення розміру вхідних даних. У загальному випадку функція об'єднання розміщується після шару згортки. Основною метою цієї функції є поступове зменшення розміру вхідних даних. Це дає можливість зменшити кількість параметрів та обчислень у мережі [11.1].

Функція пулінгу (*pooling*) замінює вихід мережі в деякій точці агрегованою статистикою сусідніх виходів. Наприклад, операція максимального

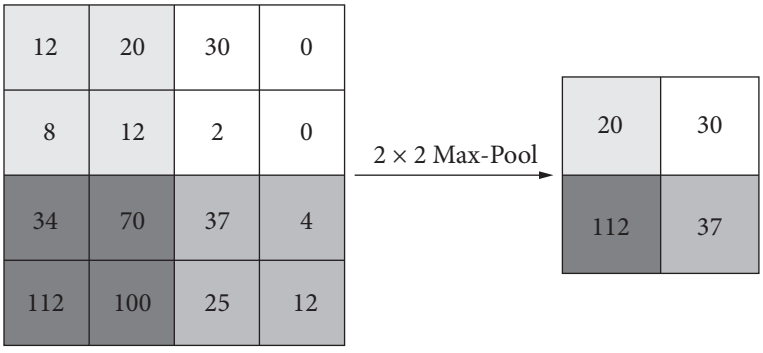


Рис. 11.5. Приклад застосування шару максимального пулінгу (*max-pooling*)

пулінгу (*max-pooling*) повертає максимальне значення в прямокутному околі. Також є усереднення в прямокутному околі. Зазвичай шари максимального об'єднання використовуються всередині згорткової мережі, а усереднення — ближче до останніх шарів, де бажано уникнути використання повнозв'язних шарів на останньому етапі [11.1].

Операція об'єднання робить зображення приблизно інваріантним відносно невеликих паралельних зсувів входу.

У найпоширенішому випадку шар пулінгу працює з фільтром розмірності 2 × 2, який застосовується з кроком 2 і зменшує вхідний сигнал вдвічі за шириною та висотою і, як результат, зменшує зображення на 75 %.

Приклад використання максимального пулінгу наведено на рис. 11.5.

Пакетна нормалізація

Пакетну нормалізацію (*batch normalization*) було використано в [11.3], де шар нормалізації пакетів застосовувався для нормалізації даних перед переданням їх до наступного шару мережі.

Припустимо, що x є мініпакетом, тоді можна обчислити нормалізоване значення $\hat{x}$, використовуючи рівняння

$$\hat{x}_i = \frac{x_i - \mu_\beta}{\sqrt{\sigma_\beta^2 + \varepsilon}}. \tag{11.3}$$

Під час навчання значення μ_β та σ_β^2 обчислюються для кожного мініпакета, де β:

$$\mu_\beta = \frac{1}{m} \sum_{i=1}^{m} x_i;$$

$$\sigma_\beta^2 = \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_\beta)^2,$$

де ε — мале додатне значення, таке як 10^{-7}, що використовується для запобігання діленню на нуль під час обчислення квадратного кореня. Застосування цього рівняння припускає, що активації після операції нормалізації пакета (*batch normalization*) матимуть середнє значення, близьке до нуля, та одиничну дисперсію.

Під час тестування значення μ_β та σ_β замінюються ковзним середнім μ_β та σ_β, що розраховуються у процесі навчання.

Виявилося, що нормалізація пакетів є досить ефективним способом зменшити кількість епох для навчання нейронної мережі. Також нормалізація мініпакетів стабілізує навчання мережі і дає змогу використовувати ширший діапазон значень швидкості навчання.

Як правило, нормалізація мініпакетів відбувається безпосередньо перед застосуванням нелінійності для нормалізаці $x = Wu + b$.

Метод виключення (*dropout*)

Метод виключення (*dropout*) є формою регуляризації, яка запобігає перенавчанню завдяки підвищенню точності тестування, можливо, за рахунок зниження точності навчання [11.2, 11.3]. Шар видалення випадково, з імовірністю p, виключає входи від попереднього шару до наступного шару для кожного мініпакета в навчальному наборі даних.

Видалення зв'язків між двома шарами зображено на рис. 11.6.

Ідея використання видалення зв'язків полягає у зменшенні перенавчання за допомогою явної зміни архітектури мережі під час навчання. Видалення зв'язків гарантує, що жоден вузол мережі не відповідатиме за активацію, якщо він буде поданий певним шаблоном. Натомість гарантується, що є кілька нейронів, які будуть активовані за наявності такого входу — що, зі свого боку, допомагає моделі виконувати узагальнення. Детально роботу методу виключення розглянуто і проаналізовано в розділі 9.

Повнозв'язний шар

Є фундаментальна відмінність між повнозв'язними та згортковими шарами: повнозв'язні шари знаходять і вивчають глобальні закономірності у вхідному просторі ознак, натомість згорткові шари аналізують локальні параметри і, у випадку зображень, знаходять закономірності у невеликих двовимірних вхідних даних. Отже, згорткові мережі мають дві надзвичайно цікаві особливості. Повнозв'язні шари зображено на рис. 11.7.

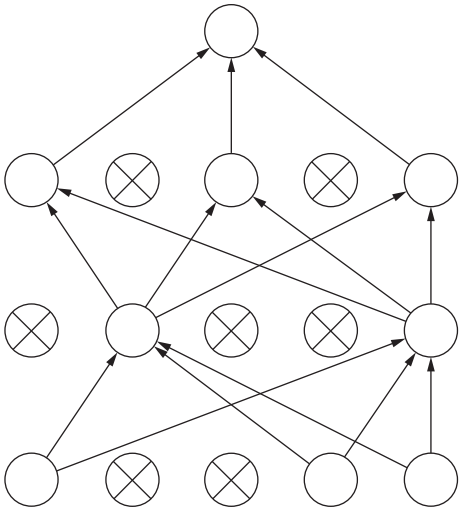


Рис. 11.6. Видалення зв'язків між двома шарами

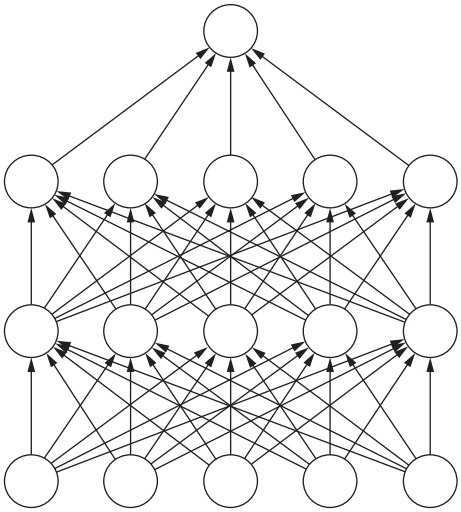


Рис. 11.7. Повнозв'язні шари

У повнозв'язному шарі всі нейрони повністю пов'язані з усіма нейронами попереднього шару, як у стандартній мережі прямого поширення. Такі шари завжди застосовуються в кінці мережі перед класифікатором зображень. Зазвичай використовують один або два повнозв'язні шари перед застосуванням класифікатора.

11.3. Архітектури згорткових нейронних мереж

Архітектура майже всіх класичних згорткових нейронних мереж використовує загальні принципи побудови, а саме: застосування згорткових шарів до вхідного зображення, зменшення розміру зображення разом зі збільшенням кількості карт ознак. Сучасні архітектури використовують нові, вдосконалені методи побудови згорткових мереж для підвищення ефективності навчання.

Принципи побудови класичних згорткових нейронних мереж можна використати як загальні рекомендації щодо проєктування нейронних мереж для класифікації зображень, виявлення об'єктів, сегментації зображень та інших завдань комп'ютерного зору.

Розглянемо основні ідеї класичних згорткових мереж, які було використано в роботі, а саме: *VGG 16, VGG 19, ResNet 50, Inception V3, Xception, Autoencoder.*

VGG 16/VGG 19

Основна ідея архітектури *VGG 16 / VGG 19* полягає у використанні згорткових шарів фільтрів розмірності 3 × 3 з кроком 1 та доповненням (*padding*), щоб розмір вхідного зображення не змінювався після згортки. Розмір зображення зменшується завжди з використанням тих самих шарів максимального пулінгу (*max-pooling*) розмірністю 2 × 2 та кроком 2. Функція *ReLU* завжди використовується як функція активації. Різниця між *VGG 16* та *VGG 19* полягає у кількості шарів навчання (16 і 19 відповідно), решта архітектури однакова.

Архітектуру *VGG 16* показано на рис. 11.8 [11.1—11.3].

Зображення розміром 224 × 224 пікселів подається на вхід першого згорткового шару. Після цього дані проходять через стек згорткових шарів, які використовують фільтри з рецептивним полем розміром 3 × 3 пікселі.

Одна з конфігурацій використовує згортковий фільтр розміром 1 × 1 піксель, який можна розглядати як лінійне перетворення вхідних каналів (і подальшу нелінійність).

Доповнення (*padding*) вхідних даних згоркового шару вибирають так, щоб зберегти просторову роздільну здатність після згортки. Операція пу-

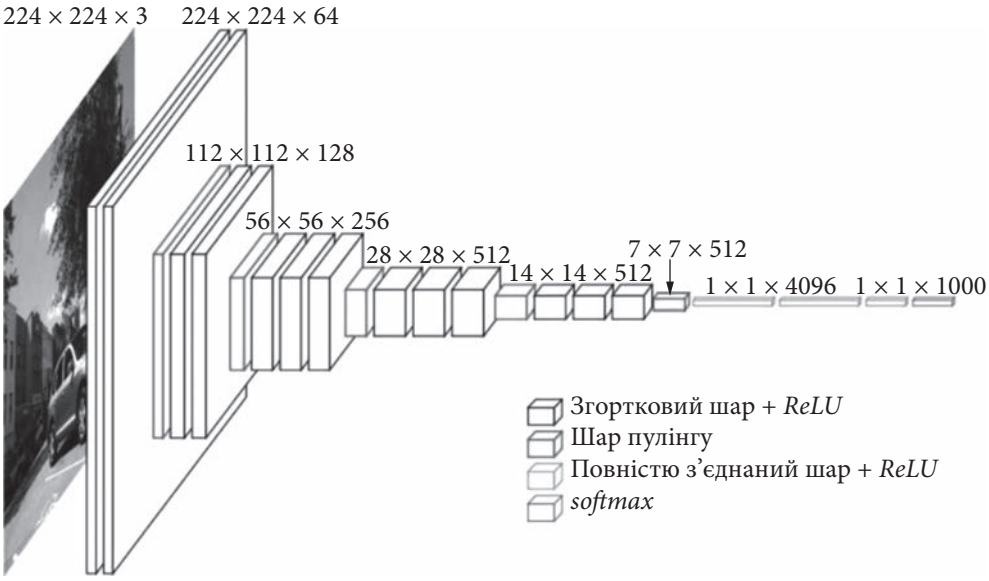


Рис. 11.8. Архітектура згорткової мережі *VGG 16*

лінгу (*maxpooling*) виконується ковзним вікном розміром 2 × 2 пікселі та з кроком 2. Після стека згорткових шарів розміщено три повнозв'язні шари: перші два мають 4 096 нейронів, а третій — 1 000 нейронів (оскільки архітектура призначена для класифікації зображень з *ImageNet*). За ним іде шар *softmax* для багатокласової класифікації вхідних зображень. Як функцію активації всі приховані шари використовують функцію *ReLU* [11.17].

Оскільки нейронні мережі *VGG 16* і *VGG 19* часто дають дуже близькі результати в задачах класифікації, інженери нерідко вибирають згорткову мережу *VGG 16* як основу для подальшого вдосконалення, оскільки вона швидше навчається і використовує менше пам'яті.

Мережа *ResNet 50*

Глибокі згорткові нейронні мережі перевищили рівень людини щодо класифікації зображень у 2015 р. [11.3]. Глибокі мережі виявляють ознаки низького, середнього та високого рівнів у перехресному шарі, а зі збільшенням кількості шарів можуть збільшити кількість ознак, які можна виділити у зображенні. Кількість шарів у стеку має вирішальне значення.

Проблема збільшення кількості шарів тісно пов'язана з проблемою градієнтів, які зникають, що перешкоджає збіжності моделі. Коли більш глибока мережа починає повертати значення градієнта, виникає проблема: зі збільшенням глибини мережі точність спочатку зростає, а потім швидко погіршується.

Щоб вирішити цю проблему, було введено т. зв. залишкову (*residual*) одиницю навчання. Замість очікування, що кілька послідовних шарів безпосередньо відповідають бажаному базовому зображенню, можна явно «дозволити» шарам відповідати залишку. Вираз $F(x) + x$ може бути реалізований за допомогою нейронних мереж з короткими шляхами з'єднань та додаванням до виходів наступних шарів (рис. 11.9).

Короткі з'єднання (*shortcut connections*) пропускають один або кілька шарів і додаються до виходів наступних шарів [11.35].

Якщо бажано поліпшити якість навчання глибокої нейронної мережі (рис. 11.10), уникаючи її перенавчання, потрібно додати скорочені шляхи, як зображено на рис. 11.11. Виявлено, що додавання скорочених шляхів значно покращує точність навчання, при цьому уникаючи перенавчання, як показано на рис. 11.12 [11.3].

Мережа *ResNet 50* складається з великого стека з 50 шарів, об'єднаних у залишкові блоки, кожен з яких має два згорткові шари 3 × 3 [11.2, 11.12]. Періодично кількість фільтрів подвоюється, а їх розмір зменшується з кроком 2.

На початку архітектури є додатковий згортковий шар. Крім того, *ResNet 50* не має повнозв'язних шарів у кінці — використовується лише

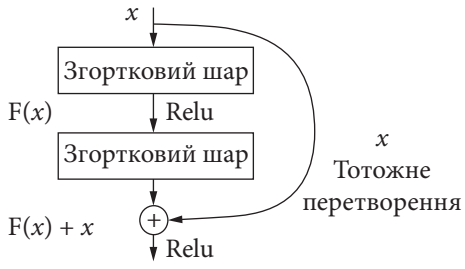


Рис. 11.9. Залишковий блок

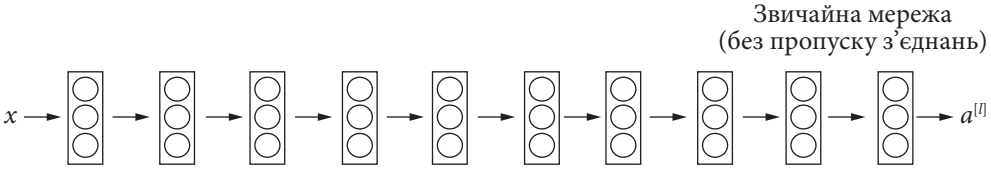


Рис. 11.10. Прості нейронні мережі глибокого навчання

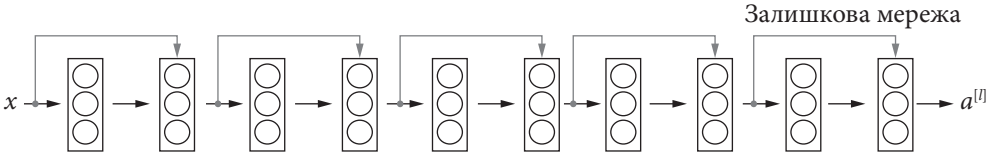


Рис. 11.11. Залишкові нейронні мережі

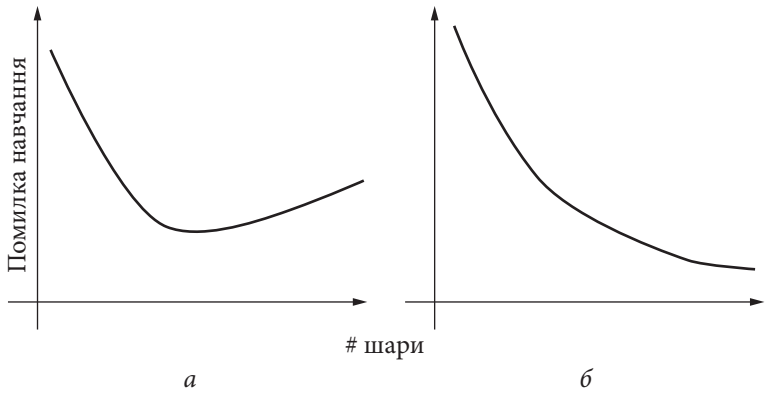


Рис. 11.12. Графік процесу навчання: *а* — звичайної згорткової нейронної мережі (*CNN*); *б* — мережі *ResNet 50*

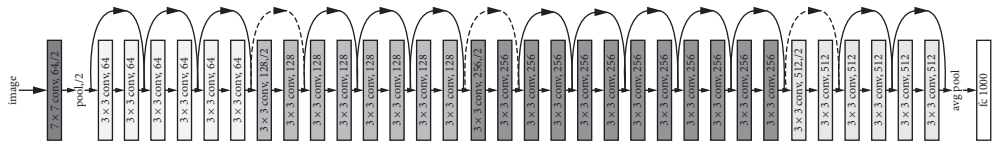


Рис. 11.13. Архітектура згорткової нейронної мережі (CNN) *ResNet 50*

один шар для класифікації зображень. Додатково після кожного згорткового шару використовується пакетна нормалізація (*batch normalization*). Для навчання застосовують метод стохастичного градієнтного спуску.

Архітектуру *ResNet 50* зображено на рис. 11.13.

Мережа *Inception*

Під час розроблення згорткової нейронної мережі потрібно вирішити, який розмір ядра фільтра використовувати для найкращої точності класифікації. Інженери *Google* вирішили не перейматися цим питанням і запропонували замість однієї згортки виконати відразу три розмірністю 1×1, 3×3, 5×5 та пулінг (*max-pooling*), а результати цих згорток додавати до стека, який і буде результатом такої операції. Цей блок, що складається зі стека результатів згорток 1×1, 3×3, 5×5 та *max-pooling*, називається початковою коміркою (*Inception cell*) [11.1].

Структуру модуля початкової комірки детально показано на рис. 11.14.

Концепцію структури модуля початкової комірки покладено в основу побудови згорткової нейронної мережі *Inception*, яка складається з великої кількості послідовно з'єднаних комірок типу *Inception*. Детальну архітектуру мережі *Inception* зображено на рис. 11.15 [11.1].

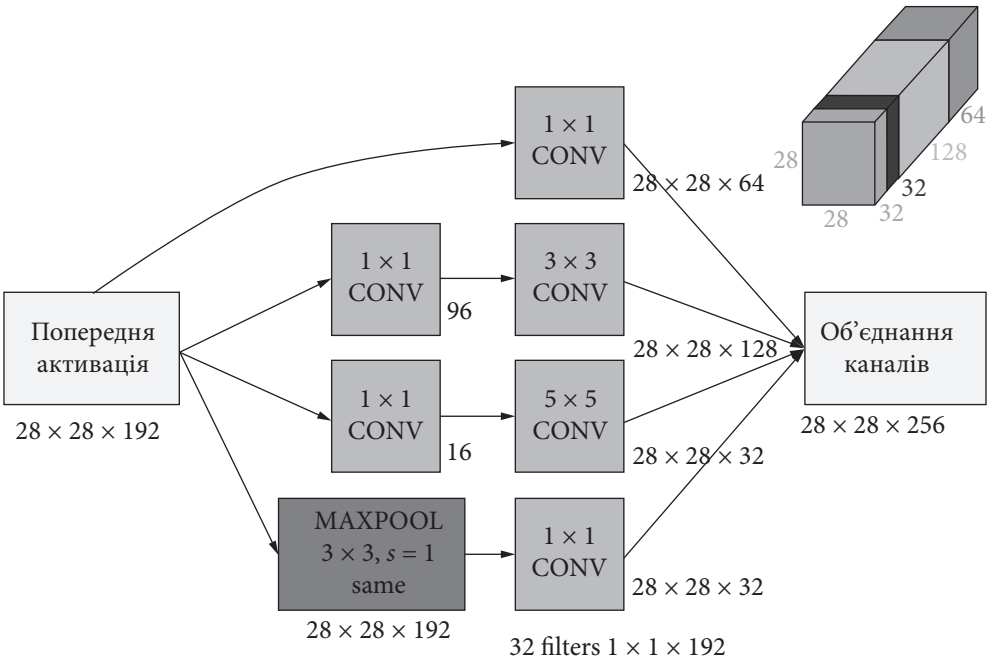


Рис. 11.14. Структура модуля початкової комірки (*Inception cell*)

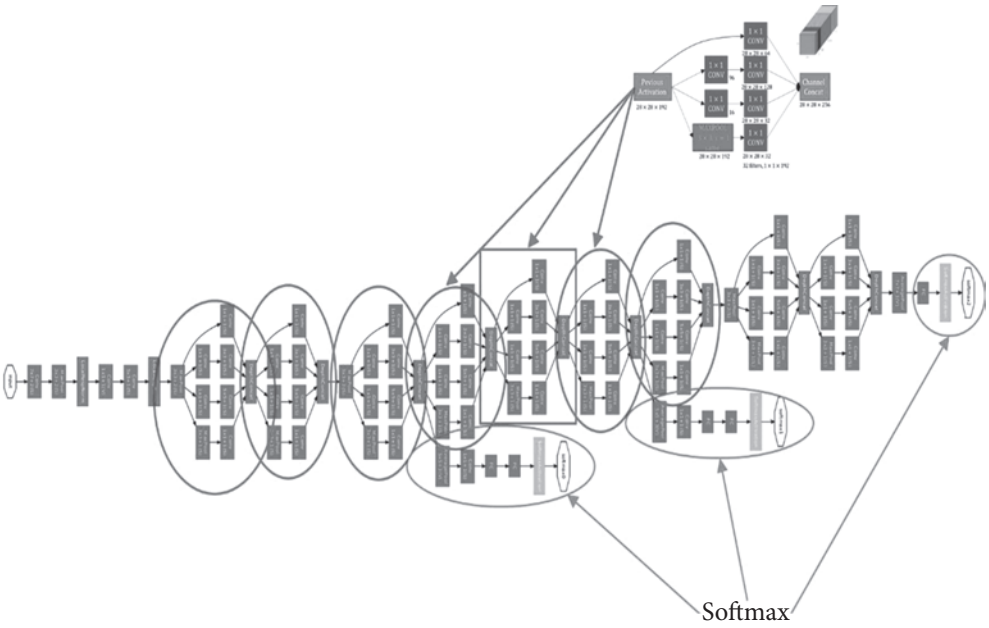


Рис. 11.15. Архітектура *CNN Inception*

Як показано на рис. 11.15, є кілька бічних виходів з повнозв'язними шарами та класифікатором. Кожен з наявних бічних виходів виконує класифікацію зображень разом з останнім шаром. Ця спроба має регулювальний вплив на початкову мережу і запобігає її перенавчанню.

Згорткова нейронна мережа *Xception*

Архітектура *Xception* — «екстремальна» версія режиму *Inception* [11.1, 11.2]. Ця версія використовує ядро 1 × 1 для перетворення всіх каналів вхідного зображення. Потім для кожного каналу зображення окремо використовується просторова згортка [11.19]. Цю операцію показано на рис. 11.16.

Ця «екстремальна» модель модуля *Inception* майже повністю ідентична операції згортки з поділом за глибиною (*depth wise separable convolution*), яка використовується в нейронних мережах з 2014 р.

Згортка з поділом за глибиною складається зі згортки за глибиною (*depthwise convolution*) та просторової згортки, яка виконується незалежно для кожного вхідного каналу і за якою йде операція поточкової згортки (*pointwise convolution*). Згортка 1 × 1 виконується над результатом операції глибинної згортки.

Архітектура моделі *Xception* повністю базується на шарах згортки за глибиною і складається з 36 шарів, об'єднаних у 14 модулів, які, зі свого боку, з'єднані лінійними залишковими зв'язками, за винятком першого і останнього шарів.

Детальну схему архітектури *Xception* зображено на рис. 11.17.

Ця мережа має дуже високу швидкість як навчання, так і прогнозування.

Згортковий автокодувальник

Одним з типів згорткових нейронних мереж, які навчаються без учителя, є архітектура згорткового автокодувальника. Ідея полягає в тому, щоб навчити глибоку нейронну мережу відтворювати на виході зображення, подібні до поданих на вхід. Загалом автокодувальник складається з трьох основних частин: кодувальника (*encoder*), кодового шару (*latent channel*) та декодувальника (*decoder*)[11.5].

Загальна архітектура кодувальника полягає в послідовному стисканні вхідного зображення на кожному наступному шарі нейронної мережі до досягнення кодового шару, розміщеного в центральній частині мережі. На рівні кодового шару нейронна мережа намагається виділити важливі ознаки вхідних зображень, а потім відтворити їх на етапі декодування.

Архітектура декодувальника повністю протилежна архітектурі кодувальника. Декодувальник намагається відтворити вхідні зображення зі стиснутих даних кодового шару з найменшою похибкою.

Архітектуру загорткового автокодувальника показано на рис. 11.18.

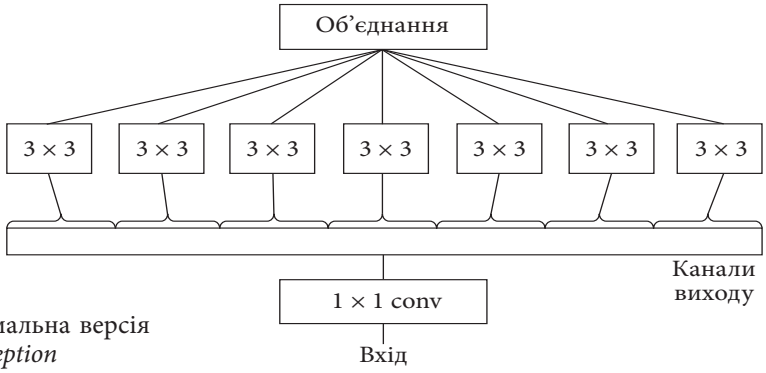


Рис. 11.16. Екстремальна версія модуля *Inception*

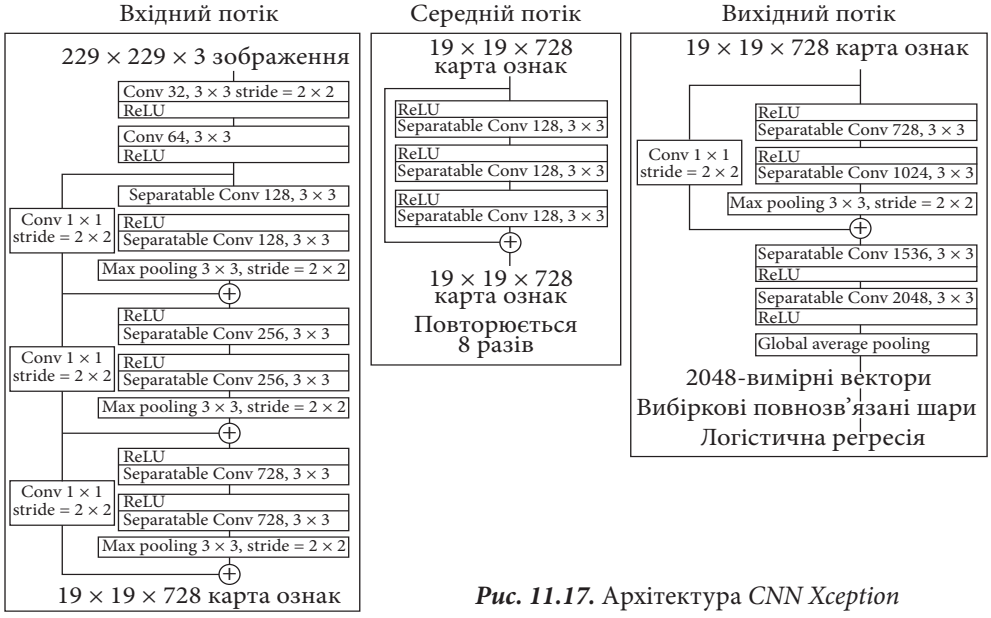


Рис. 11.17. Архітектура *CNN Xception*

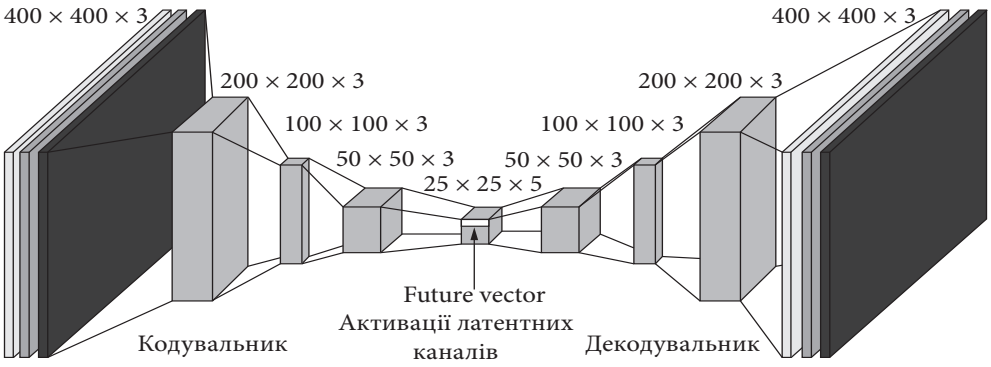


Рис. 11.18. Архітектура згорткового автокодувальника

Однією з основних переваг автокодувальника є його здатність швидко навчатися на немаркованих даних. Ідея використання автокодувальника як класифікатора зображень полягає в тому, щоб використовувати кодувальник і кодовий шар як екстрактор вектора ознак зображення, а потім застосовувати цей код для класифікації зображень. Весь процес складається з двох етапів. На першому етапі навчають автокодувальник відтворювати вхідні зображення на виході з мінімальною похибкою. На другому етапі прихований канал першої частини кодувальника використовують як екстрактор вектора ознак, додають класифікатор зображень і навчають цей класифікатор.

Висновки

У цьому розділі розглянуто основи теорії згорткових нейронних мереж (ЗНМ), наведено структуру сучасних згорткових мереж.

Розглянуто основні ідеї та структуру класичних згорткових мереж, які описано і використано в роботі: *VGG 16, VGG 19, ResNet 50 V2, Inception V3, Xception, Autoencoder;* проаналізовано їхні властивості. Застосування згорткових нейромереж описано в актуальних завданнях медичної діагностики.

СПИСОК ЛІТЕРАТУРИ

11.1. Goodfellow I. Bengio Y., Courville A. Deep Learning. MIT Press, 2016. URL: http://www.deeplearningbook.org

11.2. LeCun Y., Bottou L., Bengio Y., Haffner P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*. 1998. 86 (11). P. 227—2324. https://doi.org/10.1109/5.726791

11.3. Ioffe S., Szegedy C. Batch Normalization: Accelerating Deep Netwrk Training by Reducing Internal Covariate Shift. 2015. arXiv: https://arxiv.org/pdf/1502.03167.pdf

11.4 Szegedy C., Ioffe S., Vanhoucke V., Alemi A.A. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. 2016. arXiv: 1602.07261. https://doi.org/10.48550/arXiv.1602.07261

11.5 Chollet F. Xception: Deep Learning with Depthwise Separable Convolutions. *2017 IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*. 2017. P. 1800—1807. https://doi.org/10.1109/CVPR.2017.195

11.6. Srivastava N., Hinton G., Krizhevsky A., Sutskever I., Salakhutdinov R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. of Machine Learning Research*. 2014. Vol. 15. P. 1929—1958.

11.7. Duchi J.C., Hazan E., Singer Y. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *J. of Machine Learning Research*. 2011. Vol. 12. P. 2121—2159. URL: http://jmlr.org/papers/volume12/duchi11a/duchi11a.pdf

11.8. Tieleman T., Hinton G. RMSProp: Divide the gradient by a running average of its recent magnitude [Lecture 6e, slide 29]. *COURSERA: Neural Networks for Machine Learning.* URL: https://homl.info/58

11.9. Michelucci U. Applied Deep Learning: A Case-Based Approach to Understading Deep Neural Networks. *J. of Machine Learning Research*. 2018. Vol. 19. P. 3697—3701.

11.10. Paterson J., Gibson A. Deep Learning: A Practitioner's Approach. 1st ed. Kindle Edition, 2017. 538 p.

11.11. Kingma D. P., Ba J. Adam: A Method for Stochastic Optimization. *CoRR,* abs/1412.6980, 2014. URL: http://arxiv.rg/abs/1412.6980

11.12. Simonyan K., Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR*, abs/1409.1556. 2014. https://doi.org/10.48550/arXiv.1409.1556

11.13. He K., Zhang X., Ren S., Sun J. Deep Residual Learning for Image Recognition. *2016 IEEE Conf. on Computer Vision and Pattern Recognitio (CVPR)*. P. 770—778. https://doi.org/10.1109/CVPR.2016.90

11.14. He K., Zhang X., Ren S., Sun J. Identity Mappings in Deep Residual Networks. *2016 Europ. Conf. on Computer Vision (ECCV)*. Amsterdam, The Netherlands, Oct. 2016. P. 630—645. https://doi.org/10.1007/978-3-319-46493-0_38

11.15. Szegedy C., Liu W., Jia Y., Sermanet P., Reed S.E., & Anguelov D.et al. Going deeper with convolutios. *2015 IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*. 2015. P. 1—9. https://doi.rg/10.1109/CVPR.2015.7298594

11.16. Szegedy C., Vanhoucke V., Ioffe S., Shlens J., Wojna Z. Rethinking the Inception Architecture for Computer Vision. *2016 IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*. 2016. P. 2818—2826. https://doi.org/10.1109/CVPR.2016.308

11.17. Masci J., Meier U., Ciresan D.C., Schmidhuber J. Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction. *Int. Conf. on rtificial Neural Networks*. 2011. https://doi.org/10.1007/978-3-642-21735-7_7

11.18. Zaychenko Yu., Hamidov G. Hybrid Fuzzy CNN Network in the Problem of Medical Images Classification and Diagnostics. *Proceedings of 15th Int. Conf. on Natural Computation, Fuzzy Systems and Knowledge Discovery FSKD-2019.* & *5th Int. Conf. on Harmony Search, Soft Computing and Applications.* P. 1—6.

11.19. Zaychenko Yu., Zaychenko H., Hamidov G. Structure Optimization of New Generation Computer Networks. *The 2017 10th Int. Congress on Image and Signal Processing, BioMedical Engineering and Informatics.* China, Shanghai, October 13—16, 2017.

11.20. Zaychenko Yu., Hamidov G., Varga I. Medical images of breast tumors diagnostics with application of hybrid CNN-HHM network. *Системні дослідження та інформаційні технології*. 2018. No. 4. С. 37—47.

11.21. Зайченко Ю. П., Здор К. А., Гамидов Г. Диагностика МРТ-изображений опухолей головного мозга с использованием гибридных сверточных нейронечетких сетей. *Системні дослідження та інформаційні технології.* 2020. No. 1. С. 68—77. https://doi.org/10.20535/SRIT.2308-8893.2020.1.06

11.22. Зайченко Ю.П., Гамидов Г. Исследование сверточных нейронных сетей в задачах обработки медицинских изображений и классификации опухолей молочной железы. *Int. J. "Information Theories and Applications*". 2021. Vol. 28 (2). С. 178—199. https://doi.org/10.54521/ijita28-02-p05

11.23. Zaychenko Y., Hamidov G., Zaichenko H. Investigation of Convolutional Neural Networks in the Tasks of Medical Images Analysis and Classification of Breast Tumors. *14th Int. Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI 2021)*. Shanghai, China, 2021, 23—25 October. P. 1—6.

11.24. Zaychenko Y., Hamidov G. Hybrid convolutional neuro-fuzzy networks for diagnostics of MRI-images of brain tumors. *Advances in Intelligent Systems and Computing*. 2021. 1265 AISC. P. 147—155.

11.25. Zaychenko Y., Hamidov G., Zaichenko H. Investigation of recurrent networks LSTM in the problem of Covid-19 forecasting. *Proceedings of the 16th Int. Conf. "Computer science and Information Techologies" CSIT-2022.* Lviv, Ukraine, 22—25 September 2021. P. 56—61.

11.26. Zaychenko Yu., Hamidov G. Hybrid GMDH Deep Learning Networks. State-of art. Structure and Parameters Optimization. *Proceedings of the Int. Conf. "Information Technologies* & *Implementations" (IT&I 2021).* Kyiv, 1—3 December, 2021.

11.27. Zaychenko Yu., Naderan M., Hamidov G. Hybrid convolution network for medical images processing and breast cancer detection. *System research & Information technologies*. 2022. No. 2. P. 85—93. https://doi.org/10.20535/SRIT.2308-8893.2022.2.06

11.28. Krizhevsky A., Sutskever I., Hinton G. E. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*. 2012. Vol. 25. P. 1097—11054.

11.29. Hochreiter S., Schmidhuber J. Long Short-Term Memory. *Neural Computation*. 1997. No. 9. P. 1735—1780.

11.30. Fischer T., Krauss C. Deep Learning with Long Short-Term Memory Networks for Financial Market Predictions. *European J. of Operational Research*. 2018. No. 270. P. 654—669.

11.31. LeCun Y., Bengio Y., Hinton G.E. Deep learning. *Nature*. 2015. Vol. 521. No. 7553. P. 436—444.

11.32. Deep Learning in Healthcare. URL: https://missinglink.ai/guides/deep-learning-healthcare/deep-learning-healthcare/

11.33. How transfer learning works for tasks with medical images. URL: https://neurohive.io/ru/novosti/kak-transfer-learning-ispolzujut-dlya-zadach-s-medicinskimi-snimkami

11.34. Michelucci U. Applied Deep Learning: A Case-Based Approach to Understanding Deep Neural Networks *Berkeley, CA*: Apress, 2018. 431 p.

11.35. Paterson J., Gibson A. Deep Learning: A Practitioner's Approach. 1st ed. Kindle Edition, 2017. 538 p.

11.36. Szeliski R. Computer Vision: Algorithms and Applications. 2nd ed. URL: http://szeliski.org/Book/

11.37. Chollet F. Deep learning with Python. Shelter Island, NY: Manning, 2018. 386 c.

11.38. Convolutional Neural Networks (CNNs / ConvNets). URL: https://cs231n.github.io/convolutional-networks/

11.39. Rosebrock A. Deep Learning for Computer Vision with Python. Starter Bundle. URL: https://www.pyimagesearch.com/deep-learning-computer-vision-python-book/

11.40. Ioffe S., Szegedy C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. 2015. P. 189, 193. ar Xiv: http://arxiv.org/abs/1502.03167

11.41. Very Deep Convolutional Networks for Large-Scale Image Recognition. ar Xiv: https://arxiv.org/abs/1409.1556

11.42. VGG in Tensor Flow. URL: https://www.cs.toronto.edu/~frossard/post/vgg16/

11.43. Deep Residual Learning for Image Recognition. ar Xiv: https://arxiv.org/abs/1512.03385

РОЗДІЛ 12

ЗГОРТКОВІ І ГІБРИДНІ НЕЙРОННІ МЕРЕЖІ У ЗАВДАННЯХ АНАЛІЗУ МЕДИЧНИХ ЗОБРАЖЕНЬ ТА КЛАСИФІКАЦІЇ ПУХЛИН МОЛОЧНОЇ ЗАЛОЗИ

12.1. Опис набору даних *BreakHis*

12.2. Вибір архітектури згорткової нейронної мережі

12.3. Вибір гіперпараметрів тренувальних моделей

12.4. Результати бінарної класифікації пухлин молочної залози

12.5. Результати багатокласової класифікації зображень

12.6. Гібридна ЗНМ ННМ для класифікації пухлин молочної залози

12.7. Гібридна згорткова нейронна мережа на основі автокодувальника для виявлення раку молочної залози

Висновки

Список літератури

Рак молочної залози є найпоширенішим онкологічним захворюванням серед жінок як у розвинених, так і у менш розвинених країнах. Згідно зі статистикою за 2019 рік, захворюваність на нього становить 11,6 % серед усіх видів раку, що у 6,5 % призводить до смерті [12.1—12.2]. Також, за даними *GLOBOCAN*, захворюваність на рак молочної залози зросте з 2 млн пацієнтів у 2018 р. до понад 3 млн у 2046 р., що свідчить про збільшення таких випадків на 46 %. Через невиліковний характер та довготривалий перебіг захворювання рак молочної залози є тягарем для пацієнтів і для осіб, які за ними доглядають [12.2—12.5].

Рання діагностика раку молочної залози є дуже важливою, адже це може допомогти вчасно виявити пухлину і значно знизити ризики цього захворювання. Для діагностики раку онкологи вже багато років використовують медичні зображення, отримані за допомогою таких інструментів як комп'ютерна томографія, магнітно-резонансна томографія та рентген [12.5]. Ці методи є ефективними для виявлення багатьох видів пухлин, але, на жаль, деякі види раку досі не можна надійно діагностувати за їх допомогою.

На сьогодні розроблено декілька моделей і методів для виявлення раку молочної залози, що використовують різні алгоритми машинного навчання. За допомогою технологій штучного інтелекту, як-от нейронні мережі чи метод опорних векторів [12.6—12.8], точність класифікації пухлин підвищилася з 74 до 94 % на наборі даних із 92 зображень.

Упродовж багатьох років текстурні дескриптори були основним засобом отримання ознак зображення для подальшої класифікації. Однак через їх неуніверсальність необхідно розробляти дескриптори для кожного типу зображень. Інший підхід до отримання зображень ґрунтується на системах машинного навчання [12.9].

Сьогодні для вирішення подібних завдань дослідники звернули увагу на згорткові нейронні мережі (ЗНМ) [12.10—12.12]. Їх використання для класифікації зображень перевершує ефективність традиційних методів, заснованих на класичних методах комп'ютерного зору. З одного боку, немає нестачі даних для завдань аналізу медичних зображень, натомість для отримання якісних і правильно класифікованих наборів даних необхідні кваліфікований персонал і значні витрати часу. Саме тому пошук якісного набору даних з достатньою кількістю навчальних прикладів є певним викликом.

Передавальне навчання і збільшення кількості даних [12.13] широко використовують для вирішення проблеми ефективного обсягу навчальних прикладів. Передавальне навчання (*transfer learning*) — це дослідницька задача в машинному навчанні, зосереджена на зберіганні знань, отриманих під час розв'язання однієї задачі, та застосуванні їх до іншої, але пов'язаної задачі.

Техніка збільшення даних ґрунтується на використанні геометричних та інших перетворень з метою розширення наборів даних та забезпечення стабільності нейронних мереж.

Цікавим є вивчення ефективності згорткових нейронних мереж різної архітектури (прості згорткові мережі, *VGG 16, VGG 19, ResNet 50 V2*, *Inception V3, Xception*) та гібридних згорткових мереж у завданнях класифікації раку молочної залози.

Мета цього розділу — дослідження згорткових мереж різної архітектури, а також гібридних мереж у завданнях класифікації раку молочної залози, визначення найбільш придатного типу згорткової мережі для вирішення подібних задач. Отриманий результат можна буде використати для створення вдосконаленої нейронної моделі з метою розширення методів експрес-діагностики та її подальшого застосування у завданнях експрес-діагностики ракових пухлин.

12.1. Опис набору даних *BreakHis*

У дослідженні було використано набір даних *BreakHis*, спеціально розроблений для оцінки ефективності різних підходів і методів медичної візуалізації в діагностиці пухлин молочної залози [12.11].

Набір зображень гістопатології раку молочної залози (*BreakHis*) складається з 7909 мікроскопічних зображень тканин пухлини молочної залози, зібраних у 82 пацієнток за допомогою різних факторів збільшення (40×,

Таблиця 12.1

Структура даних *BreakHis*

Фактор збільшення (*magnification factor*)	Кількість доброякісних зображень (*number of benign images*)	Кількість злоякісних зображень (*number of malignant images*)	Загальна кількість зображень (*general number images*)
40×	652	1370	1995
100×	644	1437	2081
200×	623	1390	2013
400×	588	1232	1820
Загальна кількість зображень (*total number of images*)	2480	5429	7909

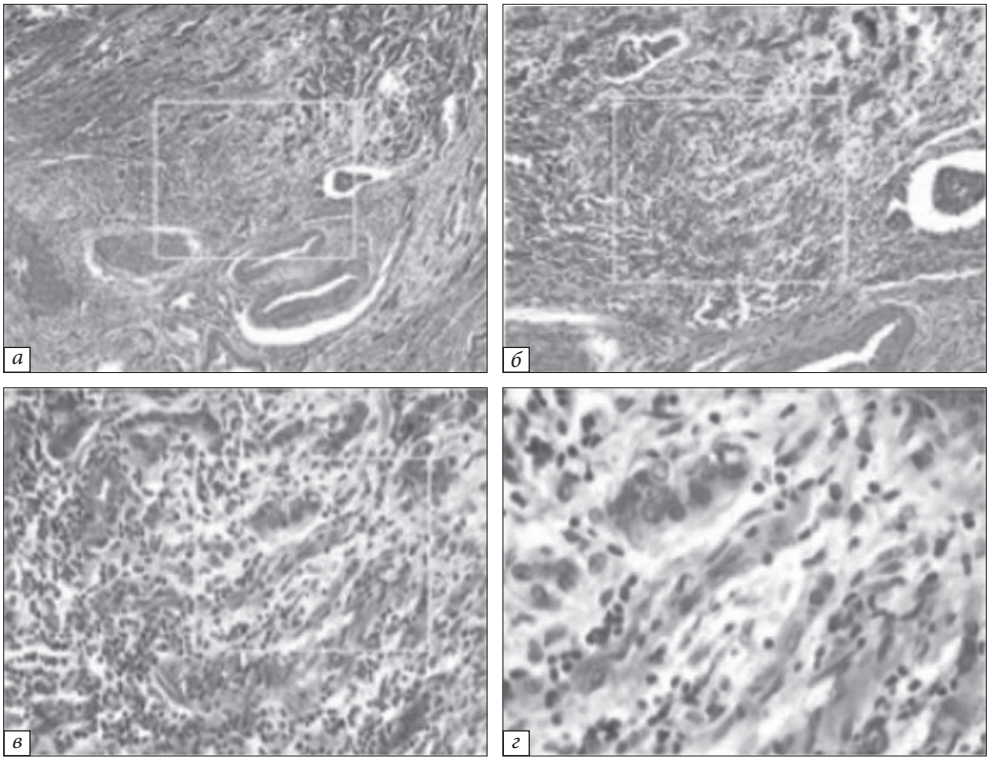


Рис. 12.1. Приклади зображень доброякісних пухлин за різних збільшень: *а* — 40×; *б* — 100×; *в* — 200×; *г* — 400×

100×, 200× і 400×). На сьогодні набір даних містить 2480 доброякісних і 5429 злоякісних зразків (700 × 460 пікселів, 3-канальний *RGB*, 8-бітова глибина в кожному каналі, формат *PNG*).

Набір даних *BreakHis* поділяється на дві основні групи: доброякісні пухлини та злоякісні пухлини. Гістологічно доброякісними називають новоутворення, що не відповідають жодним критеріям злоякісності, як-от, наприклад, помітна клітинна атипія, мітоз, аномалії базальної мембрани, метастазування тощо. Як правило, доброякісні пухлини відносно «невинні», ростуть повільно і залишаються локалізованими. Злоякісна ж пухлина є синонімом раку: ураження може проникати і руйнувати прилеглі структури (місцево інвазивно) та поширюватися на віддалені ділянки (метастазувати), що з часом призводить до смерті.

У поточній версії зразки, що входять до набору даних, було зібрано методом *SOB*, який також називають частковою мастектомією, або ексцизійною біопсією. Така процедура, порівняно з будь-якою голковою біопсією, дає можливість отримати більший зразок тканини і виконується в лікарні під загальною анестезією.

Структуру набору даних *BreakHis* наведено в табл. 12.1.

Доброякісні та злоякісні пухлини молочної залози можна сортувати за типом залежно від того, який вигляд клітини пухлини мають під мікроскопом. Різні типи пухлин молочної залози можуть мати різні прогнози та результати лікування. Наразі набір даних містить чотири гістологічно відмінні типи доброякісних пухлин молочної залози: аденоз (A), фіброаденома (F), філоїдна пухлина (PT) та трубчаста аденома (TA), а також чотири види злоякісних пухлин (раку молочної залози): протоковий рак (DC), рак молочної залози (LC), муцинозний рак (MC) та папілярний рак (PC).

Приклади зображень, отриманих за різних збільшень у цьому наборі даних, наведено на рис. 12.1.

Для роботи з нейронними мережами було використано бібліотеки *TensorFlow* версії 2.1 і *Keras* [12.15].

12.2. Вибір архітектури згорткової нейронної мережі

Для аналізу медичних зображень пухлин молочної залози було досліджено такі архітектури згорткових нейронних мереж [12.15, 12.16]:

- проста згорткова нейронна мережа;
- *VGG 16*;
- *VGG 19*;
- *ResNet 50 V2*;
- *Inception*;
- *Xception*.

Як класифікатор було використано стек із трьох повнозв'язних шарів розміром 512, 128, 64, 8 або 1 (залежно від методу класифікації — бінарної чи багатокласової). Після кожного шару виконувалась операція виключення (*dropout*) з імовірністю 0,5.

Цей вибір архітектури класифікатора ґрунтується на попередніх дослідженнях і виявився найбільш придатним для класифікації зображень згаданого набору.

Як моделі *VGG 16/VGG 19, ResNet 50 V2, Inception, Xception* було використано готові реалізації бібліотеки *Keras,* а також застосовано оригінальний класифікатор. Структуру такої моделі показано в табл. 12.2.

Глибокі згорткові нейронні мережі потребують дуже великої кількості зображень для належного навчання моделі. Цей підхід дає можливість підвищити продуктивність моделі завдяки тому, що модель навчається краще узагальнювати, зменшуючи в такий спосіб ризик перенавчання.

Здатність моделі до узагальнення характеризує її продуктивність, що виявляється вже під час тестування на наборі зображень, які модель раніше не бачила.

Використаний у роботі набір даних *BreakHis* має обмежений розмір [12.11]. Набір складається з 2480 зображень доброякісних пухлин і 5429 зображень злоякісних пухлин. Якщо аналізувати кількість за типом захворювання, кількість зображень може виявитися ще меншою.

Оригінальні зображення з набору даних мають розмір 700 × 460 пікселів. Попередні тести показали, що просто неможливо навчати нейронні мережі на вхідних даних такого розміру (пам'яті не вистачає). Тому для навчання використовувались зображення розміром 227 × 227.

Таблиця 12.2

Структура класифікатора ЗНМ *Xception*

Шар (тип)	Розмірність виходу	Кількість параметрів
Xception (Model)	(None, 7, 7, 2048)	20 861 480
Flatten (Flatten)	(None, 100352)	0
Dropout (Dropout)	(None, 100352)	0
Dense (Dense)	(None, 512)	51 380 736
Activation (Activation)	(None, 512)	0
dropout_1 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 128)	65 664
activation_1 (Activation)	(None, 128)	0
dense_2 (Dense)	(None, 8)	1032
Усього параметрів: 72 308 912		
Навчуваних: 72 254 384		
Ненавчуваних: 54 528		

Таблиця 12.3

Розподіл зображень за типом захворювання та фактором збільшення

Захворювання молочної залози	Фактор збільшення				
	40×	100×	200×	400×	Разом
Аденоз	114	113	111	106	444
Фіброаденома	253	260	264	237	1014
Пухлина Філлодеса	109	121	108	115	453
Тубулярна аденома	149	150	140	130	569
Протокова карцинома	864	903	896	788	3451
Часточкова карцинома	156	170	163	137	626
Муцинозна карцинома	205	222	196	169	792
Папілярна карцинома	145	142	135	138	560

Для розподілу набору даних на навчальну і тестову вибірки набір файлів було випадково перемішано, а потім розділено у співвідношенні 80/20 для збереження структури файлів обох вихідних наборів.

Оскільки кількість зображень обмежена, дані автоматично вирізались із середини оригінального зображення (де ймовірність ідентифікації захворювання є найбільшою) і додатково з деяким перекриттям центрального зображення (ще 8 зображень). Усі екземпляри було відображено горизонтально і вертикально. Всі обрізані файли мали оригінальне ім'я і додатково індекс, який визначав положення обрізаного зображення відносно оригінального. Структура файлової системи в остаточному наборі також зберігалась. У результаті набір даних мав приблизно 14 гігабайтів. Загальна кількість зображень у вихідному наборі становила 213 846.

Як свідчать дані табл. 12.3, розподіл кількості зображень відносно кількості класів є нерівномірним. Тому перед навчанням кількість зображень для всіх класів було вирівняно відносно класу з найменшою кількістю зображень, щоб уникнути упередженості.

12.3. Вибір гіперпараметрів тренувальних моделей

Оскільки згенерований набір даних є досить великим, використання стохастичного градієнтного спуску просто неможливе. У роботі було використано метод мініпакетного стохастичного градієнтного спуску з розміром пакета 16 (для того, щоб модель вміщалась у пам'ять графічного процесора *GPU*) [12.16].

Попередній аналіз навчання мережі показав, що неможливо також використовувати однакове значення параметра швидкості навчання. Як відомо, використання досить великого значення швидкості навчання при-

Таблиця 12.4

Кількість епох навчання для різних структур згорткових нейронних мереж

Модель	Фактор збільшення				
	40×	100×	200×	400×	Усі приклади
Проста згорткова мережа	100	100	100	100	100
VGG 16	50	50	100	100	100
VGG 19	100	100	—	100	100
ResNet 50 V2	50	30	30	30	30
Inception	30	30	30	30	30
Xception	30	30	30	30	30

зводить до того, що алгоритм градієнтного спуску проходить повз точки мінімуму і не збігається [12.16]. Тож у роботі було використано метод послідовного зменшення швидкості навчання. Діапазон значень швидкості навчання варіювався в межах [0,001, 0,00001] з поступовим зменшенням зі збільшенням номера ітерації. У такий спосіб вдалося навчити всі моделі. Кількість епох, використовуваних для навчання моделей, наведено в табл. 12.4 [12.16]. Як випливає з даних таблиці, проста згорткова нейронна модель, моделі *CNN/VGG 16* та *VGG 19* були навчені впродовж 100 епох. Інші моделі можна було навчати протягом 30 епох.

Час, витрачений на навчання моделей *VGG 16, VGG 19* з використанням усіх зображень, становив приблизно 12 годин. Мінімальний час, витрачений на навчання моделей для будь-якого конкретного вибору масштабу збільшення, становив близько 3 годин. Слід зазначити, що приблизний час для повного навчання усіх моделей перевищував два тижні безперервної роботи комп'ютера.

12.4. Результати бінарної класифікації пухлин молочної залози

Точність бінарної класифікації (доброякісна пухлина / злоякісна пухлина) навчених моделей відображує табл. 12.5. Навчання на наборі даних за збільшення 200× було виконано для визначення набору гіперпараметрів навчання. Найкращу модель *Xception* було використано для навчання на повному наборі.

Точність бінарної класифікації цієї моделі становила 98,81 %, що є дуже хорошим результатом навчання моделі на дещо обмеженому наборі даних [12.1, 12.16].

Графік зміни точності навчання та тестування для ЗНМ *Xception* залежно від кількості епох показано на рис. 12.2. Як видно з рисунка, під час

Таблиця 12.5

Точність бінарної класифікації, %, медичних зображень

Модель	Збільшення 200×	Усі приклади
Проста згорткова мережа	83,6	83,0
VGG 16	90,9	90,1
VGG 19	97,6	97,2
ResNet 50 V2	84,4	84,3
Inception	91,7	91,3
Xception	**99,75**	**98,81**

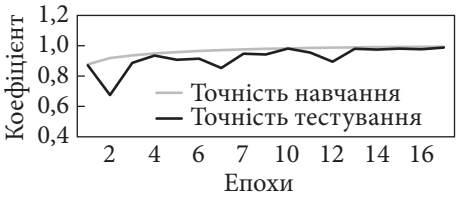


Рис. 12.2. Точність навчання та тестування моделі *Xception* на повному наборі даних

навчання моделі було важко знайти мінімальне значення часу градієнтного спуску. Використання послідовного зменшення параметра швидкості навчання виправило ситуацію, і для повного навчання моделі знадобилося лише 13 епох.

12.5. Результати багатокласової класифікації зображень

Цікавою є проблема багатокласової класифікації з використанням вибраних моделей: аденоз, фіброаденома, філоїдна пухлина, трубчаста аденома (доброякісні пухлини), протоковий рак, рак молочної залози, муцинозний рак, папілярний рак (злоякісні пухлини). Деякі зображення різних типів доброякісних та злоякісних пухлин наведено на рис. 12.3.

На рис. 12.4—12.6 подано результати навчання моделей для багатокласової класифікації з різними факторами збільшення [12.16].

З наведених результатів можна дійти висновку, що швидкість навчання моделей є нерівномірною і залежить від архітектури моделі. Найпростіша модель навчається найповільніше, а модель *Xception* — найшвидше. Навчання моделей на повному наборі даних сприяє збільшенню швидкості навчання, ймовірно, через підвищення здатності моделей до узагальнення.

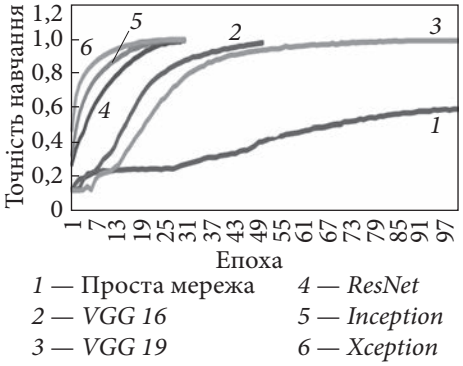
Кінцеві результати точності багатокласової класифікації всіх моделей на тестовому наборі наведено в табл. 12.6 [12.1, 12.16], з якої видно, що моделлю з найвищою точністю багатокласової класифікації є *Xception*.

Таблиця 12.6

Точність багатокласової класифікації, %, усіх ЗНМ для всіх типів пухлин

Модель	Фактор збільшення				
	40×	100×	200×	400×	Усього
Проста згорткова мережа	49,6	54,7	42,8	39,8	72,8
VGG 16	86,3	90,8	87,7	85,7	89,95%
VGG 19	86,2	88,2	—	88	88,4
ResNet 50 V2	84,4	84,7	82,8	83,2	86,1
Inception	83,04	85,6	84,09	75	82,2
Xception	**90,6**	**89 0**	**88,6**	**89**	**90,8**

Рис. 12.3. Типи пухлин: *а—г* — доброякісні, *д—з* — злоякісні



1 — Проста мережа *4 — ResNet*
2 — VGG 16 *5 — Inception*
3 — VGG 19 *6 — Xception*

Рис. 12.4. Швидкість навчання різних моделей ЗНМ з фактором збільшення 100×

1 — Проста мережа *4 — ResNet*
2 — VGG 16 *5 — Inception*
3 — VGG 19 *6 — Xception*

Рис. 12.5. Швидкість навчання різних моделей ЗНМ з фактором збільшення 200×

1 — Проста мережа *4 — ResNet*
2 — VGG 16 *5 — Inception*
3 — VGG 19 *6 — Xception*

Рис. 12.6. Швидкість навчання різних моделей ЗНМ з фактором збільшення 400×

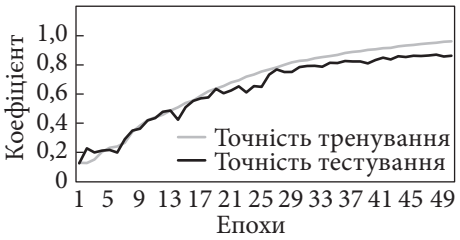


Рис. 12.7. Навчання *VGG16* на повному наборі даних з усіма факторами збільшення

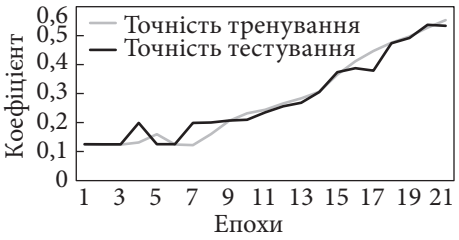


Рис. 12.8. Навчання *VGG19* на наборі даних з усіма факторами збільшення

Рис. 12.9. Навчання ЗНМ *Inception V3* на наборі даних з усіма факторами збільшення

На рис. 12.7—12.9 наведено графіки динаміки точності навчання та тестування залежно від кількості епох для розглянутих ЗНМ для всіх зображень.

Як випливає з наведених результатів, згорткові нейронні мережі *VGG 19* та *Xception* є найбільш ефективними для вирішення завдання класифікації раку молочної залози.

Отже, було досліджено ефективність класичних згорткових нейронних мереж різної архітектури у вирішенні завдання обробки медичних зображень та класифікації пухлин молочної залози. Для цієї мети було використано спеціально розроблений набір даних *BreakHis.* Оскільки набір має обмежений розмір і неможливо навчати моделі на оригінальних зображеннях через велику кількість інформації в них, було підготовлено дані і згенеровано набір зображень розміром 227 × 227.

Для підвищення збіжності моделей використовували метод зменшення параметра швидкості навчання зі збільшенням номера ітерації.

Було вирішено задачу бінарної класифікації зображень. Точність бінарної класифікації пухлин на тестовому наборі зображень з використанням моделі *Xception* сягнула 98,81 % (найвища).

Додатково досліджено завдання багатокласової класифікації пухлин молочної залози. Було з'ясовано, що моделям *VGG 16* і *VGG 19* потрібно набагато більше ітерацій для навчання, ніж *ResNet 50 V2, Inception* та *Xception.*

Проведено експериментальні порівняння розглянутих моделей ЗНМ і методів класифікації раку молочної залози. Найкращою виявилася модель *Xception* з використанням оригінального класифікатора, точність багатокритеріальної класифікації якої становила 90,8 %.

Як остаточний результат дослідження було встановлено, що згорткова нейронна мережа моделі *Xception* є досить перспективною для експрес-діагностики медичних зображень у завданні класифікації раку молочної залози [12.16].

12.6. Гібридна ЗНМ ННМ для класифікації пухлин молочної залози

У попередньому розділі розглянуто і досліджено проблему класифікації пухлин молочної залози із застосуванням різних ЗНМ стандартної структури. Виявлено найкращу структуру ЗНМ для вирішення подібних завдань. Водночас дослідження нових архітектур ЗНМ, що становлять об'єднання різних типів нейронних мереж, та визначення їх ефективності для класифікації пухлин молочної залози також мають велике значення. У цьому підрозділі розглянуто і досліджено гібридну ЗНМ ННМ для розв'язання поставленої задачі. У гібридній ЗНМ *VGG16* використовується для вилучення ознак зображень, а нечітка нейронна мережа *NEFClass* слугує класифікатором.

Нечітка нейронна мережа *NEFClass*

Нейронну мережу *NEFClass* вперше запропонували Д. Наук і Р. Круз [12.17]. Її було модифіковано і розвинено в подальших дослідженнях (так звана *NEFClass M* [12.18]).

У роботі [12.19] було розроблено і досліджено алгоритми навчання для мережі *NEFClass:* стохастичний градієнтний, метод спряжених градієнтів та генетичний алгоритм для розпізнавання зразків оптичних зображень (див. розд. 5). У наступній роботі [12.20] ННМ *NEFClass* було успішно застосовано для аналізу медичних зображень тканин шийки матки, отриманих за допомогою кольпоскопа та діагностики. Детальний опис структури і алгоритмів навчання ННМ *NEFClass* та її модифікації наведено вище.

Основними перевагами ННМ *NEFClass* як класифікатора є: можливість працювати з неповними та нечіткими вхідними даними; виконання нечіткої класифікації вхідних зразків (зображень) з використанням т. зв. функцій належності; швидкість і висока точність [12.18, 12.21].

В наступній роботі [12.21] було розроблено та досліджено гібридну ЗНМ з використанням *NEFClass* для аналізу та діагностики пухлин молочної залози. Після генерування бази правил на першому етапі навчання на другому етапі знаходяться параметри функцій належності нечітких множин. На цьому етапі використовується кілька алгоритмів навчання, таких як стохастичний градієнтний спуск, метод спряжених градієнтів, диференціальна еволюція та т. зв. «*basin hopping*»-алгоритм.

Алгоритм навчання ННМ *NEFClass*

Стохастичний градієнтний алгоритм для ННМ NEFClass. Для навчання ННМ *NEFClass* було використано критерій

$$e = \sum_{i=1}^{M}(t_i - NET_i(W))^2 \to \min, \tag{12.1}$$

де t_i — бажане вихідне значення нейронної мережі; $NET_i(W)$ — фактичне значення i-го виходу нейронної мережі для матриці вагових коефіцієнтів $W = [W^I, W^O]$, $W^I = W(x, R) = \mu_j(x)$, $W^O = W(R, C)$.

Критерієм $e(W)$ є середньоквадратична похибка апроксимації [12.21].

Нехай функцією активації для нейронів прихованого шару (нейронів правил) буде

$$O_R = \prod_{i=1}^{n}\mu_{j_i}^{(i)}(x_i), \quad j = 1, \ldots, q_i, \tag{12.2}$$

де $\mu_{j_i}^{(i)}$ — функція належності, яка має форму

$$\mu_{j_i}^{(i)}(x) = e^{-\frac{(x - a_{j_i})^2}{b_{j_i}^2}} \tag{12.3}$$

а функція активації вихідного шару нейронів має вигляд

$$NET_c = \max_{R \in U_2} W(R,C)O_R. \tag{12.4}$$

Розглянемо градієнтний алгоритм навчання нечіткого перцептрона [12.21].

1. Нехай $W(n)$ — поточне значення матриці вагових коефіцієнтів. Алгоритм має таку форму:

$$W(n + 1) = W(n) - \gamma_{n+1}\nabla_w e(W(n)), \tag{12.5}$$

де γ_n — крок на n-й ітерації; $\nabla_w e(W(n))$ — градієнт (напрямок) критерію, (12.1).

2. На кожній ітерації спочатку навчаються вхідні ваги W, які залежать від параметрів a і b [12.21]:

$$a_{ji}(n + 1) = a_{ji}(n) - \gamma_{n+1}\frac{\partial e(W)}{\partial a_{ji}}; \tag{12.6}$$

$$b_{ji}(n + 1) = b_{ji}(n) - \gamma'_{n+1}\frac{\partial e(W)}{\partial b_{ji}}, \tag{12.7}$$

де γ'_{n+1} — крок для параметра b.

Для цього знаходимо такі похідні:

$$\frac{\partial e(W)}{\partial a_{ji}} = -2\sum_{k=1}^{M}((t_k - NET_k(w)) \cdot W(R, C)) \cdot O_R \cdot \frac{(x - a_{ji})}{b_{ji}^2}; \tag{12.8}$$

$$\frac{\partial e(W)}{\partial b_{ji}} = -2\sum_{k=1}^{M}((t_k - NET_k(w)) \cdot W(R, C)) \cdot O_R \cdot \frac{(x - a_{ji})^2}{b_{ji}^3} \tag{12.9}$$

3. Після цього навчаються вихідні ваги:

$$\frac{\partial e(W^O)}{\partial W(R, C_k)} = -(t_k - NET_k(W^O))O_R; \tag{12.10}$$

$$W_k^O(n+1) = W_k^O(n) - \gamma''_{n+1}\frac{\partial e(W^O)}{\partial W(R, C_k)}. \tag{12.11}$$

4. $n := n + 1$ і переходимо до наступної ітерації.

Градієнтний метод є першим запропонованим алгоритмом навчання. Він простий у реалізації, але має такі недоліки:

- повільно збігається;
- знаходить лише локальний екстремум.

Тому було досліджено інші алгоритми навчання: а) спряжених градієнтів; б) генетичний алгоритм.

Експериментальні дослідження і аналіз мережі ЗНМ ННМ для класифікації пухлин молочної залози та медичної діагностики

Навчання можна здійснювати за двома основними сценаріями.

1. ***Вилучення ознак***. У цьому випадку останній повнозв'язний шар видаляється, а решта ЗНМ використовується як екстрактор ознак для нових наборів даних.

2. ***Додаткове налаштування***. У цьому випадку новий набір даних використовується для додаткового налаштування попередньо навченої нейронної мережі.

У дослідженні [12.21] було використано ЗНМ *VGG 16* для вилучення ознак з медичних зображень пухлин молочної залози. Як було зазначено, у цьому дослідженні автори використовували попередньо навчену ЗНМ *VGG 16*. Після цього знайдені ознаки було застосовано як вхідні дані для ННМ *NEFClass*. Як алгоритми навчання ННМ було використано: «*baisin hopping*»-алгоритм, стохастичний градієнтний спуск та диференціальну еволюцію [12.21].

Опис експериментів. Було проведено серію експериментів, а результати порівняно з роботами попередників. Результати класифікації (точність, %) з різними параметрами наведено в табл. 12.7 і 12.8 [12.21]. Усі зразки було розділено на навчальну та тестову підвибірки у співвідношенні 80/20.

У першому експерименті варіювалась кількість лінгвістичних змінних (термів) та правил для визначення найкращих значень параметрів (табл. 12.7).

З даних таблиці випливає, що, починаючи з шести нечітких множин на змінну і шести правил, точність не збільшується, але складність навчання зростає. Можна дійти висновку, що для двох класів найкращими значеннями параметрів для ННМ *NEFClass* у цій задачі є чотири нечітких множини на змінну і шість правил. Щоб оцінити ефективність гібридної ЗНМ, отри-

мані результати порівняли з результатами попередніх робіт [12.23], використовуючи різні класифікатори для тієї самої задачі (табл. 12.8).

Як видно з табл. 12.8, ННМ *NEFClass* показує кращу точність, ніж попередні класифікатори: метод опорних векторів та випадковий ліс, запропоновані в роботі [12.21].

У цьому дослідженні для навчання ННМ *NEFClass* було застосовано три алгоритми: алгоритм «*basin hopping*», стохастичний градієнтний спуск та диференціальну еволюцію. Використовуючи «*basin hopping*»-алгоритм та стохастичний градієнтний спуск, було отримано приблизно однакові результати з високою точністю, тоді як результати навчання за допомогою диференціальної еволюції виявилися набагато гіршими.

Таблиця 12.7

Результати класифікації ННМ *NEFClass*

Початкова кількість нечітких множин (лінгвістичних термів) / кількість правил	Точність класифікації, %, за збільшенням			
	40×	100×	200×	400×
2/2	73	74	74,2	73,5
4/2	75,3	74,8	75,7	75,4
6/2	78,2	79	78,4	78
8/2	76	75,4	76,5	75,8
2/4	75	74	73,8	73
4/4	78,3	76,3	75,7	75,4
6/4	82	83	82,4	83,2
8/4	82,2	81,5	81,5	83,8
2/6	75,4	73,8	74,4	73,2
4/6	90	91	90,5	90
6/6	89	89,7	90,2	89,5
8/6	90,3	90,5	92	91,2
4/8	89,3	89,8	89,7	89,3
6/8	89,2	88	89,4	88,4
8/8	88	87,2	87,2	87

Таблиця 12.8

Порівняння точності різних класифікаторів

Класифікатор	Точність класифікації, %, за збільшенням			
	40×	100×	200×	400×
Linear SVM	89	89	88	88
Polynomial SVM	88	90	89	85
Random forest	89,18	88	87,74	80
ННМ NEFClass	90	91	90,5	90

Варто зазначити, що в цій задачі кількість ознак, вилучених ЗНМ *VGG 16*, була дуже великою — 4096. Тому було вирішено скоротити кількість ознак і зменшити розмірність задачі класифікації. Для цього було застосовано метод головних компонент [12.21]. Результати такого скорочення наведено в табл. 12.9.

З табл. 12.9 випливає, що скорочення головних компонент до 250 є найбільш прийнятним, оскільки складність навчання збільшується приблизно пропорційно розмірності вхідних даних. Наступні експерименти було виконано з використанням даних з фактором збільшення 100× (2081 зображення) [12.14, 12.21].

Точність класифікації з використанням 250 ознак з різними параметрами *NEFClass* — кількістю правил та нечітких множин — наведено в табл. 12.10.

У табл. 12.11 наведено залежність точності класифікації від кількості ознак. З даних таблиці видно, що точність знизилася лише на 4—7 % після зменшення кількості ознак, але водночас тривалість навчання істотно скоротилась.

Таблиця 12.9

Залежність сумарної дисперсії від кількості компонент і часу навчання

Кількість головних компонент	Дисперсія (нормована)	Приблизний час навчання, год
100	0,84058	~2
200	0,89736	~3
250	0,91232	~4
500	0,95486	~9

Таблиця 12.10

Точність класифікації, %, з 250 ознаками

Кількість нечітких множин / кількість правил	Фактор збільшення 100×
4/4	80,64
4/6	87,24
4/8	88,18

Таблиця 12.11

Точність класифікації, %, з різною кількістю ознак

Кількість нечітких множин / кількість правил	Кількість ознак		
	100	250	4096
4/4	75,23	80,64	86,3
4/6	83,34	87,24	91
4/8	84,21	88,18	89,8

Можна також легко побачити, що точність знижується зі зменшенням кількості ознак, але це зниження на 3—5 % є незначним, якщо порівняти результати з 100 та 250 ознаками. Результати класифікації було порівняно з результатами досліджень з повним набором ознак (4096 ознак) і виявлено, що зі зменшенням кількості ознак у 20 разів точність у середньому знижується лише на 3—6 % [12.21]. Цей висновок підтверджує ефективність застосування методу градієнтного спуску для зменшення розмірності завдань класифікації медичних зображень.

Отже, було розглянуто і досліджено проблему аналізу медичних зображень пухлин молочної залози та класифікації виявлених пухлин на два класи — доброякісні та злоякісні.

Для розпізнавання зразків пухлин молочної залози запропоновано нову гібридну мережу ЗНМ-ННМ, в якій ЗНМ *VGG 16* використовується для вилучення інформативних ознак, а ННМ *NEFClass* — для класифікації виявлених пухлин.

Для навчання ННМ *NEFClass* було застосовано «*basin hopping*»-алгоритм, алгоритм стохастичного градієнтного спуску та диференціальної еволюції.

Проведено експериментальні дослідження запропонованої гібридної мережі ЗНМ-ННМ у завданні класифікації реальних зображень пухлин молочної залози з використанням набору даних *BreakHis*.

Порівняно точність класифікації запропонованої гібридної мережі ЗНМ-ННМ з відомими результатами, отриманими із залученням широко відомих алгоритмів класифікації: методу опорних векторів та випадкового лісу, що підтвердило ефективність застосування запропонованої гібридної мережі ЗНМ-ННМ [12.21].

Розглянуто проблему зменшення кількості ознак у завданні класифікації медичних зображень з використанням методу головних компонент і досліджено його ефективність.

12.7. Гібридна згорткова нейронна мережа на основі автокодувальника для виявлення раку молочної залози

Аналіз застосувань ЗНМ у задачах обробки медичних зображень молочної залози та діагностики раку

У попередніх підрозділах для задачі класифікації раку молочної залози було розглянуто і досліджено різні ЗНМ стандартної архітектури. Аналіз експериментальних результатів підтвердив високу точність класифікації декількох ЗНМ та перспективи їх застосування . Однак основним недоліком

розглянутих ЗНМ є великі обчислювальні витрати та тривалий час навчання. Крім того, основна увага в цих роботах приділялася оцінці критерію точності класифікації [12.20—12.23]. Але у вирішенні задачі класифікації раку є ще один важливий критерій — це повнота виявлення (*recall*), що оцінює частку правильно виявлених випадків раку, тобто чутливість [12.24].

Метою цього підрозділу є розгляд нової архітектури ЗНМ, яка мінімізує час навчання і має високе значення критерію чутливості для виявлення раку молочної залози.

У багатьох дослідженнях для виявлення раку молочної залози використовують ЗНМ. Однак більшість з них оцінюють лише точність класифікації раку, тоді як цей критерій для будь-якого виявлення раку не є єдиним важливим фактором, який слід враховувати [12.24]. У цій проблемі слід розглядати чутливість моделей, щоб зрозуміти, скільки разів модель неправильно класифікувала рак. Автори [12.25] запропонували передову згорткову нейронну мережу (*DenseNet*) для виявлення раку молочної залози з використанням зображень гістології цієї пухлини (*BACH*) з точністю 85,6 %. Частота неправильної класифікації для класу раку в середньому становила 14,4 %. У своїй роботі автори порівнювали чутливість (у середньому для чотирьох класів) *ResNet 50* із запропонованою ними ЗНМ, отримавши 76 та 79 % відповідно.

Після порівняння результатів, отриманих з використанням моделей ЗНМ (*VGG 16, VGG 19, Xception, ResNet, Inception*), а саме 80 % правильно визначених прикладів для багатокласової класифікації, автори запропонували модель, де точність у середньому становила 83,97 % для двох класів (доброякісного та злоякісного). Запропонована модель була комбінацією *Inception* та *ResNet* з використанням набору даних *BreakHis*, який містить 7 909 мамографічних сканувань із чотирма факторами збільшення (40×, 100×, 200× та 400×) [12.26].

У роботі [12.27] зазначено, що завдяки архітектурі *DenseNet*, в якій всі шари повністю з'єднані з кожним попереднім шаром і з коротким зв'язком між цими шарами поблизу входу та виходу, модель могла навчатися ефективніше і точніше.

У роботі [12.28] запропоновано мережу *DenseNet* із часом навчання 11 годин. Автори стверджували, що спочатку використовували ваги з *Imagenet* і додатково налаштовували модель для навчання *DenseNet*. Усі згорткові частини мережі було заморожено, але повнозв'язний шар можна було навчати. Автори [12.29] використовували атрусну (зі зміщенням) *DenseNet*, яка досягає багатомасштабного вилучення ознак через інтеграцію атрусних згорток до щільного блока. Автори [12.29] порівнювали два набори даних, *BACH* і *CCG*, де середня точність класифікації для запропонованої моделі становила 82,50 та 87,05 % відповідно для кожного набору даних.

У роботі [12.30] запропоновано нову модель згорткової нейронної мережі, де автори використали 400 зображень зі збільшенням 40× для навчальних даних і 200 даних для валідації. У [12.30] оцінено три різні архітектури *ConvNet*:

1) тришарову;

2) чотиришарову;

3) шестишарову.

Тришарова *ConvNet* містила один згортковий шар, один шар пулінгу та один повнозв'язний шар. Чотиришарова мала два згорткових шари, два шари пулінгу, а останній шар був повнозв'язним. Шестишарова архітектура *ConvNet* складалася з чотирьох згорткових і пулінгових шарів з 16 вузлами та повнозв'язного шару. Згідно з результатами, глибші архітектури показали кращий результат з точністю 1,06 %.

У роботі [12.31] автори запропонували напівконтрольоване навчання з використанням згорткових нейронних мереж. Точність розробленої моделі становила 82,43 %, а площа під кривою (*AUC*) — 88,18 %. Під час експериментів використовувалося 1874 приклади мамографічних зображень. Крім того, автори розробили три методи зважування даних з використанням експоненційної функції, гауссової функції та функції Лапласа. Згідно з результатами [12.31] порівняння з двома іншими рівняннями зважування, експоненційна функція продемонструвала кращі результати і показала точність 82,43 %, чутливість 81,00 % та специфічність 72,26 % відповідно.

Згортковий автокодувальник як спосіб зменшення розмірності зображень

Як було зазначено вище, головним недоліком застосування ЗНМ для оброблення медичних зображень та класифікації раку є великі обчислювальні витрати та значна тривалість навчання. Щоб скоротити час навчання у завданні оброблення медичних зображень, потрібно зменшити розмірність зображень і стиснути їх. Для цього дуже перспективним видається застосування автокодувальника, метою якого є навчитися стиснутому розподіленому поданню заданих даних, зазвичай для зменшення розмірності. З іншого боку, для того самого завдання (зменшення розмірності) є метод головних компонент. Однак використання автокодувальника має низку переваг.

1. Автокодувальник може навчатися нелінійних перетворень з нелінійними функціями активації та множинними шарами, на відміну від методу головних компонент.

2. Йому не потрібно навчати повнозв'язні шари, тому він може використовувати згорткові шари для навчання.

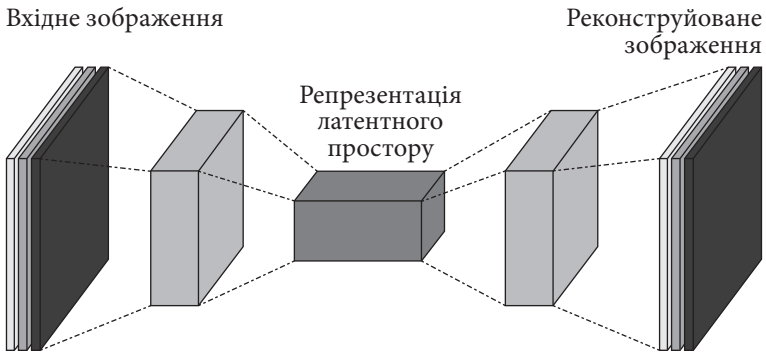


Рис. 12.10. Архітектура автокодувальника

3. Може бути ефективнішим з точки зору параметрів моделі для навчання декількох шарів з автокодувальником.

4. Дає зображення як вихід кожного шару, і наявність кількох зображень різної розмірності є практичною.

Однією з причин застосування згорткового автокодувальника під час дослідження є те, що знайти достатньо великі набори даних з мітками дуже складно, а автокодувальник — це неконтрольована модель, якій не потрібно, щоб набір даних був розмічений. Ще однією перевагою автокодувальника є його здатність робити модель меншого розміру. Відповідно, модель матиме менше параметрів і, як результат, час обчислень і навчання істотно зменшиться. Наприклад, у *DenseNet* загалом є 58 420 802 параметри, і 7 037 504 з них не підлягають навчанню. Однак у запропонованому згортковому автокодувальнику є 2 940 865 параметрів і лише 3 840 з них не підлягають навчанню.

Архітектуру автокодувальника ілюструє рис. 12.10. В автокодувальнику є шари між входом і виходом, а розміри цих шарів менші ніж шару входу. Наприклад, якщо вхідний вектор має розмірність *N*, зазвичай це означає, що вихід також матиме розмірність *N*. Вхідні дані проходять через шар розміром *P*, де значення *P* менше за *N*. Автокодувальник отримує немарковані вхідні дані, які потім кодуються для відновлення вхідних даних. Важливою частиною автокодувальника є вузьке місце. Вузьке місце (*bottleneck*) — це підхід до навчання презентаціям.

У роботі [12.32] під час експерименту було використано кілька архітектур згорткового автокодувальника. Згортковий автокодувальник було модифіковано з 18 шарами кодування та 14 шарами декодування. Між згортковими шарами використовували нормалізацію пакетів. Запропонований згортковий автокодувальник було навчено так, щоб модель вилучала інформативні ознаки у процесі кодування, а декодер міг потім відновити оригінальне вхідне зображення автокодувальника. Модель могла відновлювати оригінальне зображення, навіть якщо на ньому були наявні певні

шуми. Порівняння вхідних і відновлених зображень показано на рис. 12.11. Після створення успішної моделі автокодувальника вихід автокодувальника буде використано з повнозв'язним шаром для створення повної моделі (згорткового автокодувальника).

Рис. 12.11, *а* ілюструє зразок тестових даних з доданим шумом, а на рис. 12.11, *б* показано результат роботи згорткового автокодувальника. Точність відтворення тестових даних для згорткового автокодувальника становила 79,38 %. Було розроблено нову архітектуру гібридної ЗНМ на основі згорткового автокодувальника [12.32] (рис. 12.12). Вона складається з двох частин: автокодувальника, що стискає початкове зображення і видобуває первинні ознаки, і ЗНМ (*DenseNet*), що містить згорткові шари, шари пулінгу, один повнозв'язний шар і вихідний шар з двома нейронами для класифікації.

Після побудови архітектури гібридної ЗНМ наступною проблемою було навчання мережі. Спочатку було додатково налаштовано модель *DenseNet*, використовуючи ваги *Imagenet*, попередньо оброблено вхідні дані та обрано відповідні гіперпараметри (наприклад, швидкість навчання, розмір пакета, кількість епох тощо) для навчання моделі.

Для навчання автокодувальника потрібно було встановити чотири параметри. Перший — це розмір коду, який визначає кількість вузлів у середньому шарі, менший розмір призводить до більшого стискання. Другий параметр — кількість шарів, автокодувальник може бути настільки глибоким, наскільки потрібно. Наступним параметром є функція втрат. Останній параметр — кількість вузлів на шар. Кількість вузлів на шар зменшується в кожному наступному шарі автокодувальника і збільшується у декодері. Також декодер є симетричним автокодувальнику з точки зору структури шарів.

Для навчання *DenseNet* використовували оптимізатор *Adam* зі швидкістю навчання 0,001, тоді як у згортковому автокодувальнику кращий результат показав *RMSprop* (*lr* = 0,001) [12.1, 12.32].

Усі скани були попередньо оброблені перед використанням для навчання моделі за допомогою змінення розміру, нормалізації та методів зменшення розмірності. У цій роботі всі експерименти було виконано з використанням *Jupyter Labs, Tensorflow* 2 та *Python* 3. Програми було реалізовано на віртуальній машині з *NVIDIA Tesla GPU* та вісьмома процесорами *Intel CPU*.

Зміни функції втрат для даних навчання та валідації проілюстровано на рис. 12.13. Експерименти також показали, що кількість епох 25 дала кращий результат порівняно з великою кількістю епох, наприклад 100 або 200. Чим більше епох, тим вища ймовірність того, що модель буде перенавчатися.

Згідно з рис. 12.13, краще використовувати зменшений коефіцієнт навчання для поточної моделі. Проте цей параметр може відрізнятися для інших моделей.

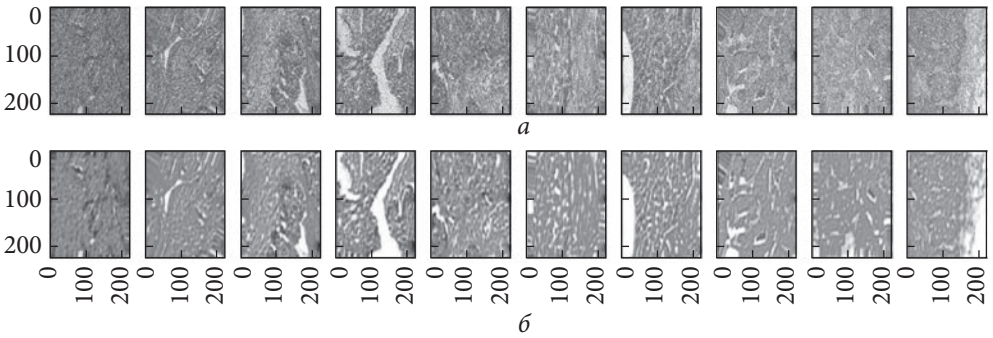


Рис. 12.11. Порівняння вхідних і відновлених зображень: *а* — зашумлені скани: до сканів додано шум; *б* — відновлені (оригінальні) тестові дані

Рис. 12.12. Архітектура гібридної ЗНМ на основі автокодувальника

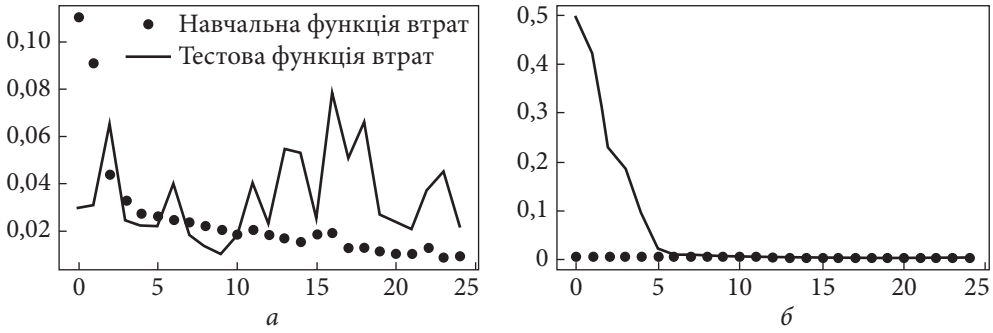


Рис. 12.13. Порівняння втрат для даних навчання та валідації з коефіцієнтами навчання 0,01 (*а*) та 0,001 (*б*)

Для визначення впливу розміру набору даних на точність класифікації було проведено експериментальні дослідження, результати яких наведено на рис. 12.14 [12.1, 12.32].

Експериментом доведено, що для навчання згорткового автокодувальника з нуля не обов'язково мати великі набори даних. Натомість *DenseNet* та *Inception V3* потребують великих наборів даних, щоб навчатися з нуля. Отже, згортковий автокодувальник має спрощену модель, а час навчання значно скорочується порівняно з *DenseNet* або *Inception V3*. На продуктивність моделі значно впливають також зовнішній вигляд і якість вхідних даних. Для порівняння використовували набори даних *BreakHis* та *Breast histology*.

Продуктивність моделі з різними вхідними даними наведено в табл. 12.12 [12.1, 12.32].

Як уже зазначалося, для аналізу якості класифікації раку точність не є достатнім показником, оскільки необхідно оцінити чутливість, повноту (*recall*) моделі. Результати застосування згорткового автокодувальника на основі ЗНМ за різними показниками (критеріями) наведено в табл. 12.13 [12.1, 12.32].

Таблиця 12.12

Точність класифікації згорткового автокодувальника для різних наборів даних

Критерій	Точність класифікації, %, для набору даних	
	BreakHis	Гістологія молочної залози
Правильність	93	90,5
Точність	93,2	91,6
Повнота	93,5	92,40
F1-оцінка	93,3	92

Таблиця 12.13

Результати розпізнавання раку молочної залози із застосуванням згорткового автокодувальника

Клас	Критерій			
	Точність, %	Повнота, %	F1-оцінка, %	Підтримка
Клас 0	90	95	93	747
Клас 1	95	90	92	1626
Макро середнє	93	93	92	2373
Зважене середнє	93	93	92	2373

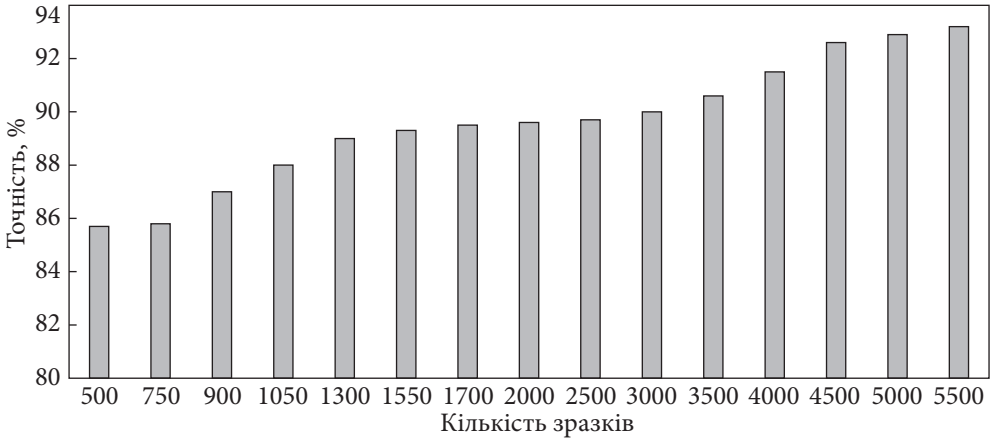


Рис. 12.14. Точність згорткового автокодувальника залежно від розміру набору даних

Аналізуючи результати, можна дійти висновку, що критерій повноти для згорткового автокодувальника кращий, ніж для ЗНМ *Inception V3*. Варто зазначити, що лише 5 % із класу «0» (рак) було класифіковано неправильно (хибно негативні).

У наступних експериментах було проведено порівняльне дослідження класифікації раку молочної залози за допомогою розробленої гібридної ЗНМ (на основі автокодувальника) та стандартної ЗНМ [12.1, 12.32]. Результати подано в табл. 12.14.

Згідно з даними табл. 12.14, гібридна згорткова мережа на основі автокодувальника показала кращі результати і найменшу тривалість навчання порівняно з альтернативними моделями ЗНМ *Inception V3* та *DenseNet* 121.

Таблиця 12.14

Порівняльний аналіз різних ЗНМ у завданні класифікації раку молочної залози

Критерій	Модель ЗНМ		
	Inception V3	*DenseNet 121*	*Hybrid CNN*
Точність, %	66,66	91,6	93,2
Повнота, %	85,84	90,50	93,5
F1-оцінка, %	74,99	91	93,3
Правильність, %	74,78	89,67	93
Час, год	~27	~24	~13

Висновки

У цьому розділі досліджено ефективність згорткових нейронних мереж різної архітектури у вирішенні завдання оброблення медичних зображень та класифікації пухлин молочної залози. Для цієї мети було використано спеціально розроблений набір даних *BreakHis.* Оскільки набір має обмежений розмір і навчати моделі на оригінальних зображеннях через велику кількість інформації в них неможливо, було підготовлено дані і згенеровано набір зображень розміром 227 × 227. Для підвищення збіжності моделей використовували метод зменшення параметра швидкості навчання зі збільшенням номера ітерації.

Вирішено завдання бінарної класифікації зображень. Точність бінарної класифікації пухлин на тестовому наборі зображень з використанням моделі *Xception* досягла 98,81 % (найвища). Додатково було досліджено завдання багатокласової класифікації пухлин молочної залози. З'ясовано, що моделям *VGG 16* і *VGG 19* потрібно набагато більше ітерацій для навчання, ніж *ResNet 50 V2, Inception* та *Xception.*

Проведено експериментальні порівняння запропонованих моделей і методів класифікації раку молочної залози з відомими роботами. Найкращою виявилася модель *Xception* з використанням оригінального класифікатора, точність багатокритеріальної класифікації якої становила 90,8 %.

Для розпізнавання зразків пухлин молочної залози запропоновано також нову гібридну мережу ЗНМ-ННМ, в якій ЗНМ *VGG 16* використовується для вилучення інформативних ознак, а ННМ *NEFClass* — для класифікації виявлених пухлин.

Проведено експериментальні дослідження запропонованої гібридної мережі ЗНМ-ННМ у завданні класифікації реальних зображень пухлин молочної залози з використанням набору даних *BreakHis*.

Аналіз експериментальних результатів підтвердив високу точність класифікації кількох ЗНМ та перспективи їх застосування. Однак у відомих роботах, присвячених класифікації пухлин, основну увагу приділено оцінці точності класифікації. Проте є ще один важливий критерій для класифікації раку — це повнота (*recall*), яка оцінює частку правильно виявлених випадків раку, тобто чутливість.

Отже, було розроблено нову гібридну згорткову мережу, призначену для підвищення критерію повноти з одночасним скороченням часу навчання. Гібридна ЗНМ використовує згортковий автокодувальник для зменшення розмірності вхідного зображення та вилучення інформативних ознак. Згортковий автокодувальник має меншу кількість параметрів порівняно з *DenseNet*, у результаті чого модель стає менш складною, що запобігає перенавчанню. Крім того, автокодувальник є неконтрольованою моделлю і не потребує розмічених даних. Було проведено експериментальні дослі-

дження розробленої гібридної ЗНМ для розпізнавання раку молочної залози і отримано такі значення критеріїв: чутливість, точність, F1-оцінка та точність моделі становлять 93,5, 93,2, 93,3 та 93 % відповідно.

Загалом гібридна згорткова мережа на основі автокодувальника показала кращі результати і найменшу тривалість навчання порівняно з альтернативними моделями ЗНМ *Inception V3* та *DenseNet* 121.

СПИСОК ЛІТЕРАТУРИ

12.1. Zaychenko Yu., Hamidov G., Chapaliuk B. Medical Images Processing and Cancer Classification in the Problem of Diagnostics. Cambridge Scholars Publishing, UK, 2023. 120 p.

12.2. Siegel R.L., Miller K.D., Jemal A. Cancer statistics, 2019. CA: *A Cancer Journal for Clinicians*. 2019. Vol. 69(1). P. 7—34.

12.3. Grunfeld E. et al. Family caregiver burden: results of a longitudinal study of breast cancer patients and their principal caregivers. *CMAJ*. 2004. Vol. 170(12). P. 1795—1801.

12.4. Kale H.P., Carroll N.V. Self-reported financial burden of cancer care and its effect on physical and mental health-related quality of life among US cancer survivors. *Cancer*. 2016. Vol. 122(8). P. 283—289.

12.5. Ryan S. The costs of breast cancer in the U.S. Costs of Care, 2015. URL: https://costsofcare.org/the-costs-of-breast-cancer-in-the-u-s/.

12.6. Zhang Y., Zhang B., Coenen F., Lu W. Breast cancer diagnosis from biopsy images with highly reliable random subspace classifier ensembles. *Machine Vision and Applications*. 2013. Vol. 24 (7). P. 1405—1420.

12.7. Zhang Y., Zhang B., Coenen F., Lu W. One-class kernel subspace ensemble for medical image classification. *EURASIP Journal on Advances in Signal Processing*. 2014. Vol. 2014 (17). P. 1—13.

12.8. Doyle S., Agner S., Madabhushi A., Feldman M., Tomaszewski J. Automated grading of breast cancer histopathology using spectral clustering with textural and architectural image features. *Proceedings of the 5th IEEE International Symposium on Biomedical Imaging (ISBI): From Nano to Macro*. IEEE, May 2008. Vol. 61. P. 496—499.

12.9. Bengio Y., Courville A., Vincent P. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2013. Vol. 35. P. 1798—1828.

12.10. Singh A., Mansourifar H., Bilgrami H., Makkar N., Shah T. Classifying Biological Images Using Pre-trained CNNs. URL: https://docs.google.com/document/d/1H7xVK7nwXcv11CYh7hl5F6pM0m218FQloAXQODP-Hsg/edit?usp=sharing

12.11. LeCun Y., Bengio Y., Hinton G. Deep learning. *Nature*. 2015. Vol. 521. P. 436—444.

12.12. Krizhevsky A., Sutskever I., Hinton G. E. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems.* 2012. Vol. 25. P. 1097—1105.

12.13. Micholucci U. Applied Deep Learning: A Case-Based Approach to Understanding Deep Neural Networks. Springer Nature, 2018. 431 p.

12.14. Kayacan E., Khanesar M.A., Kaynak O. Fuzzy Neural Networks for Real Time Control Applications. Butterworth-Heinemann, 2015. 258 p.

12.15. Giancarlo Z. Deep Learning with TensorFlow. Packt Publishing, 2017. 316 p.

12.16. Zaychenko Y., Hamidov G., Zaichenko H. Investigation of Convolutional Neural Networks in the Tasks of Medical Images Analysis and Classification of Breast Tumors. *14th*

International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI 2021). 2021, 23—25 October. Shanghai, China. P. 1—6.

12.17. Nauck D., Kruse R. New learning strategies for NEFCLASS. *Proc. Seventh International Fuzzy Systems Association World Congress IFSA'97*. 1997. IV. P. 50—55.

12.18. Zgurovsky M., Zaychenko Yu. The Fundamentals of Computational Intelligence: System Approach. Springer International Publishing AG, Switzerland, 2016. 375 p.

12.19. Zaychenko Yu.P., Petrosyuk I.M., Jaroshenko M.S. The investigations of fuzzy neural networks in the problems of electro-optical images recognition. *System Research and Information Technologies*. 2009. Vol. 4. P. 61—76.

12.20. Zaychenko Y., Huskova V. Recognition of objects on optical images in medical diagnostics using fuzzy neural network NEF class. *International Journal Information Models and Analysis.* 2015. Vol. 4 (1). P. 13—22.

12.21. Zaychenko Yu., Hamidov G., Varga I. Medical images of breast tumors diagnostics with application of hybrid CNN —FNN network. *System research & Information technologies.* 2018. No. 4. P. 37—47.

12.22. Chollet F. Xception: Deep Learning with Depthwise Separable Convolutions. URL: https://openaccess.thecvf.com/content_cvpr_2017/papers/Chollet_Xception_Deep_Learning_CVPR_2017_paper.pdf.

12.23. Lin C.-T., Yeh C.-M., Liang S.-F., Chung J.-F., Kumar N. Support-Vector-Based Fuzzy Neural Network for Pattern Classification. *IEEE Transactions on Fuzzy Systems*. 2006. Vol. 14. P. 31—41.

12.24. Olson B., Hashmi I., MolloyK., Shehu A. Basin Hopping as a General and versatile optimization framework for the characterization of biological macromolecules. *Advances in Artificial Intelligence.* 2012. No. 3. P. 115—122.

12.25. Araujo T., Aresta G., Castro E., Rouco J., Aguiar P., Eloy C. et al. Classification of breast cancer histology images using Convolutional Neural Networks. *PLoS ONE*. 2017. Vol. 12. No. 6. e0177544. https://doi.org/10.1371/journal.pone.0177544

12.26. BreakHist-Dataset-Image-Classification, URL: https://github.com/Anki0909/BreakHist-Dataset-Image-Classification

12.27. Huang G., Liu Z., Weinberger Q. K. Densely Connected Convolutional Networks. ar Xiv: https:// arXiv:1608.06993v3. 2018

12.28. Nawaz M., Sewissy A. A., Soliman H. T. A. Multi-Class Breast Cancer Classification using Deep Learning Convolutional Neural Network. *(IJACSA) International Journal of Advanced Computer Science and Applications*. 2018. Vol. 9. No. 6.

12.29. Li Yu., Xie X., Shen L., Liu S. Reversed Active Learning based Atrous DenseNet for Pathological Image Classification. ar Xiv: https:// arXiv:1807.02420. 2018

12.30. Cruz-Roa A., Gilmore H., Basavanhally A. et al. Accurate and reproducible invasive breast cancer detection in whole-slide images: A Deep Learning approach for quantifying tumor extent. *Sci Rep*. 2017. 7. 46450 https:// doi:10.1038/srep46450

12.31. Schmidhuber J. Deep Learning in Neural Networks: An Overview. Neural Networks. 2015. Vol. 61. P. 85—117. https://doi.org/10.1016/j.neunet.2014.09.003

12.32. Naderan M., Zaychenko Yu., Napoli A. Using convolutional neural networks for breast cancer diagnosing. *System Research and Information Technologies*. 2019. No. 4.

12.33. Zaychenko Yu., Naderan M., Hamidov G. Hybrid convolution network for medical images processing and breast cancer detection. *System research* & *Information technologies.* 2022. No. 2. P. 85—93. https://doi.org/10.20535/SRIT.2308-8893.2022.2.0

РОЗДІЛ 13

СИСТЕМА ВИЯВЛЕННЯ РАКУ ЛЕГЕНІВ НА ОСНОВІ СТРАТЕГІЇ ГЛИБОКОГО НАВЧАННЯ

13.1. Двовимірні згорткові нейронні мережі
із багаторазовим навчанням

13.2. Рекурентні нейронні мережі
з механізмом уваги та 2D-ЗНМ

13.3. Тривимірна згорткова нейронна мережа
для виявлення раку легенів

13.4. Набори даних та їх попереднє оброблення

13.5. Експериментальні дослідження і аналіз результатів

Висновки

Список літератури

Рак легенів — один з найагресивніших видів раку, і його виявлення на ранній стадії дає можливість врятувати багато життів пацієнтів у всьому світі. Автоматизована система виявлення раку легенів може прискорити процес його виявлення і, отже, допомогти врятувати людські життя. У цьому розділі розглянуто три підходи до побудови та проєктування систем виявлення раку легенів. Ці підходи передбачають використання двовимірних згорткових нейронних мереж (ЗНМ) із завданнями багаторазового навчання, двовимірних ЗНМ з рекурентними нейронними мережами або конвеєрів систем виявлення з тривимірними ЗНМ, які можуть складатися з одного або декількох етапів. Наведено результати експериментів для кожного підходу, які було порівняно з результатами, викладеними в інших новітніх публікаціях, щоб з'ясувати, яка стратегія побудови системи виявлення раку легенів є найефективнішою.

Основна мета цього розділу — розглянути архітектуру автоматизованої системи виявлення раку легенів і дослідити, як архітектура нейронної мережі та дизайн конвеєра системи впливають на загальну точність класифікації.

13.1. Двовимірні згорткові нейронні мережі із багаторазовим навчанням

Для скринінгу раку легенів використовується КТ грудної клітки людини, яка є тривимірним зображенням з високою роздільною здатністю. Це зображення використовується як вхідні дані для системи виявлення раку легенів, що зазвичай будується за допомогою штучних нейронних мереж. Цей факт накладає обмеження на пам'ять і обчислювальні можливості системи виявлення, оскільки весь КТ-знімок не може вміститися в пам'яті відеокарти для досить великої моделі, а обчислення 3D-даних і використання пам'яті зростають у кубічній прогресії.

Перша і найпростіша стратегія побудови наскрізної системи виявлення раку легенів полягає в обробленні кожного зрізу КТ-сканування окремо, а потім в об'єднанні результатів за допомогою попередньо визначених правил. Кожен зріз КТ може бути анотовано, а мітки результатів можна використовувати для розгляду кожного зрізу окремо. У такому разі система

виявлення працює з 2D-даними, які займають набагато менше пам'яті, ніж загальне 3D-зображення пацієнта, створене зі зрізів КТ-скану [13.1]. Крім того, така стратегія дає можливість повторно використовувати найпопулярніші попередньо навчені архітектури 2D-згорткових нейронних мереж (ЗНМ), такі як *ResNet* [13.2], *DenseNet* або *Xception* [13.3], щоб пришвидшити процес навчання та підвищити загальну точність системи.

Однак набір даних *Data Science Bowl* 2017 (*DSB* 2017), який широко використовується в цьому дослідженні, має лише одну мітку на весь КТ-знімок пацієнта, що робить такий спрощений підхід непрактичним. Тому краще розглядати проблему як багатоекземплярну задачу, де екземпляром є один зі зрізів на КТ-скані. Остаточне рішення щодо прогнозу наявності раку можна прийняти, використовуючи загальне припущення: за умови наявності позитивного екземпляра позитивним є і весь пакет. Аналогічно, увесь пакет є негативним, якщо всі його екземпляри негативні.

Під час роботи з двовимірними даними вхідні зображення зазвичай розбивають на ділянки, що перекриваються, і розглядають їх як незалежні екземпляри. Це дає змогу розглядати завдання класифікації двовимірних зображень як *MIL*-задачу [13.4]. Наприклад, саме в такий спосіб у роботі [13.5] розв'язується задача класифікації мамограми. Більш того, щоб повністю дослідити можливості глибокого мультикласового навчання (*MIL*), її автори перетворюють традиційний підхід *MIL*, про який ідеться вище, на задачу призначення розріджених міток, де розрідженість пояснюється тим, що злоякісна тканина може бути наявна лише на невеликій частині вхідних ділянок зображення. Можна застосувати той самий підхід для КТ легенів, де кожний шар сканування є екземпляром у пакеті, якому потрібно присвоїти мітку. Як і в задачі класифікації двовимірної мамограми, рак легенів локалізований лише на невеликій частині зображень КТ; отже, вирішили повторно використати цей розріджений *MIL*-підхід до задачі класифікації раку легенів на зображеннях КТ і перевірити, наскільки добре він буде працювати на тривимірних даних.

Для формалізації задачі припустимо, що у нас є вхідний знімок пацієнта S, який містить k зображень. Мета моделі — передбачити, чи містить КТ-знімок пацієнта злоякісний легеневий вузол, чи ні. З огляду на це, а також за припущення, що модель розглядає весь КТ-знімок, розв'язуємо задачу бінарної класифікації. Для кожного зрізу КТ вихідний прогноз раку можна записати у вигляді

$$r_n = \sigma(w \cdot CNN(k_m) + b), \tag{13.1}$$

де σ — сигмоїдна функція; w — ваговий коефіцієнт логістичної регресії; *CNN* — функція, яка означає прямий прохід згорткової нейронної мережі; k_m —зображення на скані пацієнта в позиції m; b — зсув.

Ураховуючи ймовірність раку для кожного зображення в скануванні пацієнта, загальну ймовірність наявності раку можна записати так:

$$p(y = 1 \mid S,k) = \max\{r_1, r_2, \ldots, r_n\}; \tag{13.2}$$

$$p(y = 0 \mid S,k) = 1 - \max\{r_1, r_2, \ldots, r_n\}, \tag{13.3}$$

де y — істинна мітка з набору даних.

Отже, функцію втрат для моделі *MIL* можна записати як

$$L = \frac{1}{N}\sum_{n=1}^{N}(-\log(p(y_n \mid S, k)) + \mu||r_n'||_1), \tag{13.4}$$

де N — кількість знімків пацієнта; μ — коефіцієнт розрідженості, який є компромісом між важливістю зображення та припущенням про розрідженість раку; $||r_n'||_1$ —L_1 норма вихідного вектора кожного скану зображення.

Отримані втрати використовують для навчання двовимірної згорткової нейронної мережі. Для цього в роботі [13.3] мережу *2D-DenseNet*, попередньо навчену на наборі даних *ImageNet*, було використано для ініціалізації нейронної мережі на початку процесу навчання. Під час навчання вихід *DenseNet* для кожного зрізу КТ-скану накопичувався і передавався у функцію втрат з істинною міткою. В кінцевому підсумку такий підхід дає лише 0,62 точності. Низьке значення точності можна пояснити насамперед тим, що ЗНМ втрачає тривимірну природу інформації, яка є важливою для розуміння злоякісності легеневих вузликів.

13.2. Рекурентні нейронні мережі з механізмом уваги та 2D-ЗНМ

Щоб пом'якшити проблему втрати просторової інформації в попередньому підході, можна взяти за приклад досвід реальних рентгенологів, які визначають, чи є пухлина або підозрілі аномальні легеневі вузлики на КТ-скані, чи ні. Зазвичай рентгенолог досліджує зрізи КТ один за одним, ретельно комбінуючи інформацію з сусідніх зрізів, щоб зрозуміти, чи є легеневий вузлик злоякісним. Можна застосувати такий самий підхід, використовуючи рекурентну нейронну мережу (РНМ), яка є класичною нейронною мережею для роботи з послідовностями. Цю рекурентну нейронну мережу можна використати для послідовного аналізу всіх шарів сканування, щоб у кінці повернути ймовірність наявності раку. У наступних підрозділах обговоримо, який тип рекурентної нейронної мережі потрібен для розв'язання задачі виявлення раку легенів на КТ-знімках. Надамо детальне пояснення такого архітектурного вибору.

Рекурентні нейронні мережі

Рекурентні нейронні мережі (РНМ) — це сімейство штучних нейронних мереж, створених спеціально для оброблення довгих послідовностей даних змінної довжини. На відміну від інших традиційних нейронних мереж РНМ не розглядають кожний вхід як щось нове і мають властивість пам'яті, яка дає їм можливість пов'язувати інформацію між різними елементами послідовності, навіть коли вони перебувають далеко один від одного. Ця здатність дає змогу вирішити широкий спектр завдань, що потребують історичної інформації для розуміння поточних елементів. Наприклад, для розуміння розділу книги часто стає в пригоді згадування про зміст її попередніх розділів.

Здатність рекурентних нейронних мереж послідовно працювати з довгостроковими залежностями досягається завдяки додаванню циклічних зв'язків між нейронами нейронної мережі. Інакше кажучи, один шар нейронної мережі має зв'язок зі станом мережі на попередньому часовому кроці. Рекурентні стани нейронної мережі, які зберігаються на кожному часовому кроці послідовності, називаються прихованими станами. Математично їх можна описати як

$$h^{(t)} = f(h^{(t-1)}, x^{(t)}; W), \tag{13.5}$$

де t — поточний часовий крок послідовності в момент часу t; $h^{(t-1)}$ — прихований стан РНМ для попереднього кроку в момент часу $t - 1$; h^t — прихований стан РНМ, обчислений для поточного моменту часу з кроком t; $x^{(t)}$ — дані, передані на вхід РНМ у момент часу t; W — ваги нейронної мережі, що означають зв'язок між прихованими станами РНМ; f — функція, яка відображує прихований стан у момент часу t на стан у момент часу $t + 1$.

З рівняння (13.5) видно, що РНМ має здатність запам'ятовувати вхідну інформацію x з попереднього кроку у вигляді деякого прихованого стану h. Коли ця мережа вчиться передбачати наступний елемент, вона вчиться використовувати і свій прихований стан для узагальнення всієї інформації, яку вона бачила на попередніх часових кроках. Інакше кажучи, приховані стани РНМ можна вважати узагальненням історичної інформації, що надає найцінніші деталі, але втрачає деякі не пов'язані або неважливі деталі для прогнозування наступного часового кроку.

РНМ може мати різну кількість прихованих станів залежно від розміру вхідної послідовності. Часто рекурентну нейронну мережу недоцільно зображувати у вигляді рекурентної функції, оскільки нейронна мережа має динамічну структуру і незрозуміло, наскільки вона глибока. У такому разі рекурентну нейронну мережу зображують у вигляді розгорнутого графа (рис. 13.1).

Розгорнуте графове подання для РНМ має декілька основних переваг [13.7].

1. РНМ не залежить від довжини вхідної послідовності і завжди працює з фіксованими розмірами вхідних даних. Це відбувається тому, що структура нейронної мережі визначається в термінах переходу з одного стану в інший.

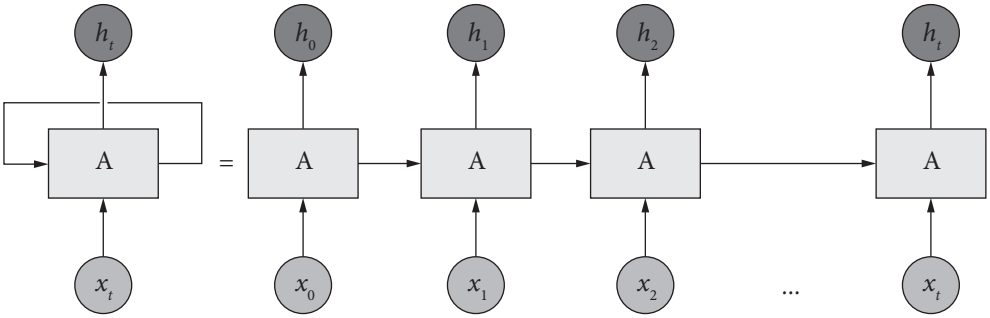


Рис. 13.1. Приклад зображення рекурентної нейронної мережі. Ліворуч — кругова діаграма, праворуч — розгорнутий графік для тієї самої РНМ

Наприклад, припустимо, що використовується згорткова нейронна мережа. У такому разі маємо визначити її структуру в термінах довжини вхідної послідовності, навчити окрему нейронну мережу для кожної довжини вхідної послідовності, а потім об'єднати їх усі, щоб отримати остаточний результат.

2. Завдяки структурі РНМ можна повторно використовувати одну й ту саму функцію переходу стану f та її вагу на кожному часовому кроці послідовності. Це дає нейронній мережі змогу краще вивчати високорівневі ознаки, які узагальнюються на послідовності різної довжини, навіть якщо в навчальному наборі даних може не бути таких довгих послідовностей. Крім того, структура РНМ дає можливість повторно використовувати ваги між різними послідовностями, що позитивно впливає на ефективність навчання.

РНМ можна використовувати для апроксимації будь-яких послідовностей вхідних даних обмеженої довжини за умови достатньої кількості нейронів. Це доводиться універсальною теоремою про апроксимацію для РНМ у роботі [13.8]. Виходячи з цього, можна використовувати рекурентні нейронні мережі для апроксимації комп'ютерної томографії для виявлення раку легенів, якщо розглядати її серію шарів як послідовність двовимірних зображень.

Як і традиційні нейронні мережі, навчання РНМ складається з двох основних етапів: прямого поширення, яке обчислює вихід мережі на основі вхідної послідовності, і зворотного поширення, яке використовується для обчислення градієнтів для алгоритму градієнтного спуску. Всі процеси навчання можна виразити трьома рівняннями:

$$h^{(t)} = \sigma(b + Wh^{(t-1)} + Ux^{(t)}); \tag{13.6}$$

$$o^{(t)} = c + Vh^{(t)}; \tag{13.7}$$

$$\hat{y}^{(t)} = g(o^{(t)}), \tag{13.8}$$

де b і c — вектори зсуву; U — вагова матриця зв'язків між вхідним шаром і прихованим шаром; V — вагова матриця зв'язків між прихованим шаром і

вихідним шаром; W — вагова матриця рекурентних зв'язків у прихованому шарі (тобто між послідовними моментами часу); σ — функція активації, що зазвичай реалізується як гіперболічний тангенс (tanh); g — функція активації вихідного шару, яка залежить від типу завдання та архітектури РНМ.

Зокрема, для задач, пов'язаних із передбаченням або класифікацією послідовностей, доцільно використовувати функцію активації *softmax*, яка перетворює вихід на ймовірнісний розподіл.

Коли визначено $o^{(t)}$, можна використати це значення для обчислення функції втрат, яку потрібно мінімізувати за допомогою градієнтного спуску. Для РНМ, що будує вихідну послідовність на основі вхідної, функція втрат послідовності може мати вигляд [13.7]

$$L(\{x^{(1)}, \ldots, x^{(T)}\}, \{y^{(1)}, \ldots, y^{(T)}\}) = \Sigma_t L^{(t)} =$$
$$= -\Sigma_t \log p_{\text{model}} (y^{(t)} \mid \{x^{(1)}, \ldots, x^{(t)}\}), \qquad (13.9)$$

де $p_{\text{model}} (y^{(t)} \mid \{x^{(1)}, \ldots, x^{(t)}\})$ обчислюється за послідовністю зчитування y з кроком у часі t з виходу РНМ $\hat{y}^{(t)}$.

Градієнти розраховуються на основі функції втрат (13.9). Вони обчислюються на розгорнутому графі РНМ за допомогою алгоритму зворотного поширення. Оскільки цей алгоритм застосовується до часових кроків вхідної послідовності, його називають зворотним поширенням у часі.

Є різні типи архітектур РНМ, які можна використовувати залежно від типу завдання, що потрібно вирішити.

1. Рекурентні нейронні мережі, які мають рекурентні зв'язки між прихованими станами (ці мережі видають результат після кожного часового кроку вхідної послідовності).

2. Рекурентні нейронні мережі, які мають рекурентні зв'язки лише між виходом з попереднього часового кроку та прихованим станом поточного часового кроку (ці мережі видають результат для кожного часового кроку вхідної послідовності).

3. Рекурентні нейронні мережі, які мають рекурентні зв'язки між прихованими станами і повертають єдиний результат для всієї вхідної послідовності.

Розглядаючи задачу діагностики раку легенів на КТ-знімках, потрібно використовувати третій тип РНМ, оскільки необхідно повернути єдиний прогноз про наявність раку легенів для всієї вхідної послідовності шарів КТ-сканувань. І все ж РНМ не зовсім придатні для аналізу шарів КТ-скану в описаній вище формі, оскільки вони можуть розглядати лише попередні шари, які були передані нейронній мережі перед поточним шаром. Коли рентгенологи аналізують КТ-знімки, вони можуть дивитися на шар в обох напрямках від поточного шару. Це потрібно для того, щоб краще зрозуміти

форму та морфологію аномальної ділянки в третій просторовій осі. Якщо рекурентна нейронна мережа може робити те саме, можна отримати більше просторової інформації про аномальну ділянку, надаючи мережі більше контексту для кращого прогнозування. Цього можна досягти за допомогою двоспрямованих рекурентних нейронних мереж.

Двоспрямовані рекурентні нейронні мережі

Для вирішення проблеми залежності кожного кроку від усієї послідовності, а не від минулих кроків, було винайдено рекурентні нейронні мережі [13.9]. Основна ідея такої мережі полягає в об'єднанні виходів двох РНМ, які розглядають послідовність від початку до кінця та від кінця до початку відповідно. Інакше кажучи, пропускаємо вхідну послідовність від початку до кінця через перший шар рекурентної нейронної мережі, потім пропускаємо ту саму послідовність від кінця до початку через другий шар рекурентної нейронної мережі. Нарешті, вихідні дані обох шарів об'єднуються на третьому шарі, формуючи вихідні дані всієї мережі (рис. 13.2). У випадку завдання класифікації раку легенів останній шар має повернути єдиний вихід з імовірністю наявності раку легенів.

Завдяки двоспрямованій структурі нейронної мережі кінцевий вихідний шар має всю інформацію про послідовність, включно з попередніми та наступними часовими кроками. Це дає мережі змогу навчитися високорів-

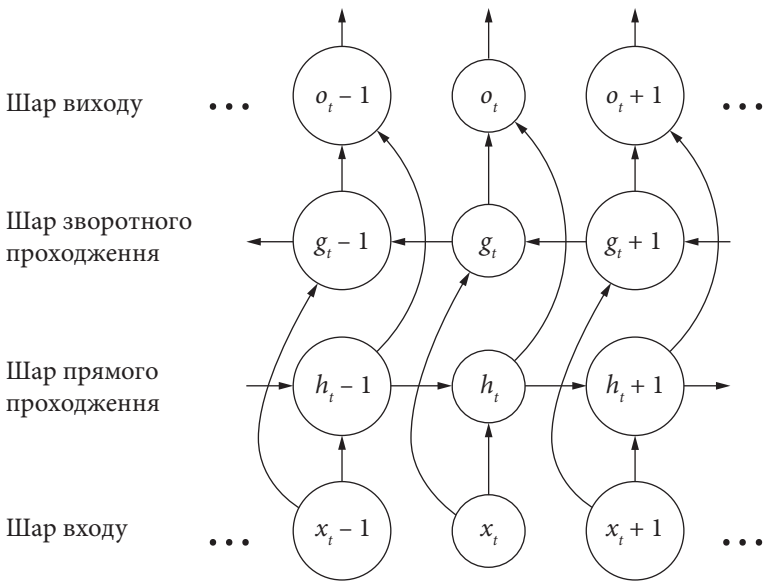


Рис. 13.2. Приклад розгорнутого графа двоспрямованої рекурентної нейронної мережі

невих функцій, які можуть вибирати найбільш релевантну інформацію з усієї послідовності на кожній часовій позначці.

Зворотне поширення за часовим алгоритмом буде таким самим, як і для двоспрямованої РНМ. Єдина відмінність полягає в тому, що обчислення градієнта слід починати з вихідного шару, а потім обчислювати градієнти для шарів прямого і зворотного поширення.

Двоспрямовані РНМ є досить ефективними в різних задачах у багатьох галузях, зокрема розпізнавання мови [13.10] та біоінформатики [13.11]. Однак прості РНМ усе ще мають суттєвий недолік у разі їх практичного застосування, який обмежує їх використання для аналізу КТ-сканів. Проблема полягає в тому, що вони зазвичай погано працюють з довгими послідовностями і запам'ятовують їхні останні елементи краще, ніж попередні. Якщо є важлива інформація на початку послідовності, а елемент перебуває досить далеко від поточного часового кроку, мережа, скоріше за все, не зможе поєднати цю інформацію і втратить її. Причина такої поведінки пов'язана з градієнтами, що вибухають та зникають [13.12]. Що довша послідовність, яку використовують як вхідні дані, то більша кількість рекурентних зв'язків і прихованих блоків отримують на розгорнутому графіку РНМ. Це призводить до появи дуже глибокої нейронної мережі, яку зазвичай важко навчати через поширення градієнта між шарами. Як відомо, дуже глибокі нейронні мережі схильні потерпати від градієнтів, що зникають та вибухають, тому доводиться використовувати спеціальні методи нормалізації, щоб навчити мережу правильно. РНМ мають іншу структуру і поведінку, тому неможливо легко застосувати до них традиційні методи нейронних мереж. Через це було винайдено різні типи рекурентних нейронних мереж — т. зв. вентильні рекурентні нейронні мережі, засновані на рекурентному вентильному вузлі.

Вентильні рекурентні нейронні мережі

Вентильні РНМ засновані на ідеї проходження через часові кроки послідовності з похідними, які не вибухають і не зникають. Ідея полягає у запровадженні замкненого самоциклу, що може вирішувати, яку інформацію слід забути, а яку зберегти. Це рішення має ґрунтуватися на особливостях, які нейронні мережі засвоюють у процесі навчання. Однією з найуспішніших архітектур, що реалізують цю ідею, є рекурентна нейронна мережа з довгою короткочасною пам'яттю (*Long-Short-Term-Memory, LSTM*) [13.13].

LSTM замінює звичайний блок РНМ на блок *LSTM*, який має внутрішню пам'ять і може рекурентно приєднуватися до інших блоків *LSTM. LSTM* має два типи з'єднань: зовнішні з'єднання, подібні до рекурентного з'єднання між прихованими блоками РНМ, і внутрішній стан $s_i^{(t)}$, який має рекурентне внутрішнє з'єднання із самим собою. Крім того, *LSTM*-блок має три основні

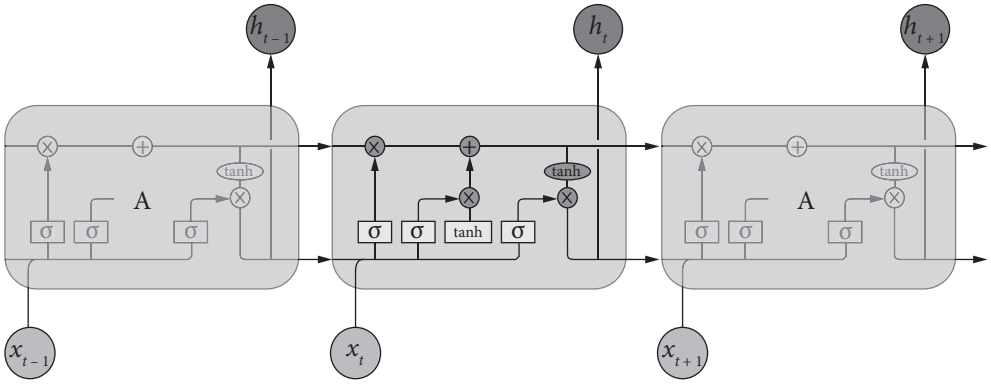


Рис. 13.3. Проста схема блока *LSTM*

вентилі, що контролюють потік інформації і вирішують, чи слід забути, чи зберегти нову інформацію і чи потрібно зберігати в пам'яті попередню інформацію [13.12]. Приклад архітектури *LSTM* наведено на рис. 13.3.

Пряме проходження через один блок *LSTM* складається з декількох основних кроків, які мають взаємодіяти з внутрішнім станом мережі. Першим кроком є блок забування, який вирішує, яку інформацію слід стерти у внутрішньому стані. Внутрішній стан зберігає всю інформацію з усіх попередніх кроків. Процес «забування» інформації з попередніх кроків можна описати за допомогою сигмоїдної функції та вагових матриць:

$$f_i^{(t)} = \sigma(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}), \tag{13.10}$$

де $x^{(t)}$ — поточний вхідний вектор на часовому кроці t; h^t — прихований одиничний вектор на поточному кроці, який містить інформацію з виходів блоків *LSTM* на попередніх кроках; b_i^f — вектор зсуву вентилів забування; U^f — матриця вхідних ваг для вентилів забування; W^f — матриця рекурентних ваг для вентилів забування.

Наступний крок для блока *LSTM* складається з декількох проміжних кроків. По-перше, вхідний вентиль вирішує, яку інформацію у внутрішньому стані слід поповнити новими даними. Далі мережа створює список нових елементів, які відображають нову інформацію, що має бути додана до внутрішнього стану. Насамкінець мережа об'єднує всю інформацію з попередніх кроків і оновлює внутрішній стан $s_i^{(t)}$. Усі ці операції описуються такими рівняннями:

$$s_i^{(i)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)}); \tag{13.11}$$

$$g_i^{(t)} = \sigma(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)}), \tag{13.12}$$

де b — вектор зсуву в блок *LSTM*; U — вхідні ваги в блок LSTM; W — рекурентні ваги у блоці *LSTM*; $g_i^{(t)}$ — зовнішня вхідна функція гейт-функції.

На останньому кроці роботи блока *LSTM* вирішується, яка інформація має бути повернута на виході. Вихідне значення обчислюється за допомогою механізму вихідних вентилів:

$$h_i^{(t)} = \tanh(s_i^{(t)} q_i^{(t)}); \tag{13.13}$$

$$q_i^{(t)} = \sigma(b_i^o + \sum_i U_{i,j}^o x_j^{(t)} + \sum_j W_{i,j}^o h_j^{(t-1)}), \tag{13.14}$$

де b^o, U^o, W^o — вектор зсуву, вхідна та рекурентна матриці ваг вихідного вентиля відповідно.

Архітектура *LSTM* є ефективною в реальних задачах у різних галузях і демонструє набагато кращу здатність працювати з довгостроковими залежностями порівняно з РНМ [13.10, 13.13]. Крім того, *LSTM* може бути двоспрямованою. Щоб досягти цього, слід дотримуватись підходу, подібного до стандартного для РНМ: одна *LSTM* має виконати пряме поширення від початку до кінця послідовності, потім — зворотне поширення від кінця до початку послідовності, наприкінці потрібно об'єднати виходи з обох шарів. Вихідні значення можна обчислити так само, як для стандартної двоспрямованої РНМ.

Успішність *LSTM* і можливість її розширення до двоспрямованого режиму роблять цю мережу придатним кандидатом для створення системи виявлення раку легенів. Для оброблення двовимірних зображень з шару КТ-сканування можна модифікувати *LSTM* для роботи з пікселями зображення, розглядаючи послідовності пікселів зображення зліва направо, справа наліво, знизу вгору і згори вниз із призначенням окремого блока *LSTM* для кожної послідовності. Однак такий підхід потребує великих обчислювальних потужностей і ускладнює збіжність нейронної мережі. Поліпшити його можна, використовуючи згорткову нейронну мережу для роботи з двовимірними зображеннями із заміною повнозв'язного шару на двоспрямований *LSTM*-блок. Утім, таку нейронну мережу складно навчити на невеликому наборі даних, який підготовлено для цього дослідження, тому для підвищення ефективності навчання потрібно використати механізм уваги.

Механізм уваги в комп'ютерному зорі

Механізм зорової уваги є одним з основних інструментів складного процесу мислення у людини, в тому числі процесу розуміння візуальної інформації [13.14]. Наприклад, рентгенологи, аналізуючи КТ-знімок легенів людини, не розглядають усі ділянки одночасно. Зазвичай людина концентрує свою увагу на деяких конкретних ділянках знімка. Спочатку рентгенолог оглядає різні місця КТ-зображення, щоб перевірити, чи немає на ньому підозрілих ділянок. Після виявлення підозрілих ділянок кожну таку ділянку розглядають окремо в деталях. Фокусуючись на певному місці на знімку, рентгенологи звертають увагу на дрібніші деталі, іноді переносячи

свою увагу на інші, сусідні структури на зображенні. Отже, людина більш обчислювально та ефективно працює з візуальною інформацією порівняно зі штучними нейронними мережами, оскільки людське зорове сприйняття не залежить від розміру та масштабу візуального вхідного сигналу.

Глибокі нейронні мережі обробляють вхідне зображення відразу, не розбиваючи його на менші частини. Чим більше вхідне зображення, яке потрібно обробити, тим більше пам'яті та обчислювальної потужності потрібно для оброблення — а отже, під час побудови нейронних мереж завжди є серйозні обмеження цих показників. І все ж будь-яка штучна нейронна мережа має власний неявний механізм внутрішньої уваги, що концентрує увагу нейромережі на конкретних ділянках вхідного зображення, які є найбільш важливими для виконання завдання. На рис. 13.4 показано, як глибока згорткова нейронна мережа шукає конкретні шаблони і відбирає пов'язані з ними вхідні фрагменти на кожному шарі. На останніх шарах мережі патерни виглядають досить подібними до об'єкта, який мережа має розпізнати.

Отже, згорткова нейронна мережа вибирає ділянки, пов'язані з об'єктом, який вона намагається розпізнати, тоді як інші, не пов'язані з ним, ігноруються або мають незначний вплив на кінцевий результат. Наприклад, на рис. 13.5 показано нейронну мережу, навчену робити текстовий опис вхідного зображення, яка концентрує свою увагу на об'єкті, що відповідає назві цього об'єкта в тексті. Аналогічну картину можна спостерігати і в завданні класифікації. Цю властивість глибоких згорткових нейронних мереж можна розглядати як певну форму механізму уваги, якого вони набувають на основі навчальних даних.

Дія механізму неявної уваги пояснюється тим, що нейронна мережа з глибоким навчанням вчиться розпізнавати об'єкти, збільшуючи або зменшуючи їхні ваги. Інакше кажучи, нейромережа вчиться сильніше реагувати на деякі локальні патерни на зображенні, ігноруючи інші. У результаті на зображенні виділяються лише релевантні пікселі. Часто властивість глибоких нейронних мереж використовують для аналізу та інтерпретації їхньої роботи. Можна обчислити матрицю Якобі (Карл Густав Якобі (Carl Gustav Jacob Jacobi, 1804—1851)), щоб виділити місце, до якого чутлива мережа.

Матриця Якобі містить інформацію про те, наскільки чутливі виходи моделі до входів:

$$J = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_k} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_k} \end{bmatrix}, \tag{13.15}$$

де J — матриця Якобі, обчислена під час зворотного поширення; x — вхідний вектор розміром k; y — вихідний вектор розміром m.

Рис. 13.4. Візуалізація функції *GoogleNet* після навчання на *ImageNet*

Рис. 13.5. Приклад механізму уваги в нейронних мережах, які використовують для вирішення завдання опису зображень

На відміну від штучних нейронних мереж людина може керувати своєю увагою. Наприклад, людина може прочитати додаткову інформацію в довіднику і використати її, щоб сконцентрувати свою увагу на певній властивості зображення в певному його місці. Можна спробувати створити подібні механізми уваги в глибоких нейронних мережах, щоб підвищити ефективність процесу їх навчання. У контексті глибокого навчання цей механізм зазвичай називають механізмом явної уваги, оскільки він дає можливість надавати явну інформацію про властивості, що мають привернути більше уваги нейронної мережі для отримання правильних результатів. Одним з чудових прикладів того, як цей механізм можна застосувати до нейронних мереж, наведено в роботі [13.17], де автори намагаються реалізувати модель класифікації цифр, використовуючи рекурентні нейронні мережі на обмеженій частині вхідного повідомлення.

Основні переваги використання механізму явної уваги в нейронних мережах такі:

1. *Обчислювальна ефективність навчання.* Завдяки механізму явної уваги можна виділити найбільш релевантні ознаки на етапі навчання нейронної мережі, що прискорює процес навчання і дає змогу отримати вищу точність мережі.

2. *Масштабованість.* Завдяки механізму уваги робота з цілим зображенням не потрібна, оскільки можна виділити найбільш релевантні локації і працювати тільки з ними.

3. *Більш прямолінійна інтерпретація* результатів роботи мережі.

Вплив механізму явної уваги на нейронні мережі залежить від типу використовуваної уваги [13.19]. Є два основні типи механізмів явної уваги — м'який і жорсткий. Найпоширенішим є м'який механізм уваги [13.20], оскільки мережі, які його використовують, усе ще є повністю диференційованими і можуть навчатися за допомогою алгоритму градієнтного спуску та зворотного поширення.

Механізм м'якої уваги реалізовано у вигляді адаптивної матриці ваг (маски, що накладається на шар нейронної мережі). Отже, мережа має можливість відбирати та ізолювати важливу інформацію для поточного входу. Після застосування маски неважлива інформація не поширюється на наступні шари через мережу. Це відбувається тому, що всі ваги, пов'язані з неважливими пікселями зображення, на виході шару дорівнюють нулю. Ваги стають нульовими в процесі навчання. Процес навчання мережі є більш ефективним, оскільки зменшується кількість зв'язків між різними частинами вхідних даних і мережі не потрібно вивчати цей шаблон з нуля. Додатковою перевагою м'якого механізму уваги є простота реалізації та повна диференційованість нейронної мережі, що дає змогу повторно використовувати алгоритми градієнтної оптимізації. Недоліком м'якої уваги є те, що не всі ваги для незв'язаних входів дорівнюють нулю, а отже, частина неважливої інформації все одно проходить через шари мережі. Це може негативно вплинути на точність прогнозування мережі.

З іншого боку, механізм жорсткої уваги використовує лише підмножину найважливіших вхідних даних: усі неважливі вхідні значення відфільтровуються і не проходять через наступні шари мережі. Це робить механізм жорсткої уваги ефективнішим за механізм м'якої, оскільки відбувається зменшення розміру вхідних даних з повним вилученням усіх неважливих значень. Утім, таке вилучення є дискретною операцією, яка не є диференційованою, тому її застосування робить усю нейронну мережу недиференційованою. Одним із способів подолання цієї проблеми є перетворення завдання навчання під наглядом на завдання навчання з підкріпленням [13.17]. З іншого боку, деякі вчені намагаються використовувати

функцію припущення як жорстку увагу. Наприклад, як функцію припущення l^2 можна використати норму [13.19]. І все-таки спосіб подолання цієї проблеми досі залишається активним напрямом досліджень.

Враховуючи ефективність механізму явної уваги, можна використати цю ідею для покращення ефективності навчання нейронної мережі *LSTM* для досягнення вищої точності.

Рекурентна нейронна мережа з механізмом уваги

Добрим прикладом використання механізму м'якої уваги є двоспрямована рекурентна нейронна мережа *RADNET* [13.21], створена для виявлення крововиливів на КТ головного мозку. У цьому дослідженні використовується згорткова нейронна мережа *DenseNet* для роботи з кожним шаром зображення КТ-скану і *LSTM* як рекурентна нейронна мережа для оброблення КТ-скану як послідовності. Основною особливістю цієї нейронної мережі є введення допоміжних завдань між блоками *DenseNet* у згортковій нейронній мережі. Вони дають змогу повторно використовувати наявні мітки локалізації крововиливу в мозку в наборі даних у вигляді механізму м'якої уваги. У *RADNET* допоміжні завдання реалізовано у вигляді шару деконволюції, який додається після останнього шару з'єднання щільного блока. Шар деконволюції збільшує вибірку даних до початкового розміру зображення, щоб побудувати функцію втрат завдання сегментації. Далі вихід кожної допоміжної задачі вводиться у функцію втрат нейронної мережі, яка обчислює суму функції перехресної ентропії для завдання класифікації та функції втрат допоміжних задач. Результати показують, що механізм уваги значно підвищує точність моделі на завданні класифікації і містить інформацію про місцезнаходження крововиливу в мозок.

Для завдання виявлення раку легенів повторно використовуємо ідею надання інформації про місцезнаходження у вигляді м'якої уваги, але з модифікованим механізмом уваги, аби підвищити ефективність навчання моделі. Детальніше цю тему розкрито у розділі з описом процесу навчання.

Основна перевага методу полягає в тому, що нейронна мережа може вивчати ознаки та їх просторові залежності в наскрізний спосіб. Крім того, модель дає змогу повторно використовувати знання рентгенологів про те, де саме знаходиться злоякісний вузлик, щоб прискорити процес навчання і підвищити загальну точність моделі. Поточні експерименти демонструють, що ці моделі не дають кращих результатів без механізму уваги, ніж наскрізна 3D-згорткова нейронна мережа (див. підрозд. 13.3).

Основним недоліком згаданого підходу є необхідність додаткового маркування для ідентифікації розміщення легеневих вузликів на КТ-скані. Як правило, додаткове маркування є «дорогою» частиною розроблення

системи виявлення з точки зору значних витрат робочого часу експерта. Крім того, модель не надає інформації про розміщення злоякісного легеневого вузла, що може змусити лікаря-рентгенолога не довіряти системі виявлення і уникати її використання.

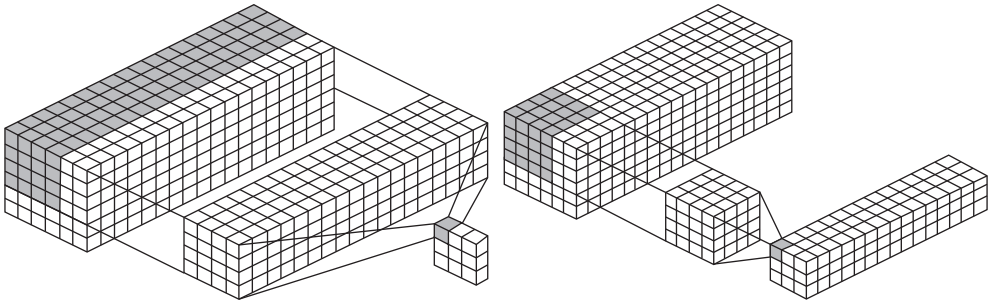
13.3. Тривимірна згорткова нейронна мережа для виявлення раку легенів

Наступна доступна стратегія, що є найпопулярнішою на сьогодні, полягає у використанні тривимірної природи КТ-сканування і роботі з даними як з тривимірними об'єктами. КТ-скан для діагностики раку легенів становить собою тривимірне зображення грудної клітки людини, яке містить зображення легенів пацієнта. Складені разом усі зрізи КТ-скану формують тривимірне зображення грудної клітки людини, яке можна використовувати як вхідні дані для нейронної мережі. Крім легеневої тканини на остаточному зображенні грудної клітки видно судини, кістки та інші тканини людського тіла, не пов'язані з легенями. Через це система виявлення раку легенів зазвичай використовує додаткові кроки попереднього оброблення, щоб очистити їх і спростити процес навчання на невеликому наборі даних. У разі, якщо для етапу навчання доступна велика кількість даних, а обчислювальних потужностей достатньо для побудови досить великої нейронної мережі, складене 3D-зображення можна використати без етапів очищення, що може надати ЗНМ додаткову інформацію для вивчення складніших ознак.

У попередніх підрозділах розглянуто згорткові нейронні мережі, які працюють з двовимірними зображеннями. В ідеальному випадку потрібно повторно використовувати ці мережеві архітектури для роботи з тривимірними даними. Утім, усі архітектури працюють з двовимірними даними, а отже, вони використовують двовимірну операцію згортки, яка обчислюється як сума всіх значень каналів, помножених на відповідні значення ядра. Інакше кажучи, операція двовимірної згортки не може описати просторові відношення у всіх трьох напрямках, оскільки вона рухається тільки у двох напрямках — по ширині й висоті. Це призводить до того, що нейронна мережа втрачає просторову інформацію в напрямку глибини (рис. 13.6, *а*).

Оскільки двовимірна згортка втрачає просторову інформацію третьої просторової осі, вона непридатна для роботи з усім КТ-знімком. Проте для пом'якшення цієї проблеми можна використати операцію тривимірної згортки.

Операція тривимірної згортки має глибину фільтра, меншу за глибину вхідних даних, а отже, ковзне вікно рухається в усіх трьох напрямках у ви-



а *б*

Рис. 13.6. Відмінність між 2D- і 3D-згортками: *а* — робота 2D-згортки з тривимірними даними; *б* — застосування 3D-згорткою кубічного ядра до тривимірних даних

гляді куба (рис. 13.6, *б*). Тривимірну згортку в позиції x, y, z у каналі i можна записати як

$$S_{i,x,y,z} = \sum_l \sum_{p=0}^{P-1} \sum_{q=0}^{Q-1} \sum_{r=0}^{R-1} I_{l,x+p-1.y+q-1,z+r-1} K_{i.l.p,q,r}, \qquad (13.16)$$

де p, q, r — координати тривимірного ядра операції згортки; K — ядро; I — тривимірні дані по каналу l.

Тривимірна згортка працює подібно до двовимірної, застосовуючи до вхідних даних близькі архітектурні концепції, як-от локальні сприйнятливі поля та спільні ваги. Єдина відмінність полягає в тому, що операція тривимірної згортки застосовується до куба з використанням меншого куба як ядра, натомість операція двовимірної згортки просто підсумовує всі значення в третьому вимірі.

Щоб згорткова нейронна мережа працювала з тривимірними даними, можемо замінити всі двовимірні компоненти їхніми тривимірними аналогами: двовимірну згортку, а також двовимірний шар об'єднання слід замінити їх тривимірними версіями. Тривимірне об'єднання застосовується так само, як і у двовимірних згорткових нейронних мережах, за винятком того, що воно працює з трьома вимірами, даючи можливість мережі оперувати кубами замість квадратів.

У рамках цієї роботи навчені нейронні мережі C3D або 3D-*DenseNet* [13.24] досягли точності 0,63 та 0,70 відповідно. Мережі 3D-ЗНМ навчалися на основі зміненого 3D-зображення, оскільки оригінальне 3D-зображення, отримане за допомогою 3D-ЗНМ, не вміщується в пам'ять відеокарти. Приклад архітектури *3D-DenseNet* наведено на рис. 13.7.

Як показують результати, побудова та навчання моделі безпосередньо на 3D-зображенні легенів не дає достатньо повних відомостей. Однією з причин цього є ризик втрати деяких деталей під час зміни розміру 3D-скану до нижчої роздільної здатності, які б дали змогу побудувати більш сильні

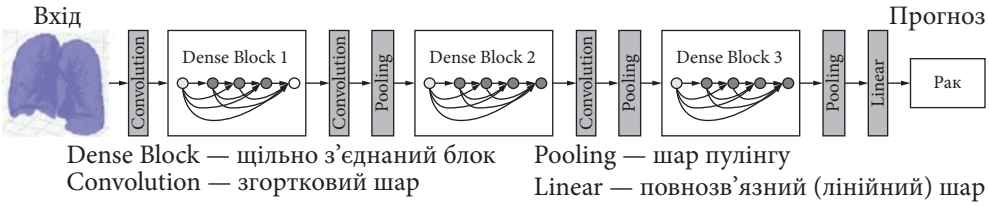


Рис. 13.7. Архітектура *3D-DenseNet,* навчена на повному 3D-зображенні пацієнта, створеному зі зрізів КТ

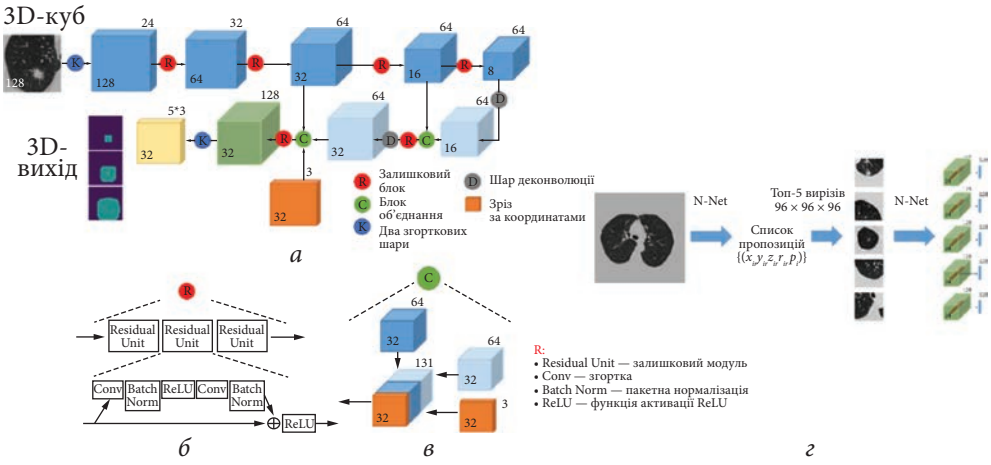


Рис. 13.8. Вбудована двоетапна система виявлення раку легені: *а — в* — мережа сегментації; *г* — мережа класифікації

ознаки. Крім того, не вистачає навчальних даних для отримання мережею більшої кількості інформації про природу злоякісних вузликів. Тому для досягнення кращої продуктивності слід використовувати двоетапний підхід, як і в роботі [13.26], що забезпечує виконання завдання класифікації раку легенів на рівні людини.

Основна ідея двоетапного підходу полягає в тому, що завдання виявлення раку легенів можна розділити на два менші завдання: сегментацію і класифікацію. На етапі сегментації нейронна мережа знаходить всі легеневі вузлики на КТ-скані, а потім передає їх на наступний етап класифікації. На етапі класифікації система виявлення вибирає найбільш імовірні злоякісні легеневі вузлики і вирішує, чи є рак на зображенні, використовуючи припущення про багаторазове навчання. Впровадження двоетапності в систему виявлення раку легенів — це можливість адаптувати будь-які сучасні моделі сегментації, що можуть повернути менший фрагмент оригінального 3D-зображення з легеневими вузликами замість усього зображення легенів. Це спрощує розроблення етапу класифікації, оскільки менші фраг-

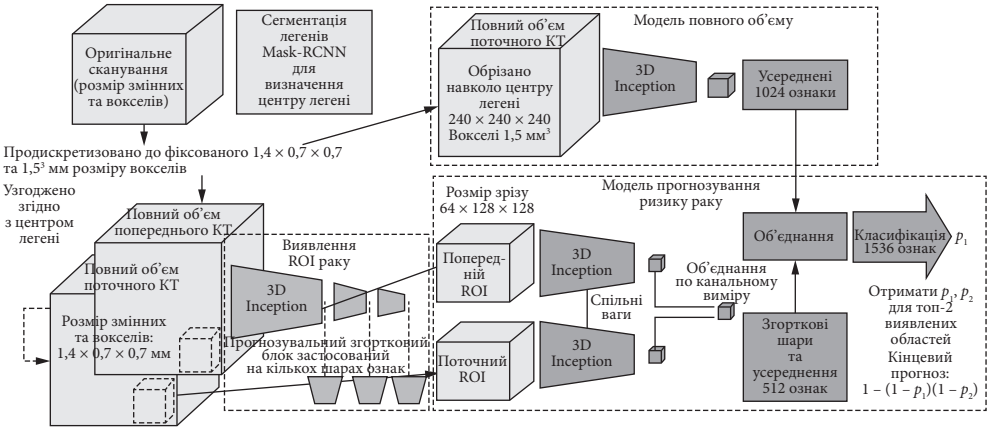


Рис. 13.9. Чотириетапна наскрізна система виявлення раку легенів високої точності

менти набагато простіше помістити в пам'ять відеокарти. Крім того, це дає можливість навчати системи виявлення легенів на меншій кількості даних.

На сьогодні завдання сегментації виконується, як правило, за допомогою *V-Net* [13.27], *Faster-RCNN* [13.28] або *Mask-RCNN* [13.29], адаптованих до 3D-даних [13.24]. Зазвичай на етапі класифікації використовують просту ЗНМ, яка виводить імовірність того, що легеневий вузол є раком. На етапі класифікації ЗНМ «ділиться» вагами із ЗНМ, що функціонувала на етапі сегментації з метою прискорення процесу навчання і підвищення точності класифікації, а також виявлення легеневих вузликів завдяки зменшенню проблем перенавчання на обох етапах. Як зазначено в роботі [13.26], двоетапні системи виявлення раку легенів досягли рівнів точності 0,87 і 0,92 (рис. 13.8).

Попри те, що двоетапний підхід сприяє досягненню точності на рівні людини, він має кілька недоліків. По-перше, підготовка ЗНМ до етапу сегментації потребує додаткової роботи з маркування, щоб додати до набору даних інформацію про позитивних і негативних кандидатів щодо злоякісних вузлів у легенях. Маркування є дорогим і трудомістким процесом у медичній галузі, що потребує значних витрат робочого часу рентгенологів. По-друге, на етапі прийняття рішення про наявність раку модель втрачає глобальну інформацію про легеню і положення легеневого вузлика-кандидата. Така інформація допомагає визначити злоякісність легеневого вузла і сприяє кращому визначенню наявності раку. Також двоетапний підхід усе ще ігнорує інформацію про ріст легеневого вузлика з часом. Цю проблему можна вирішити, додавши до конвеєра ще кілька етапів.

Автори статті [13.29] в журналі «*Nature Medicine*» описують систему скринінгу раку легенів, яка розділяє проблему виявлення на чотири компоненти або етапи (рис. 13.9). По-перше, нейронна мережа сегментації легенів використовується для сегментації легенів із загального КТ-скану. Мета

цього етапу — очистити всі не зв'язані елементи легеневої тканини і залишити на зображенні лише легені та пов'язані з ними тканини, які можуть бути використані на наступних етапах. На другому етапі використовується інша ЗНМ, що генерує зображення ділянки легеневих вузликів подібно до першого етапу двоетапного підходу. На третьому етапі використовується наскрізна 3D-згорткова нейронна мережа, яка навчається на основі сегментованого зображення легенів з першого етапу. На завершальному етапі використовуються наскрізні 3D-згортки, згенеровані ЗНМ, з метою надання більш глобального контексту для конкретного легеневого вузлика. Нарешті, модель прогнозування ризику раку використовує згенеровані ознаки з третього етапу і кандидатів у легеневі вузлики з другого етапу для прогнозування ризику раку. Крім того, останній етап може використовувати попереднє сканування пацієнта для точнішого прогнозу. Як видно зі схеми на рис. 13.9, система виявлення генерує кандидатів у легеневі вузлики на попередньому КТ-скануванні і поєднує їх з кандидатами на поточному КТ-скануванні на рівні ознак. Отримані ознаки об'єднано з ознаками з третього етапу, а потім використано для прогнозування ризику розвитку раку легенів. Чотириступеневий підхід дав можливість досягти вищої точності на використаному наборі даних порівняно із середньостатистичним лікарем-рентгенологом.

13.4. Набори даних та їх попереднє оброблення

Дані та набори даних. Рак легенів —захворювання, яке виникає, коли клітини в легенях мутують і утворюють підозрілі утворення. Є два основних типи раку легенів — дрібноклітинний і недрібноклітинний, з яких найпоширенішим є недрібноклітинний рак легенів [13.31].

Залежно від місця локалізації розрізняють три різні типи недрібноклітинного раку легенів.

1. ***Аденокарцинома*** — форма недрібноклітинного раку легенів, яка часто локалізується у зовнішній частині легенів. Розвивається в клітинах епітеліальних тканин, які вистилають порожнини і поверхню тіла та утворюють залози.

2. ***Плоскоклітинний рак*** — форма недрібноклітинного раку легенів, яка зазвичай виявляється в центрі легені поруч з дихальною трубкою (бронхом).

3. ***Великоклітинний рак*** — форма недрібноклітинного раку легенів, яка може виникнути в будь-якій частині легенів і має тенденцію рости і поширюватися швидше ніж аденокарцинома чи плоскоклітинний рак.

Усі ці типи раку небезпечні, швидко прогресують і часто перебігають без специфічних симптомів. Тому найкращою практикою вважають регулярне проведення скринінгу на рак легенів, особливо для тих, хто належить до групи ризику. Стандартним інструментом для скринінгу є низькодо-

зова комп'ютерна томографія грудної клітки [13.31, 13.32]. Низькодозова комп'ютерна томографія — це особливий вид рентгенівського дослідження, що складається з серії рентгенівських знімків під різними кутами і з різних боків для отримання поперечних (томографічних) зображень тканин, кісток або кровоносних судин людського тіла. Залежно від потреб рентгенолога його можна розглядати у 2D- або 3D-форматі.

Зазвичай рентгенолог виявляє рак легенів, переглядаючи зріз за зрізом, щоб знайти пухлину або аномальні ділянки з ознаками раку. Такі аномальні ділянки на КТ легенів називаються легеневими вузликами. Більшість вузликів, які видно на КТ, не є раком, але легеневі вузлики діаметром понад 6 мм мають високий ризик бути небезпечними [13.32]. Якщо аномальна ділянка невелика, пацієнта можуть попросити повторити скринінгове обстеження через пів року або рік. Під час наступного скринінгу рентгенолог перевірить, чи не збільшилася аномальна ділянка в розмірах. Якщо розмір ділянки значно збільшився, є висока ймовірність раку легенів, у такому разі для підтвердження діагнозу необхідно зробити біопсію легенів. Тому автоматичні системи виявлення раку легенів часто надають інформацію про розміщення легеневого вузла, додаючи до конвеєра системи етап сегментації вузла. Це дає рентгенологам більше подробиць про діагнози, передбачені системою.

В експериментах роботи було використано набори даних *LUNA* [13.31] та *Data Science Bowl* 2017 (*DSB* 2017) [13.33]. Набір даних *DSB* 2017 містить КТ-скани понад 1000 пацієнтів з мітками «наявність раку легенів». Один рядок набору даних відображує КТ грудної клітки пацієнта з відповідною міткою. Мітка показує, чи міститься в якомусь місці КТ-знімка рак. Діагноз було підтверджено у лікарні. *DSB* 2017 розділений на навчальний, валідаційний і тестовий набори даних, які містять відповідно 1397, 198 і 506 КТ-знімків легенів пацієнта. Навчальний набір даних дуже незбалансований і містить 1035 прикладів без раку і 365 КТ-знімків, які містять рак легенів. Приклад КТ-скану з набору даних наведено на рис. 13.10.

Набір даних *LUNA* містить інформацію про результати КТ-скринінгу 888 пацієнтів. Набір даних було створено і анотовано професійними рентгенологами, він містить анотацію для 1186 легеневих вузликів.

DSB 2017 і *LUNA* — абсолютно різні набори даних [13.32, 13.33]. *LUNA* надає багато анотацій для вузликів розміром менш як 6 мм, тоді як у наборі даних *DSB* 2017 використовуються вузлики з більшими діаметрами. Наприклад, більшість анотованих вузликів у наборі даних *LUNA* мають розмір 5—20 мм, тоді як у *DSB* 2017 — 10—44 мм (рис. 13.11).

Щоб вирівняти розподіл між двома наборами даних, було видалено всі вузлики діаметром менш як 6 мм, оскільки вони, скоріше за все, не є раком.

Етапи попереднього оброблення даних. Одиницею виміру в КТ є одиниця Хаунсфілда *(HU)* (Годфрі Ньюболд Хаунсфілд (Godfrey Newbold

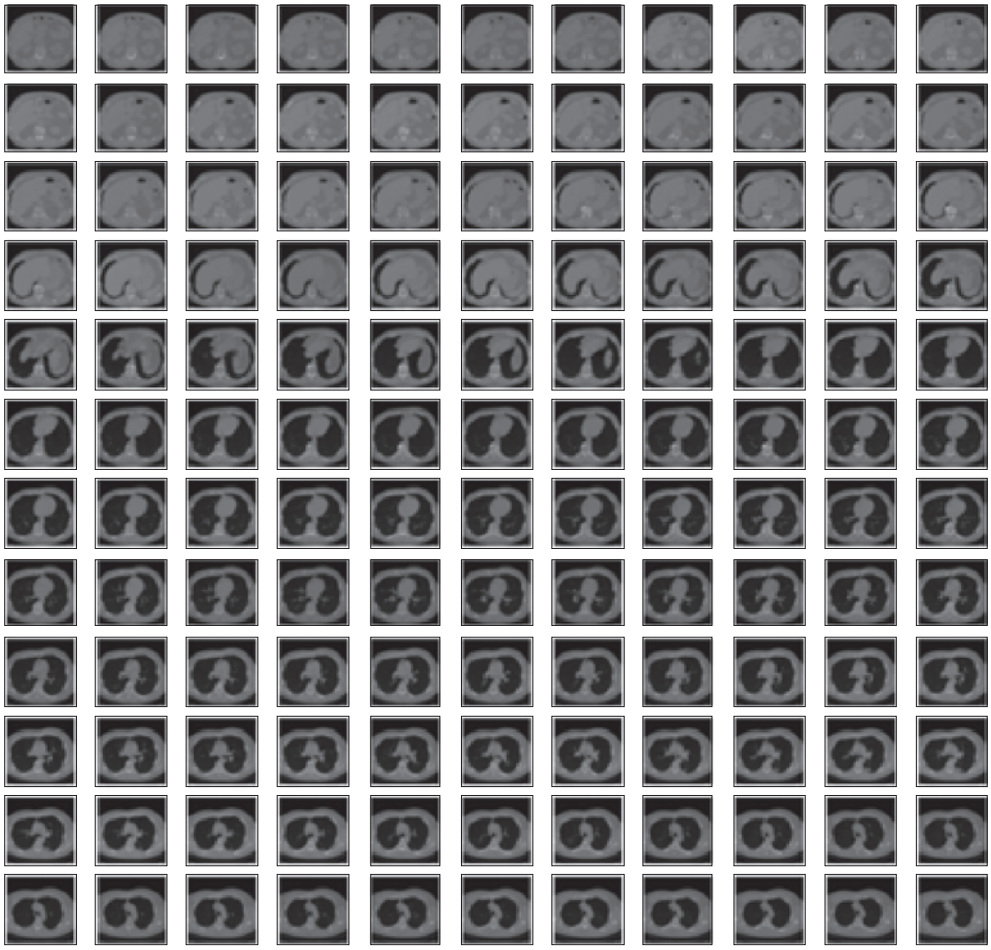


Рис. 13.10. Приклад КТ-скану з набору даних *DSB* 2017

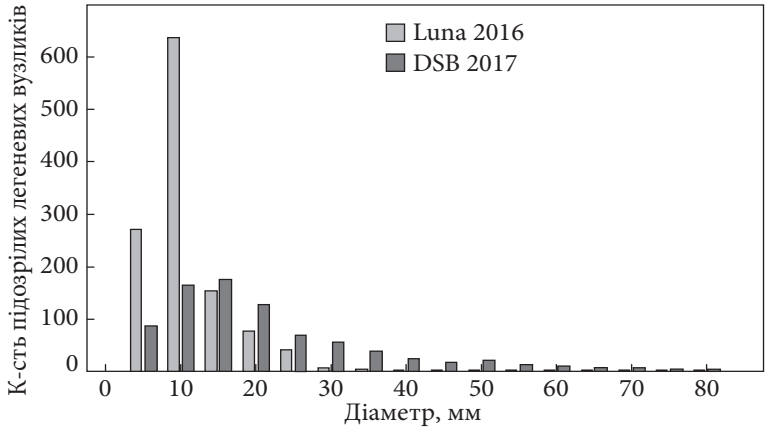


Рис. 13.11. Розподіл діаметрів потенційно злоякісних легеневих вузликів у наборах даних *LUNA* 16 та *DSB* 2017

Hounsfield, 1919—2004)), яка є стандартною кількісною шкалою для опису інтенсивності випромінювання. Всі КТ-сканери ретельно відкалібровано для точного вимірювання. У наборі даних *DSB* 2017 р. за замовчуванням дані зберігаються у відносному значенні радіоінтенсивності. Тому перед переданням нейромережі їх потрібно перетворити. На щастя, необхідні значення зручно зберігаються в метаданих файлів КТ-сканувань:

$$\text{HU} = \text{pixel_value} \times \text{rescale_slope} + \text{rescale_intercept}, \qquad (13.17)$$

де pixel_value — значення пікселя, отримане зі сканера; rescale_slope та rescale_intercept — параметри масштабу для відносних значень радіаційної щільності.

Набір даних *LUNA* не потребує цього кроку, оскільки значення пікселів КТ-сканувань вже переведено в *HU*. Кожна тканина (кістка, кров та ін.) мають певне значення *HU*. Цей факт можна використати для того, щоб відфільтрувати із зображення елементи, які не належать до речовини легеневої тканини. У нашому випадку це всі значення, які менші або дорівнюють 600. Після цього на 3D-зображенні легенів знаходять найбільші зв'язані ділянки. Як правило, обрана ділянка представляє легені; отже, можна видалити все, що знаходиться за межами цієї ділянки, щоб залишити лише тканини, пов'язані із зображенням легенів. Останніми кроками попереднього оброблення є зміна розміру зображення до відповідного та нормалізація значень пікселів зображення. У наступних розділах описано новий розмір зображення для кожної моделі окремо, оскільки різні моделі використовують різні розміри зображень, щоб відповідати вимогам до пам'яті. Нормалізація здійснюється за допомогою рівняння:

$$\text{new_pixel} = \frac{\text{pixel}–\text{image_min_bound}}{\text{image_max_bound}–\text{image_min_bound}}, \qquad (13.18)$$

де *pixel* — поточне значення пікселя зображення; *image_max_bound* = 400 — це граничне значення, вище якого інтенсивність у шкалі Гаунсфілда (*HU*) не враховується. Це пов'язано з тим, що всі значення понад 400 відповідають різним типам кісткової тканини з високою щільністю, що для даної задачі не диференціюються; *image_min_bound* = –1000 — це щільність повітря за *HU*. Усі значення нижчі за це вважаються нерелевантними для подальшої обробки.

Доповнення даних. Доповнення даних (аугментація) — ще один метод подолання перенавчання у процесі навчання глибоких нейронних мереж. Це досягається застосуванням заздалегідь визначених перетворень до наявних даних і використанням їх для навчання. Зазвичай трансформації не збільшують розмір навчального набору даних, але збільшують його варіативність завдяки випадковому вибору трансформації. Трансформація визначається заздалегідь і в довільній кількості. Потім, під час навчання, випадково вибирається оригінальне зображення або одна з трансформацій.

Оскільки в цій роботі використовуються тривимірні дані, кількість можливих перетворень більша ніж для двовимірних даних. Отже, техніка доповнення даних має бути потужною для навчального процесу. У процесі навчання використовували такі перетворення.

1. Випадковий поворот зображення на певний кут за однією з осей.

2. Зменшення розміру зображення у декілька разів (0 … 0,15) і заповнення нулями, щоб повернути початковий розмір зображення.

3. Випадкове переміщення зображення на деяку випадкову відстань (діапазон 15 %) від напрямку однієї з осей.

4. Випадкове обертання зображення в усіх трьох напрямках.

Усі ці перетворення допомогли збільшити кількість варіацій зображень і поліпшити узагальнення нейромережі. Оскільки навчання нейронної мережі відбувається впродовж багатьох епох, кожне перетворення застосовується більш як один раз.

13.5. Експериментальні дослідження і аналіз результатів

Розглянемо експерименти, виконані для кожної стратегії, описаної в підрозд. 13.1. Експерименти було проведено з використанням наборів даних *LUNA* та *DSB* 2017, описаних у підрозд. 13.2 [13.3, 13.32]. Як апаратне забезпечення використовували витискальний обчислювальний екземпляр у *Google Cloud* з однією картою *Nvidia Tesla* P100, 8 ГБ оперативної пам'яті та процесором *Intel* для стандартного типу машини. Як фреймворк для машинного навчання було використано *Keras* з бекендом *Tensorflow*. У цьому підрозділі розглядаються кроки, реалізовані для навчання моделей за кожною стратегією.

***DenseNet* та завдання багаторазового навчання (*MIL*) з розрідженим призначенням міток.** Ця модель відповідає підходу, описаному в підрозд. 13.1. Як 2D-згорткову нейронну мережу обрали *DenseNet* [13.3], оскільки ця мережа суттєво зменшує кількість параметрів і водночас демонструє найсучасніші результати на наборі даних *ImageNet*. Для навчання моделі повторно використали попередньо навчену *DenseNet* на *ImageNet* з глибиною 121 для ініціалізації ваг нейронної мережі на початку процесу навчання. Кожен зріз КТ-скану було приведено до розмірності 200 × 200; кількість зрізів не змінювалася для цієї моделі, навіть якщо вона була різною для кожного КТ-скану. Під час навчання вихідні дані *DenseNet* для кожного зрізу КТ-скану накопичувалися і передавалися до функції втрат, розрахованої за рівнянням (13.4). Використовували коефіцієнт розрідженості, що дорівнює 0,0001. Після отримання значень функції втрат для всіх зрізів КТ-скану для обчислення градієнтів застосовували алгоритм зворотного поширення. Для оптимізації функції втрат використовували алгоритм

оптимізації *Adam* з налаштуванням швидкості навчання вручну в процесі навчання і початковим значенням 0,00005. Процес навчання зупинявся, коли втрати не зменшувалися протягом декількох епох.

Після завершення навчання модель показала низький рівень точності — 0,62 на валідаційному наборі. Основною причиною такого результату навченої нейронної мережі є те, що вона втрачає тривимірну інформацію про природу, яка є важливою частиною розуміння наявності раку легенів.

Рекурентна нейронна мережа з механізмом уваги для системи виявлення раку легенів. Щоб застосувати рекурентну нейронну мережу, можна розглянути КТ-скан як послідовність двовимірних зображень. Оскільки маємо послідовність, можна використовувати рекурентну нейронну мережу для оброблення даних. Для роботи з двовимірним зображенням було вирішено використати архітектуру *DenseNet*-121, попередньо навчену на наборі даних *ImageNet* (щоб мати кращу ініціалізацію ваг на початку навчання). Аналогічно до [13.22], додали допоміжні завдання після кожного щільного блока після останнього шару з'єднання блока. Ці допоміжні завдання використовуються як механізми м'якої уваги для надання нейронній мережі інформації про місцезнаходження злоякісного легеневого вузла. Допоміжні завдання, на відміну від [13.22], реалізували у мережі *U-Net* [13.36], де кожен шар дискретизації пов'язаний з відповідним шаром згортки (рис. 13.12). Такий зв'язок має зменшити кількість втраченої інформації після послідовного використання операцій згортки та підвищення дискретизації. Вихідна маска допоміжного завдання є меншою за вхідне зображення, щоб зменшити обмеження пам'яті для моделі. Шар підвищеної вибірки складається з декількох згорткових шарів розміром 1×1 та деконволюційного шару.

Розміщення пухлини на рис. 13.13 подано у вигляді бінарної маски, доступної на навчальному наборі даних.

Для навчання побудованої моделі здійснили додаткові кроки з попереднього оброблення даних. У наборі даних *LUNA* є мітки розміщення легеневих вузликів, які використали для створення бінарної маски. У наборі даних *DSB* 2017 немає таких міток, тому було повторно використано мітки, описані в [13.26].

LSTM було використано як рекурентну нейронну мережу. Кінцева функція втрат є зваженою сумою втрат перехресної ентропії для завдань класифікації та сегментації, виконаних трьома додатковими шарами семплінгу. Навчання здійснюється за допомогою стохастичного градієнтного спуску зі швидкістю навчання, що регулюється вручну в процесі навчання, і початковим значенням 0,0005. Фінальна навчена модель показує точність 0,81 на перевірному наборі даних.

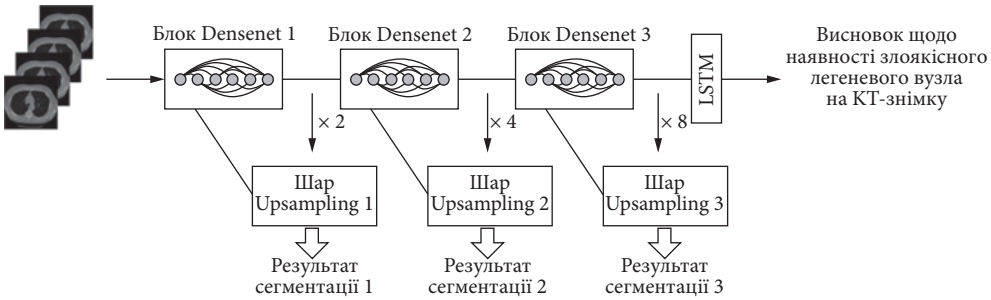


Рис. 13.12. Навчена рекурентна нейронна мережа з механізмом м'якої уваги

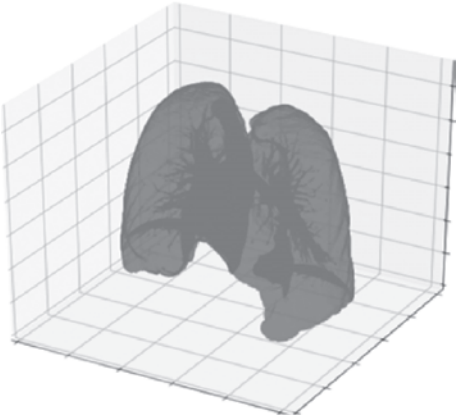


Рис. 13.13. Тривимірне зображення легенів після кроків попереднього оброблення

Таблиця 13.1

Порівняння точності систем виявлення раку легенів

Модель	Набір даних	Точність
Завдання *DenseNet* + *MIL* з розрідженим призначенням міток	*Luna* + *DSB* 2017	0,62
C3D	*Luna* + *DSB* 2017	0,65
3D-DenseNet	*Luna* + *DSB 2017*	0,71
DenseNet + *LSTM* + М'яка увага	*Luna* + *DSB* 2017	0,81
Модель-переможець *DSB* 2017	*Luna* + *DSB* 2017	0,87
DeepLung	*Luna*	0,92
Модель скринінгу легенів	*Luna* + *NLST* (набір даних *DSB* 2017 на основі цього набору даних)	0,95

Тривимірні згорткові нейронні мережі *C3D-* та *3D-DenseNet.* *C3D-* та *3D-DenseNet* [13.24] — це тривимірні згорткові нейронні мережі (ЗНМ), навчені наскрізним способом. Було складено всі зрізи КТ разом, щоб використовувати 3D-зображення легенів пацієнта як вхідні дані для 3D-ЗНМ. Кількість зрізів для кожного КТ-скану неоднакова, оскільки використовуються різні КТ-сканери з різною роздільною здатністю. Однак усі КТ-знімки було змінено до однакової роздільної здатності — 120 × 120 × 120, щоб мати вхідні дані зі статичним розміром зображення. Функції втрат для обох мереж визначено як зважену функцію перехресної ентропії. Тривимірне зображення легенів пацієнта, подане на вхід мережі, наведено на рис. 13.13.

З метою пом'якшення проблеми незбалансованості набору даних використовували зважену функцію перехресної ентропії. Навчання проводили за допомогою алгоритму оптимізації *Adam* зі швидкістю навчання 0,0001 та 0,00004 для *C3D-* та *3D-DenseNet* відповідно.

Отримані дані свідчать, що навчені мережі не показують хороших результатів: *C3D* має точність 0,65, *3D-DenseNet* — 0,71. Це пов'язано з тим, що ознаки, за якими можна розпізнати рак легенів, дуже складні. Для того щоб НМ могла вивчити їх наскрізно, набори даних мають містити велику кількість анотованих даних. Проблему можна вирішити, розділивши процес навчання на кілька етапів.

Підсумовування результатів. На додачу до описаних експериментів наведемо кілька результатів з інших публікацій, див. табл. 13.1. Відзначимо перш за все результати з роботи [13.26], яка виграла конкурс *Data Science Bowl* у 2017 р. У цій роботі було навчено двоетапну систему виявлення раку легенів і досягнуто точності на рівні людини на наборі даних *DSB* 2017. Нейронна мережа *DeepLung* [13.26] розширює ідею попередньої роботи і покращує точність системи виявлення завдяки оптимізації архітектури нейронної мережі. Інша робота [13.31] представила чотириетапний підхід (див. рис. 13.9) з можливістю перевірки злоякісних змін легеневих вузлів між попереднім і поточним КТ-скануваннями. У цій роботі було досягнуто вищої точності (0,91) порівняно із середньостатистичним рентгенологом. Усі експерименти, описані в цьому розділі, та результати з інших публікацій дають глибше, повніше розуміння сучасних тенденцій побудови системи виявлення раку легенів. Узагальнені результати наведено в табл. 13.1.

Висновки

Останні дослідження в галузі автоматизованого виявлення раку легенів засвідчують значний прогрес. Сучасні системи на основі глибокого навчання вже досягають точності, співставної з рівнем досвідченого радіолога, а іноді — й перевищують його, особливо на стандартизованих на-

борах даних. Більше того, деякі з найкращих моделей демонструють високу продуктивність навіть за умов обмеженої кількості даних — що є типовою ситуацією для медичної діагностики.

Результати експериментальних досліджень указують, що для досягнення високої точності виявлення раку легенів доцільно використовувати підходи, які враховують тривимірну природу КТ-зображень. Це дає змогу системі витягувати просторову інформацію, критично важливу для точного розпізнавання злоякісних утворень. Використання двовимірних згорткових нейронних мереж (*2D-CNN*) у поєднанні з рекурентними нейронними мережами (*RNN*) демонструє задовільні результати і дає змогу повторно використовувати вже навчені 2D-моделі, наприклад, на основі *ImageNet*. Проте такий підхід поступається багатоступеневим системам на основі *3D-CNN*, які здатні гнучко адаптуватися до структури завдання.

Причина цього полягає в тому, що підхід *2D-CNN + RNN* менш гнучкий і не дає можливості ефективно розподіляти задачу на окремі етапи обробки, що підвищує вимоги до обсягу анотованих медичних даних — ресурсу, який є обмеженим і дорогим у медичній сфері [13.28].

Найсучасніші системи виявлення раку легенів базуються на конвеєрному підході з використанням 3D-згорткових нейронних мереж. Такі системи обробляють КТ-знімки як повноцінні тривимірні об'єкти, що дає змогу точно аналізувати просторову структуру тканин легенів. Як правило, завдання поділяється на окремі етапи: сегментація, класифікація та визначення ймовірності злоякісності виявленого вузла. При цьому провідні моделі також інтегрують додаткові компоненти, здатні враховувати глобальний або часовий контекст, що істотно покращує якість виявлення.

Гнучкість архітектури таких систем — можливість додавання нових модулів, які використовують додаткову інформацію для прийняття рішення — підвищує їхню надійність, знижує залежність від великих обсягів навчальних даних і забезпечує точність, порівнянну з результатами фахівців-радіологів. Отже, використання 3D-архітектур із модульною побудовою є найбільш перспективним напрямом розвитку автоматизованих систем ранньої діагностики раку легенів.

СПИСОК ЛІТЕРАТУРИ

13.1. He K., Zhang X., Ren S., Sun J. Deep Residual Learning for Image Recognition. *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*. 2016. P. 770—778. https://doi.org/10.1007/s00330-012-2627-7

13.2. Huang G., Liu Z., v. d. Maaten L., Weinberger K.Q. Densely Connected Convolutional Networks. *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*. 2017. P. 2261—2269. https://doi.org/10.1109/CVPR.2017.243

13.3. Chollet F. Xception: Deep Learning with Depthwise Separable Convolutions. *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*. 2017. P. 1800—1807. https://doi.org/10.1109/CVPR.2017.195

13.4. Dietterich T.G., Lathrop R.H., Lozano-Pérez T. Solving the Multiple Instance Problem with Axis-Parallel Rectangles. *Artificial Intelligence*. 2002. Vol. 89 (1—2). P. 31—71. https://doi.org/10.1016/S0004-3702(96)00034-3

13.5. Zhu W., Lou Q., Vang Y.S., Xie X. Deep Multi-Instance Networks with Sparse Label Assignment for Whole Mammogram Classification. *Medical Image Computing and Computer Assisted Intervention — MICCAI 2017. Lecture Notes in Computer Science*. Springer, Cham, 2017. Vol. 10435.

13.6. Olah C. Understanding LSTM Networks. August 27, 2015. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

13.7. Goodfellow I., Bengio Y., Courville A. Deep Learning. MIT Press, 2016. 800 p. URL: http://www.deeplearningbook.org

13.8. Hammer B. On the Approximation Capability of Recurrent Neural Networks. *Neurocomputing*. 1998. Vol. 31. P. 107—123. URL: https://www.sciencedirect.com/science/article/pii/S0925231299001745

13.9. Schuster M., Paliwal K.K. Bidirectional Recurrent Neural Networks. *IEEE Transactions on Signal Processing*. 1997. Vol. 45. P. 2673—2681. URL: https://www.researchgate.net/publication/3316656_Bidirectional_recurrent_neural_networks

13.10. Graves A. Generating Sequences with Recurrent Neural Networks. 2013. URL: https://arxiv.org/abs/1308.0850

13.11. Chen J., Chaudhari N.S. Capturing Long-Term Dependencies for Protein Secondary Structure Prediction. Advances in Neural Networks — ISNN 2004. *Lecture Notes in Computer Science*. Vol. 3174. Berlin, Heidelberg: Springer, 2004. P. 494—500. URL: https://link.springer.com/chapter/10.1007/978-3-540-28648-6_79

13.12. Graves A. Supervised Sequence Labelling with Recurrent Neural Networks. Berlin, Heidelberg: Springer, 2012. 142 p. URL: https://link.springer.com/book/10.1007/978-3-642-24797-2

13.13. Hochreiter S., Schmidhuber J. Long Short-Term Memory. *Neural Computation*. 1997. Vol. 9. P. 1735—1780. URL: https://direct.mit.edu/neco/article/9/8/1735/6109/Long-Short-Term-Memory

13.14. Çukur T., Nishimoto S., Huth A.G., Gallant J.L. Attention during Natural Vision Warps Semantic Representation across the Human Brain. *Nature Neuroscience*. 2013. Vol. 16 (6). P. 763—770. URL: https://www.nature.com/articles/nn.3381

13.15. Olah C., Mordvintsev A., Schubert L. Feature Visualization. Distill, 2017. URL: https://distill.pub/2017/feature-visualization

13.16. Xu K., Ba J., Kiros R. et al. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. *Proceedings of the 32nd Int. Conf. on Machine Learning (ICML)*. 2015. P. 2048—2057. arXiv: https://arxiv.org/abs/1502.03044

13.17. Mnih V., Heess N., Graves A., Kavukcuoglu K. Recurrent Models of Visual Attention. *Advances in Neural Information Processing Systems (NIPS)*. 2014. P. 2204—2212. arXiv: https://arxiv.org/abs/1406.6247

13.18. Ba J., Mnih V., Kavukcuoglu K. Multiple Object Recognition with Visual Attention. *CoRR*. 2014. Vol. abs/1412.7755. URL: ar Xiv://arxiv.org/abs/1412.7755

13.19. Malinowski M., Doersch C., Santoro A., Battaglia P.W. Learning Visual Question Answering by Bootstrapping Hard Attention. *Proceedings of the 15th Europ. Conf. on Computer Vision (ECCV)*. 2018. P. 3—20. arXiv: https://arxiv.org/abs/1806.07468

13.20. Grewal M., Srivastava M.M., Kumar P., Varadarajan S. Radnet: Radiologist Level Accuracy Using Deep Learning for Hemorrhage Detection in CT Scans. *IEEE 15th Int. Symp. on Biomedical Imaging (ISBI 2018)*. 2017. P. 281—284. arXiv: https://arxiv.org/abs/1710.04934

13.21. Ardila D., Kiraly A.P., Bharadwaj S. et al. End-to-End Lung Cancer Screening with Three-Dimensional Deep Learning on Low-Dose Chest Computed Tomography. *Nature Medicine*. 2019. Vol. 25 (6). P. 954—961. URL: https://www.nature.com/articles/s41591-019-0447-x

13.22. Tran D., Bourdev L., Fergus R., Torresani L., Paluri M. Learning Spatiotemporal Features with 3D Convolutional Networks. *IEEE Int. Conf. on Computer Vision (ICCV)*. 2015. P. 4489—4497. URL: https://www.doi.org/10.1109/ICCV.2015.510

13.23. Karpathy A., Toderici G., Shetty S., Leung T., Sukthankar R., Fei-Fei L. Large-scale Video Classification with Convolutional Neural Networks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2014. P. 1725—1732. https://doi.org/10.1109/CVPR.2014.223

13.24. Bui T.D., Shin J., Moon T. 3D Densely Convolutional Networks for Volumetric Segmentation. 2017. arXiv:https://arxiv.org/abs/1709.03199

13.25. Liao F., Liang M., Li Z., Hu X., Song S. Evaluate the Malignancy of Pulmonary Nodules Using the 3D Deep Leaky Noisy-or Network. *IEEE Transactions on Neural Networks and Learning Systems*. 2019. Vol. 30 (11). P. 3484—3495. https://doi.org/10.1109/TNNLS.2019.2892409

13.26. Zhu W., Liu C., Fan W., Xie X. DeepLung: Deep 3D dual path nets for automated pulmonary nodule detection and classification. *Proceedings — 2018 IEEE Winter Conference on Applications of Computer Vision, WACV*. 2018, V. 2018-Janua. P. 673—681. https://doi.org/10.1109/WACV.2018.00079

13.27. Milletari F., Navab N., Ahmadi S.-A. V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation. *3D Vision (3DV), 2016 Fourth International Conference on 3D Vision (3DV)*. 25—28 Oct. 2016. P. 565—571. https://doi.org/10.1109/3DV.2016.79

13.28. Ren S., He K., Girshick R., Sun J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2017. Vol. 39 (6). P. 1137—1149. https://doi.org/10.1109/TPAMI.2016.2577031

13.29. He K., Gkioxari G., Dollar P., Girshick R. Mask R-CNN. *Proceedings of the IEEE Int. Conf. on Computer Vision*. Vol. 2017-October. P. 2980—2988. https://doi.org/10.1109/ICCV.2017.322

13.30. Neroladaki A., Botsikas D., Boudabbous S., Becker C.D., Montet X. Computed tomography of the chest with model-based iterative reconstruction using a radiation exposure similar to chest X-ray examination: Preliminary observations. *European Radiology*. 2013. Vol. 23 (2). P. 360—366. https://doi.org/10.1007/s00330-012-2627-7

13.31. American Lung Association. What Are the Types of Lung Cancer? March 19, 2025. URL: https://www.lung.org/lung-health-diseases/lung-disease-lookup/lung-cancer/symptoms-diagnosis

13.32. Setio A.A.A., Traverso A., de Bel T. et al. Validation, comparison, and combination of algorithms for automatic detection of pulmonary nodules in computed tomography images: The LUNA16 challenge. *Medical Image Analysis*. 2017. Vol. 42. P. 1—13. https://doi.org/10.1016/j.media.2017.06.015

13.33. Setio A. A. A., Traverso A., de Bel T., Berens M. S., van den Bogaard C., Cerello P. et al. A benchmark for lung nodule detection in low-dose CT scans: LUNA16. *Medical Image Analysis*., 2017. 42. P. 1—13. https://doi.org/10.1016/j.media.2017.06.005

13.34. MacMahon H., Naidich D.P., Goo J.M. et al. Guidelines for management of incidental pulmonary nodules detected on CT images: From the Fleischner Society 2017. *Radiology*. 2017. Vol. 284 (1). P. 228—243. URL: https://pubs.rsna.org/doi/full/10.1148/radiol.2017161659

13.35. Fiorio C., Gustedt J. Two linear time Union-Find strategies for image processing. *Theoretical Computer Science*. 1996. Vol. 154. P. 165—181. https://doi.org/10.1016/0304-3975(94)00262-2

13.36. Ronneberger O., Fischer P., Brox T. U-Net: Convolutional Networks for Biomedical Image Segmentation. *Medical Image Computing and Computer-Assisted Intervention — MICCAI 2015*. Lecture Notes in Computer Science. Vol. 9351. Cham: Springer, 2015. P. 234—241. https://doi.org/10.1007/978-3-319-24574-4_28

13.37. Zaychenko Y., Hamidov G., Chapaliuk B. The Application of CNN and Hybrid Networks in Medical Images Processing and Cancer Classification. Cambridge Scholars Publishing. 2023. 138 p. URL: https://www.cambridgescholars.com/product/978-1-5275-1539-0

РОЗДІЛ 14

СИСТЕМИ ОБРОБКИ ПРИРОДНОЇ МОВИ

14.1. Базові засади обробки природної мови

14.2. Головні завдання і сфери застосування *NLP*

14.3. Як працюють моделі обробки природної мови?

Висновки

Список літератури

14.1. Базові засади обробки природної мови

Головні тенденції розвитку систем природної мови

Обробка природної мови (*Natural Language Processing, NLP*) — це дисципліна, яка ґрунтується на комп'ютерних науках і лінгвістиці, зосереджуючись на взаємодії між комп'ютерами та людськими (природними) мовами. Метою *NLP* є розроблення алгоритмів і моделей, що дають змогу комп'ютерам розуміти, інтерпретувати та генерувати мову людини. Практичне впровадження *NLP* реалізується у вигляді складних програмно-технічних систем, які формуються на основі взаємопов'язаних моделей, методів і технологій, як-от генератори текстів, що створюють зв'язні есе; чатботи, здатні імітувати реакції людини на запити; програми для перетворення тексту на зображення, які створюють фотореалістичні картини на основі описів природною мовою, та багато іншого [14.1].

Обробка природної мови є ключовим напрямом розвитку сучасного штучного інтелекту (ШІ). Завдяки цьому ШІ стає більш ефективним у багатьох сферах діяльності людини: комунікації, бізнесі, охороні здоров'я, освіті, автоматизованій підтримці клієнтів, аналізі настроїв, перекладі текстів тощо.

Сучасні системи *NLP* здатні генерувати складну прозу на різні теми, створювати музичні композиції та відеоматеріали, підтримувати зв'язні розмови з людьми. Наприклад, *Google* використовує *NLP* для вдосконалення результатів пошуку, а соціальні мережі, такі як *Facebook,* застосовують *NLP* для виявлення і фільтрації антигуманних і антисоціальних текстів.

Системи *NLP* швидко розвиваються, інтегруються у повсякденне життя як надійні помічники, стаючи дедалі складнішими. Проте попереду ще великі перспективи. Сьогодні *NLP* притаманні певні недосконалості: схильність до упередженості, непослідовність або хаотичні дії. Попри це інженери машинного навчання мають багато можливостей для вдосконалення *NLP,* роблячи його дедалі важливішим для функціонування суспільства.

Основними концепціями, що застосовуються для аналізу, розуміння та генерації людської мови комп'ютерами, є метод, модель і алгоритм.

• ***Метод*** — це підхід або процедура для вирішення завдань у *NLP.* Він охоплює алгоритми, стратегії оброблення, а також моделі. Метод задає послідовність дій або правила для роботи з текстовими даними з метою досягнення конкретної мети.

• ***Модель*** — це математична або алгоритмічна конструкція для прогнозування ймовірності появи певної послідовності слів у тексті. Модель навчається на великих масивах даних і використовується для генерації текстів, аналізу запитів, синтаксису, семантики тощо.

• ***Алгоритм*** — це визначена послідовність конкретних кроків або процедур для навчання моделі, її оптимізації та застосування. Алгоритм визначає, які саме дані обробляються і як навчається модель.

Метод є основою для створення моделей, тоді як модель реалізує ці методи у вигляді алгоритмів або архітектур для виконання конкретних завдань.

Отже: 1) методи — це загальні концепції або стратегії виконання задач *NLP*; 2) моделі — це практичні реалізації методів; 3) алгоритми — це послідовності кроків для реалізації методу і побудови моделі.

Відмінність між методами, моделями та алгоритмами відображає різні рівні абстракції: методи — високий рівень, моделі — середній, алгоритми — операційний.

Разом із тим грань між моделлю і методом не завжди чітка. З розвитком *NLP* окремі підходи, які раніше вважалися методами, стали частиною комплексних моделей. У науковій літературі та практиці терміни «модель» і «метод» іноді використовують у різних контекстах, що ускладнює їх розмежування.

Надалі розглядатимемо моделі, методи й алгоритми оброблення природної мови з урахуванням їхніх зазначених особливостей і взаємозв'язку між ними.

Розглянемо головні типи, функції та тенденції розвитку мовних моделей. Є такі головні типи мовних моделей:

1. ***N*-грамові моделі —** це статистичні моделі, що прогнозують наступне слово в послідовності, ґрунтуючись на попередніх *n* словах. Вони аналізують частоти появи підрядних послідовностей слів (*n-грам*) у текстовому корпусі і використовують ці частоти для передбачення наступних слів у реченні. Модель прогнозує ймовірність появи слова на основі попередніх $n - 1$ слів, де n визначає кількість слів, що враховуються для передбачення.

2. **Моделі на основі нейронних мереж** — це моделі, які використовують нейронні мережі для аналізу, розуміння та генерації тексту природною мовою. Вони навчаються на великих обсягах текстових даних, формуючи складні подання мовних структур. Як правило, для роботи з послідовностями слів застосовуються глибокі нейронні мережі.

3. **Трансформерні мовні моделі** — це моделі оброблення природної мови, що використовують механізм самоуваги. Цей підхід дає змогу ефективно обробляти послідовності слів, враховувати контекст на різних рівнях, незалежно від довжини тексту, і виконувати такі завдання, як переклад, текстове порозуміння, генерація тексту тощо.

Головні функції мовних моделей:

1. ***Прогнозування слів.*** Передбачення наступного слова або послідовності слів на основі попереднього контексту.

2. ***Аналіз семантики та синтаксису.*** Розпізнавання значення слів та їхніх взаємозв'язків у реченні.

3. ***Генерація тексту.*** Створення нових текстів на основі заданого контексту або теми.

4. ***Мовний переклад.*** Переклад тексту іншою мовою.

5. ***Розпізнавання та класифікація тексту.*** Визначення категорії тексту, розпізнавання іменованих сутностей, аналіз настроїв тощо.

Узагальнюючи висновки, зроблені в [14.2], можна виокремити такі головні тенденції розвитку моделей *NLP* на сучасному етапі, які відображають різні аспекти людської діяльності, роблячи її більш зручною, ефективною та етично відповідальною:

1. *Зростання масштабів та складності моделей.* Моделі *NLP* стають дедалі більшими і складнішими, досягаючи високої точності у завданнях генерації тексту, перекладу та аналізу великих обсягів тексту.

2. *Розширення мультимовних можливостей.* Моделі здатні працювати з багатьма мовами, що дає можливість створювати універсальні рішення для глобальних задач у різних сферах людської діяльності.

3. *Ефективність і стійкість.* Зростає увага до оптимізації енергоспоживання і навчання моделей. Використовуються методи дистиляції моделей, квантової обробки тощо, що зменшує витрати ресурсів.

4. *Інтеграція з іншими технологіями. NLP* усе більше інтегрується з комп'ютерним зором, системами рекомендацій та іншими ШІ-технологіями, створюючи багатофункціональні та адаптивні системи.

5. *Персоналізація.* Системи *NLP* стають більш адаптивними до контексту та уподобань користувачів, що підвищує ефективність і точність взаємодії.

6. *Автоматизація робочих процесів. NLP* активно використовується для автоматизації рутинних завдань у таких сферах, як право, охорона здоров'я, фінанси тощо, що підвищує ефективність і точність оброблення текстових даних.

7. *Покращення генеративних моделей.* Сучасні моделі демонструють значний прогрес у генерації тексту та візуального контенту, розширюючи можливості їх використання у творчих і комерційних сферах.

8. *Етичні стандарти і прозорість.* Зростає увага до етичного підходу в розробленні та використанні *NLP*, зокрема до дебайзингу (усунення упередженостей у моделях), забезпечення приватності даних і прозорості роботи моделей.

9. *Мультизадачність і універсальність.* З'являються моделі, здатні виконувати кілька завдань одночасно, наприклад переклад, аналіз настроїв і класифікацію тексту, що дає змогу використовувати одну модель для різних завдань.

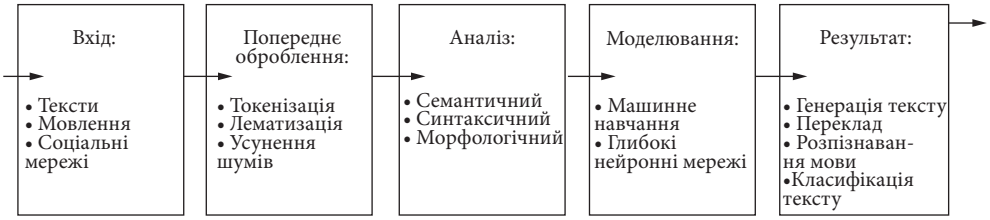


Рис. 14.1. Алгоритм роботи NLP як складної адаптивної системи

Зазначені тенденції відображають загальні напрями розвитку *NLP,* акцентуючи увагу на масштабуванні, ефективності, етичних стандартах та інтеграції з іншими технологіями. Це сприяє ширшому й ефективнішому застосуванню *NLP* у різних галузях.

NLP як складна адаптивна система

Сукупність методів, моделей та алгоритмів для оброблення природної мови можна розглядати як складну адаптивну систему з точки зору теорії систем. Вона характеризується взаємодією численних компонентів, динамічністю, нелінійністю та наявністю зворотних зв'язків. Система складається з різних компонентів, які взаємодіють між собою для аналізу, розуміння та генерації мови людини. Алгоритм роботи цієї системи наведено на рис. 14.1.

Взаємодія компонентів складної адаптивної системи

1. ***Отримання вхідних даних (Receiving Input Data).*** Система отримує необроблені дані у вигляді тексту, мовлення або інформації з інших джерел (наприклад, із соціальних мереж, документів). Ці дані подаються на вхід блоку попереднього оброблення.

2. ***Попереднє оброблення (Preprocessing).*** На цьому етапі вхідні дані очищаються і готуються до аналізу. Процес охоплює:

- токенізацію (розбиття тексту на окремі слова або фрази);
- лематизацію (перетворення слів до їх базових форм);
- усунення шумів (видалення зайвих символів, нормалізацію тексту).

Оброблені дані передаються на наступний етап.

3. ***Аналіз (Analysis).*** На цьому етапі дані піддають глибокому аналізу, який містить:

- *семантичний аналіз* — розуміння значень слів і фраз у контексті;
- *синтаксичний аналіз* — визначення граматичної структури речень;
- *морфологічний аналіз* — аналіз форми слів залежно від контексту.

Результати аналізу передаються до блоку моделювання.

4. ***Моделювання (Modeling).*** На цьому етапі застосовують алгоритми машинного навчання або глибокі нейронні мережі. Вони виконують оброблення великих обсягів тексту, розпізнають мовні шаблони і приймають рішення на основі проаналізованих даних. Моделювання може охоплювати:

- розпізнавання тексту;
- прогнозування наступних слів;
- класифікацію тексту тощо.

5. ***Формування вихідного результату (Formation of the Output Result).*** На заключному етапі результати роботи системи надають користувачу або іншій системі. Це може бути:

- згенерований текст (створений на основі вхідних даних);
- перекладений текст (результат автоматичного перекладу);
- результат розпізнавання мови (перетворення мовлення на текст);
- результат класифікації тексту (визначення категорії чи теми).

Отже, *NLP* є складною системою, яка адаптується до різних типів вхідних даних і вимог користувача. Усі її блоки взаємопов'язані й виконують послідовні кроки для досягнення мети.

Головні характеристики NLP як складної адаптивної системи

1. ***Багатокомпонентність.*** *NLP* містить різні компоненти, як-от морфологічний аналіз, синтаксичний аналіз, семантичний аналіз, розпізнавання мови, генерація тексту тощо. Кожний компонент є окремою підсистемою.

2. ***Взаємодія компонентів.*** Наприклад, успішність синтаксичного аналізу залежить від результатів морфологічного аналізу, а розуміння семантики неможливе без попереднього синтаксичного розбору.

3. ***Нелінійність.*** Невеликі зміни в одному компоненті можуть суттєво вплинути на загальний результат обробки.

4. ***Динамічність.*** *NLP* постійно адаптується до нових даних і завдань, що потребує оновлення алгоритмів і методів.

5. ***Зворотний зв'язок.*** Моделі машинного навчання постійно вдосконалюються на основі нових даних і відгуків користувачів.

6. ***Складність.*** Різноманітність мовних структур, контекстів і завдань потребує інтеграції численних підходів і методів.

Дослідження останніх років принесли революцію у здатність комп'ютерів розуміти людські мови, мови програмування, а також біологічні та хімічні послідовності, наприклад, ДНК і білкові структури. Новітні моделі ШІ аналізують значення тексту і генерують осмислені результати.

На відміну від низки теоретичних розділів ШІ *NLP* є інженерною дисципліною, спрямованою на створення технологій для виконання завдань *NLP.* Процеси оброблення природної мови поділяються на дві взаємопов'язані складові:

1) *розуміння природної мови (Natural Language Understanding, NLU)* зосереджене на семантичному аналізі тексту;

2) *генерація природної мови (Natural Language Generation, NLG)*, що фокусується на автоматичному створенні тексту.

У поєднанні з технологіями розпізнавання мови *NLP* дає можливість перетворювати мовлення на текст і навпаки.

14.2. Головні завдання і сфери застосування *NLP*

NLP використовують для вирішення широкого спектра завдань, пов'язаних з обробленням природної мови, у тому числі автоматичними перекладами, відповідями на запитання, класифікацією тексту, спілкуванням з користувачами тощо. Нижче наведено приклади важливих завдань, які вирішує *NLP*, та основні сфери їх застосування [14.1].

Аналіз настроїв — це процес оцінювання і класифікації емоційного змісту тексту. Вхідними даними для моделі класифікації настроїв є текстовий фрагмент, а вихідним сигналом — ймовірність того, що виражене почуття є позитивним, негативним або нейтральним. Ця ймовірність визначається двома підходами:

1. **Перший підхід** ґрунтується на згенерованих вручну ознаках, як-от *N*-грамові слова та *TF-IDF:*

• *N-грамові слова* — це послідовності з *N* слів (або символів) з тексту, які використовуються для розпізнавання контексту та структури мови. Вони допомагають враховувати зв'язки між словами й прогнозувати наступне слово в тексті. Наприклад, у біграмовій моделі для слів «природної мови» наступним прогнозованим словом може бути «аналіз»;

• *TF-IDF (Term Frequency-Inverse Document Frequency)* — це статистичний показник, що оцінює важливість слова в документі відносно до всієї колекції документів.

Складові TF-IDF:

• *TF (Term Frequency)* — частота терміна у документі, що визначається як відношення кількості поява слова *t* у документі *d* до загальної кількості слів у *d*.

• *IDF (Inverse Document Frequency)* — міра, яка зменшує вагу загальних слів і підвищує вагу рідкісних. Розраховується як логарифм відношення кількості всіх документів до кількості документів, що містять слово *t*.

• *Формула для обчислення TF-IDF: TF-IDF(t,d) = TF(t,d) × IDF(t).*

Приклад. Розглянемо три документи:
0) «Я люблю програмування»;
1) «Програмування є цікавим»;
2) «Я люблю читати про програмування».
TF для слова «програмування»:
• у першому документі 0,333;
• у другому 0,333;
• у третьому 0,2.
IDF = log(3/3) = 0. Отже, *TF-IDF* = 0.

Завдання, що вирішуються за допомогою TF-IDF.

1. *Інформаційний пошук.* Ранжування документів за релевантністю до запиту.

2. *Текстова класифікація.* Використання *TF-IDF* як ознак для алгоритмів машинного навчання.

3. *Кластеризація тексту.* Групування подібних документів у кластери.

2. **Другий підхід** ґрунтується на використанні моделей глибокого навчання, які розпізнають короткострокові та довгострокові залежності в тексті, покращуючи точність і ефективність обробки. Ці моделі здатні значно підвищувати точність і ефективність багатьох завдань під час оброблення природної мови, що робить їх незамінними інструментами. Вони особливо корисні для завдань, де контекст і порядок слів мають велике значення.

Основні моделі глибокого навчання для *NLP*

1. ***Рекурентні нейронні мережі (Recurrent Neural Network, RNN)*** здатні обробляти послідовні дані, зберігаючи інформацію про попередні стани через приховані шари. Вони добре підходять для таких завдань як машинний переклад або аналіз настроїв, у яких важливий порядок слів. Недоліком цього типу мереж є те, що вони можуть мати проблеми із запам'ятовуванням довгострокових залежностей у зв'язку з проблемою зникнення градієнта.

2. ***Нейронні мережі з тривалою короткочасною пам'яттю (Long Short-Term, LSTM).*** Це спеціальний тип мереж *RNN,* розроблений для подолання проблеми зникнення градієнта. Вони мають структури, відомі як комірки пам'яті, що можуть зберігати інформацію на тривалі періоди часу, що дає можливість моделі ефективно запам'ятовувати довгострокові залежності.

3. ***Гейтовані рекурентні блоки (Gated Recurrent Unit, GRU).*** Це тип рекурентних нейронних мереж (*RNN*), який є спрощеним варіантом *LSTM. GRU* застосовують для ефективного оброблення послідовних даних, зокрема тексту, і подолання деяких обмежень, традиційних для *RNN,* як-от проблема зникнення градієнта. Ці мережі мають менше параметрів, що забез-

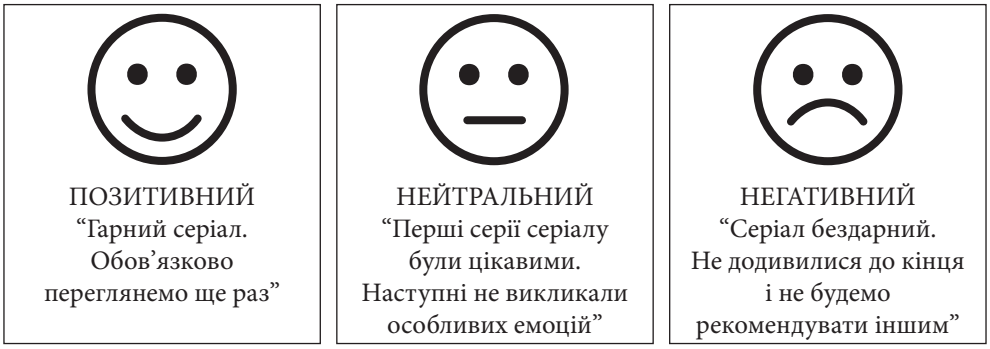


Рис. 14.2. Методи аналізу настроїв, які дають можливість класифікувати емоційний зміст поданого тексту

печує швидше навчання, але також можуть бути менш гнучкими ніж *LSTM*.

4. ***Трансформери є основою сучасних моделей NLP.*** Вони використовують механізм уваги, що дає можливість моделі фокусуватися на важливих частинах вхідного тексту, незалежно від відстані між словами. Трансформери добре виконують завдання, які потребують розуміння як короткострокових, так і довгострокових залежностей [14.3].

У практичній діяльності аналіз настроїв людей використовується для різних потреб, наприклад, для визначення реакції суспільства на ті чи інші події, в соціальних дослідженнях (рис. 14.2) [14.1]; для класифікації відгуків клієнтів на різних онлайн-платформах у бізнесі; для нішевих програм, з метою виявлення ознак психічних захворювань за онлайн-коментарями, у медицині та інших сферах.

Класифікація токсичності тексту

Метою цього завдання є аналіз настроїв користувачів, які недоброзичливо спілкуються в соціальних мережах, з визначенням токсичних намірів та класифікацією категорій недоброзичливості, таких як погрози, образи, непристойності та мова ненависті до певних ідентичностей. Вхідними даними для такої моделі є текст, а результатом — ймовірність кожного класу токсичності.

Сфери застосування. Моделі класифікації токсичності використовують для модерації та покращення онлайн-спілкування, зокрема для:

- видалення образливих коментарів;
- виявлення мови ворожнечі;
- сканування документів на предмет наклепу.

Приклад. У *Facebook* застосовано автоматизовані системи для блокування мови ворожнечі. Згідно з позицією компанії, запобігання вірусному поширенню ненависті, страху та недовіри через соціальні мережі стає дедалі важливішим. Експерти встановили зв’язок між мовою ворожнечі в Ін-

тернеті та реальним насильством. Для цього у *Facebook* створено детектор мови, здатний розпізнавати антигуманні висловлювання. Система вивчає до 10 різних аспектів кожної публікації (текст, зображення, відео, коментарі, зовнішній контекст з Інтернету). Моделі *NLP* виділяють із цих елементів вектори ознак, об'єднують їх і класифікують публікацію як доброякісну або таку, що містить ознаки ворожнечі.

Машинний переклад

Машинний переклад автоматизує процес перекладу текстів іншими мовами. Вхідними даними для такої моделі є текст мовою оригіналу, а вихідними — текст цільовою мовою. Наприклад, *Google Translate* є, мабуть, найвідомішим і найпопулярнішим додатком для машинного перекладу. Його робота ґрунтується на кількох методах і нейронних мережах, які забезпечують якісний автоматизований переклад текстів.

Основні методи і моделі, які застосовують для машинного перекладу

1. ***Рекурентні нейронні мережі (Recurrent Neural Network, RNN).*** Це були перші нейронні мережі, які *Google Translate* використовував для машинного перекладу. Вони виявилися корисними для роботи з послідовними даними, як-от текст.

2. ***Модель секвенція до секвенції (Sequence-to-Sequence Model, Seq2Seq)*** містить дві основні частини — кодер, який обробляє вхідний текст і перетворює його на вектор, і декодер, що використовує цей вектор для генерації вихідного тексту.

3. ***Механізм уваги (Attention Mechanism)*** удосконалює *Seq2Seq* модель, даючи змогу декодеру фокусуватися на різних частинах вхідного речення під час генерації кожного слова вихідного тексту. Це значно поліпшує якість перекладу, особливо для довгих речень.

4. ***Трансформери (Transformers) —*** сучасна архітектура нейронних мереж, яка замінила *RNN* у багатьох завданнях машинного перекладу. Вона ґрунтується виключно на механізмі уваги, що дає можливість обробляти послідовності даних паралельно, істотно підвищуючи продуктивність та ефективність.

5. ***Нейронний машинний переклад (Neural Machine Translation, NMT).*** *Google Translate* перейшов на *NMT* у 2016 р., що значно покращило якість перекладу. Основними компонентами *NMT* є такі [14.3]:

• *кодер-декодер архітектура.* Кодер перетворює вхідний текст на векторне зображення, яке відображає його зміст у компактному форматі. Декодер використовує цей вектор для створення перекладеного тексту;

• *трансформери* забезпечують механізм уваги, що дає можливість моделі зосереджуватись на важливих частинах вхідного тексту. Це підвищує точність передавання контексту та змісту вхідного тексту. Трансформери також обробляють слова одночасно, а не послідовно, що робить їх ефективнішими;

• *навчання на великих даних.* Моделі *NMT* тренуються на великих корпусах текстів, що сприяє поліпшенню якості перекладу на основі численних прикладів [14.3].

Ключові переваги *NMT* порівняно з традиційними методами машинного перекладу:

• *контекстуальність* — враховує контекст усього речення або навіть абзацу, зберігаючи зміст та граматику. Традиційні методи часто ігнорували контекст, що призводило до втрати сенсу;

• *гнучкість і точність* — адаптується до різних стилів і тональностей тексту, покращуючи точність перекладу;

• *зменшення помилок* — знижує ймовірність типових помилок, таких як неправильний порядок слів або граматичні помилки;

• *розуміння довгих залежностей* — завдяки трансформерам краще опрацьовує довгі залежності в тексті, зберігаючи зв'язки між різними частинами речення;

• *адаптація до нових доменів* — моделі можуть адаптуватися до специфічних тематик (наприклад, медичний або технічний переклад) за допомогою додаткового навчання;

• *ідентифікація мови* — автоматично визначає мову тексту, що полегшує подальше оброблення.

Отже, переваги *NMT* роблять її однією з найефективніших технологій сучасного машинного перекладу. Її застосовують для поліпшення комунікації між людьми на платформах соціальних мереж, як-то *Facebook* чи *Skype,* та в багатьох інших сферах.

Розпізнавання іменованих сутностей

Метою цього завдання є витягування сутностей із тексту у заздалегідь визначених категоріях, як-от особисті імена, організації, місця, важливі наративи тощо. Вхідними даними для такої моделі зазвичай є текст, а вихідними — іменовані сутності разом із їх початковими та кінцевими позиціями.

Сфери застосування. Розпізнавання іменованих сутностей є важливим у таких програмах:

• узагальнення новинних статей;

• боротьба з дезінформацією;

• розвідка з відкритих джерел (*Open Source Intelligence, OSINT*).

Приклад. Розвідка з відкритих джерел активно використовує методи та моделі оброблення природної мови (*NLP*) для аналізу великих обсягів текстової інформації, зібраної з публічних джерел, таких як соціальні мережі, новини, вебсайти та форуми. Завдяки досягненням у сфері *NLP OSINT* отримала можливість ефективно аналізувати, структурувати та інтерпретувати неструктуровані текстові дані, перетворюючи їх на цінну інформацію для прийняття рішень.

Основою оброблення текстових даних є токенізація і нормалізація тексту, що дає можливість розбивати текст на окремі слова чи речення, уніфіковувати форми слів та спрощувати подальший аналіз. Розпізнавання іменованих сутностей, наприклад імен, організацій, локацій чи дат, реалізується завдяки моделям, типу *BERT* або *spaCy.* Це дає змогу *OSINT* визначати ключових учасників подій та важливі об'єкти, що згадуються в текстах. Класифікація тексту є ще одним важливим напрямом, що забезпечує автоматичне сортування текстових документів за категоріями, наприклад, для аналізу тематик новин чи ідентифікації джерел загроз. Аналіз тональності тексту дає змогу визначити емоційне забарвлення повідомлень, що може бути критично важливим для моніторингу громадської думки або виявлення загрозливих коментарів.

Сучасні *NLP* моделі також дають можливість ефективно здійснювати пошук релевантної інформації в масиві текстів, використовуючи методи семантичного пошуку та векторизації тексту. Завдяки моделям типу *Word2Vec, GloVe* чи *Sentence-BERT* тексти можна перетворювати на вектори, що зберігають їхні семантичні властивості, полегшуючи пошук подібних фрагментів інформації. Переклад текстів з рідкісних мов або багатомовне оброблення даних забезпечуються моделями на кшталт *MarianMT* чи *Google Translate API*, що розширює можливості *OSINT* для аналізу даних з різних мовних джерел.

Додатково *OSINT* використовує моделі для генерації тексту та відповіді на запитання. Завдяки моделям типу *GPT-3* чи *T5* аналітики можуть автоматично генерувати резюме текстів або отримувати чіткі відповіді на специфічні запити. Темо-моделювання, реалізоване за допомогою *LDA* чи *BERTopic*, дає змогу визначати основні теми у великих обсягах текстів, що значно полегшує розуміння загальної картини подій. Особливу увагу в *OSINT* приділяють виявленню фейкових новин і дезінформації. Для цього застосовують сучасні класифікаційні моделі, які можуть аналізувати джерела і зміст текстів, визначаючи їхню достовірність.

Серед основних переваг *NLP* для *OSINT* слід відзначити швидкість оброблення великих обсягів інформації, точність аналізу та універсальність. Ці методи дають можливість автоматизувати процеси моніторингу, аналізу та узагальнення даних, роблячи *OSINT* потужним інструментом у сферах безпеки, бізнесу, політики та досліджень. Завдяки сучасним *NLP* моделям *OSINT* може не лише спрощувати оброблення текстових даних, а й забезпечувати високу якість аналітики, що є критично важливим у стратегічному плануванні та прийнятті рішень.

Виявлення спаму

Виявлення спаму є поширеним завданням бінарної класифікації в обробці природної мови (*NLP*). Метою є класифікація електронних листів як спам чи не спам. Детектори спаму приймають на вхід текст листа разом із метаданими, такими як заголовок та ім'я відправника, і визначають імовірність того, що лист є спамом. Наприклад, постачальники послуг електронної пошти, як-от *Gmail,* використовують спеціальні моделі для виявлення небажаних електронних листів та їх автоматичного переміщення до папки «Спам».

Методи і моделі для виявлення спаму

1. ***Байєсові класифікатори***, зокрема наївні байєсові класифікатори, які застосовують теорему Байєса для обчислення ймовірності того, що повідомлення є спамом, на основі слів і фраз у тексті.

2. ***Фільтри на основі правил*** використовують заздалегідь визначені правила, наприклад виявлення ключових слів або фраз, характерних для спаму.

3. ***Алгоритми на основі частотного аналізу (TF-IDF)*** оцінюють важливість слів у документі на основі частоти їх появи в цьому документі та рідкості в загальному корпусі текстів.

4. ***Методи на основі дерев рішень,наприклад***, *Random Forest* — потужний алгоритм машинного навчання, що ґрунтується на методі ансамблів і використовує набір рішень дерев (*decision trees*) для розв'язання задач класифікації, регресії та інших задач машинного навчання. У контексті виявлення спаму він класифікує електронні листи як спам або не спам, аналізуючи різні ознаки повідомлення (наприклад, частоту певних слів, форматування, заголовок і метадані).

5. ***Підтримувальні векторні машини (SVM)*** використовують для побудови гіперплощини, яка максимально розділяє класи спаму і не спаму у багатовимірному просторі ознак.

6. ***Нейронні мережі та глибоке навчання:***

• *рекурентні нейронні мережі (RNN)* обробляють текст, враховуючи послідовність слів;

• *нейронні мережі типу LSTM (Long Short-Term Memory)* зберігають інформацію про послідовність даних на тривалий час;

• *трансформери*, наприклад *BERT (Bidirectional Encoder Representations from Transformers)* або GPT *(Generative Pre-Trained Transformer)*, забезпечують контекстуальне розуміння тексту для підвищення точності класифікації.

7. ***Комбіновані методи.*** Гібридні моделі поєднують кілька різних підходів для покращення результатів [14.1].

Моделі виправлення граматичних помилок

Ці моделі кодують граматичні правила для виправлення помилок у тексті. Завдання розглядається як перетворення однієї послідовності на іншу, де вхідними даними є граматично некоректне речення, а вихідними — виправлене. Наприклад, онлайн-платформа *Grammarly* [14.4], заснована в Україні у 2009 р., використовує технології штучного інтелекту для покращення письмового спілкування англійською мовою і надає рекомендації щодо граматики, орфографії, стилю та тону тексту, допомагаючи користувачам писати чіткіше й ефективніше. *Grammarly* використовує сучасні технології для перевірки граматики, стилю та орфографії тексту.

Головні компоненти Grammarly

1. ***Збір і попереднє оброблення даних:***
- введення тексту через вебдодаток або розширення браузера;
- очищення тексту, розбиття на речення і слова.

2. ***Морфологічний і синтаксичний аналіз:***
- токенізація тексту;
- визначення частин мови для кожного токену;
- створення дерева залежностей для граматичного аналізу речення.

3. ***Аналіз і виправлення помилок:***
- використання граматичних правил для виявлення поширених помилок;
- застосування моделей машинного навчання для виявлення складніших помилок;
- використання глибоких нейронних мереж для контекстуального розуміння.

4. ***Генерація пропозицій:***
- виправлення граматичних помилок;
- пропозиції для поліпшення стилю, спрощення речень і заміни слів.

5. ***Зворотний зв'язок і навчання моделі:***
- користувачі можуть приймати або відхиляти пропозиції, допомагаючи системі поліпшувати моделі.

6. ***Безпека і конфіденційність:***
- шифрування даних і дотримання політик конфіденційності, враховуючи те, що *Grammarly* дотримується суворих політик щодо збереження конфіденційності та коректного використання особистої інформації користувачів.

Тематичне моделювання

Тематичне моделювання виконується за допомогою інтелектуального аналізу корпусу документів, що дає змогу виявляти абстрактні теми у текстах. Вхідними даними для тематичної моделі є набір документів, а вихідними — список тем із відповідними словами для кожної теми, а також пропорційний розподіл цих тем у кожному документі.

Одним із найпопулярніших методів тематичного моделювання є **латентний розподіл Діріхле (*Latent Dirichlet Allocation, LDA*)**. Цей метод розглядає документ як суміш тем, а кожну тему — як набір слів. Основна ідея *LDA* полягає в тому, що документи представлені як випадкові комбінації тем, кожна з яких характеризується певним розподілом слів.

Сфери застосування:

- дослідження складних соціальних систем;
- бізнес-аналітика;
- юриспруденція (обґрунтування юридичних документів);
- інші сфери діяльності, які потребують аналізу текстових даних.

Приклад. Розглянемо корпус із трьох документів, у якому потрібно виявити дві теми. *LDA* може встановити, що:

- одна тема пов'язана зі словами «футбол», «матч», «гол»;
- інша — зі словами «музика», «концерт», «пісня».

Кожний документ буде описано як суміш цих тем:

- документ 1 — 70 % «спорт», 30 % «музика»;
- документ 2 — 10 % «спорт», 90 % «музика»;
- документ 3 — 50 % «спорт», 50 % «музика».

Отже, кожний документ містить слова, які переважно відповідають визначеним темам.

Генерування тексту

Генерування тексту, відоме також як **технологія генерації природної мови (*Natural Language Generation, NLG*)**, створює текст, подібний до написаного людиною. Ця технологія може бути налаштована для створення текстів у різних жанрах і форматах — твітах, блогах або навіть комп'ютерних кодах.

Методи генерування тексту. Генерування здійснюється за допомогою таких підходів:

- марковські процеси;
- нейронні мережі (*LSTM, BERT, GPT-3, GPT-4, LaMDA*) [14.5, 14.6];
- інші сучасні підходи.

Сфери застосування:

- автозаповнення;
- чатботи;

• автоматичне написання промов, доповідей, статей;

• створення рекламних текстів.

Алгоритм роботи:

1. ***Збір даних.*** Збирається великий обсяг текстових даних для тренування моделі. Джерела можуть охоплювати книги, статті, твіти, блоги тощо.

2. ***Попереднє оброблення даних.*** Дані очищуються, форматуються, видаляються зайві символи, виправляються помилки, текст приводиться до уніфікованого вигляду.

3. ***Тренування моделі на основі підготовлених даних.*** Є кілька різних підходів до тренування моделей. Головні з них основані на використанні марковських процесів, нейронних мереж на зразок довготривалої короткотермінової пам'яті (*LSTM*), двоспрямованих рекурентних нейронних мереж *(Bidirectional Encoder Representations from Transformers, BERT* [14.5]), генеративно попередньо навчених трансформерів *GPT-3, GPT-4, LaMDA* [14.6] та інших.

4. ***Генерування тексту.*** Після тренування модель створює текст на основі вхідних даних. Це може бути як простий текстовий рядок, так і складніші структури, наприклад абзаци чи цілі статті. Модель може бути адаптована для роботи в різних жанрах і стилях.

5. ***Пост-оброблення.*** Згенерований текст зазнає додаткового оброблення для поліпшення, стилю та виправлення можливих граматичних помилок. Це забезпечує подібність тексту до написаного людиною.

Автозаповнення в обробленні природної мови

Автозаповнення використовується для прогнозування наступного слова або фрази на основі попередньо введеного тексту. Ці моделі значно поліпшують зручність і швидкість набору тексту в різних додатках, таких як текстові редактори, електронна пошта, пошукові системи та месенджери.

Основні принципи роботи моделей автозаповнення

1. ***Збирання даних.*** Моделі навчаються на великих наборах текстових даних, які можуть містити книги, статті, вебсторінки, чати та інші текстові джерела. Це дає моделі можливість зрозуміти закономірності мови і зв'язки між словами.

2. ***Навчання моделі.*** Модель тренується на зібраних даних, навчаючись передбачати наступне слово або фразу на основі контексту. Використовуються різні алгоритми машинного навчання та глибокого навчання для створення точних прогнозів.

Генерування прогнозів. Коли користувач вводить текст, модель аналізує вже введені слова і на основі цього передбачає наступне слово або фразу. Прогнози можуть з'являтися у вигляді підказок, які користувач може вибрати для автозаповнення [14.7]

3. ***Технології і підходи, на яких заснована робота моделей автозаповнення:***

• ***Марківські ланцюги —*** найпростіші методи для автозаповнення, які дають можливість здійснювати прогнозування наступного слова, ґрунтуючись на ймовірностях переходу від одного слова до іншого. Разом з тим цей підхід має обмежену точність.

• ***N-Грамові моделі.*** В цих моделях використовуються послідовності з *N* слів для прогнозування наступного слова. Що більше значення *N*, то точніші прогнози, але зі зростанням *N* збільшуються також складність алгоритму і обсяги необхідних даних та обчислень.

• ***Рекурентні нейронні мережі (RNN)*** краще обробляють послідовності даних, зберігаючи контекст на кожному кроці. Вони можуть передбачати наступні слова на основі більш довгого контексту, що покращує точність.

• ***Довготривала короткотермінова пам'ять (LSTM).*** Це поліпшена версія *RNN*, яка долає проблему затухання градієнта і краще зберігає довготривалі залежності між словами.

• ***Трансформери.*** Сучасні моделі *GPT (Generative Pre-trained Transformer)* можуть ефективно обробляти великі послідовності тексту і генерувати високоточні прогнози. Вони тренуються на величезних обсягах даних і використовують механізм уваги для збереження контексту.

• ***Генеративні моделі типу GPT-3 та GPT-4.*** Ці моделі здатні генерувати текст, дуже подібний до тексту, написаного людиною. Вони можуть передбачати не тільки окремі слова, а й цілі фрази або речення на основі введеного контексту [14.7].

Чатботи в завданнях оброблення природної мови

Чатботи в обробленні природної мови можна розподілити на дві категорії, які різняться за складністю та метою використання. Обидві категорії спрямовані на поліпшення взаємодії між користувачами та системами:

1) ***чатботи, що здійснюють запит до бази даних.*** Ці чатботи використовують для отримання відповідей з наявної бази даних. Вони відповідають на запити користувачів, використовуючи заздалегідь визначені пари «запитання—відповідь». Основна функція таких чатботів — надавати точну інформацію на основі запитів користувачів.

2) ***чатботи, що генерують розмову.*** Ці чатботи здатні вести діалог із користувачем, імітуючи спілкування людей. Вони використовують складні алгоритми оброблення природної мови для створення природних відповідей. Прикладом є мовна модель *LaMDA (Language Model for Dialogue Applications)* від *Google,* яка веде розлогі розмови, надаючи відповіді, максимально наближені до людського стилю та емоційного забарвлення [14.6].

Основні аспекти роботи LaMDA:

1. ***Глибоке навчання.*** *LaMDA* ґрунтується на архітектурі нейронних мереж, що дає їй можливість обробляти великі обсяги тексту. Модель навчається на різноманітних даних з інтернету (форуми, блоги, новини тощо), що забезпечує її розуміння контексту та здатність до обговорення різних тем.

2. ***Розуміння контексту.*** На відміну від традиційних чатботів, які відповідають на запити за наперед визначеними сценаріями, *LaMDA* враховує попередні повідомлення у діалозі, забезпечуючи послідовність і релевантність відповідей.

3. ***Генерування природних відповідей.*** *LaMDA* створює відповіді, які звучать природно й імітують людське спілкування. Це досягається завдяки складним моделям мовлення, що формують зв'язні та осмислені речення.

4. ***Широкий спектр тем.*** Модель може підтримувати розмови на різні теми, відповідати на запитання, давати поради, вести світські бесіди і навіть брати участь у творчих дискусіях.

5. ***Постійне вдосконалення.*** *LaMDA* вдосконалюється завдяки навчанню на нових даних і зворотному зв'язку з користувачами. Це дає змогу моделі адаптуватися до змін у мові та культурних тенденцій.

6. ***Етичні міркування.*** *Google* приділяє значну увагу етичним аспектам, працюючи над запобіганням генеруванню небажаного або шкідливого контенту, а також забезпечуючи конфіденційність даних користувачів.

Пошук інформації в системах оброблення природної мови

Системи оброблення природної мови дають можливість знаходити документи, найбільш релевантні до запиту. Це критична проблема, яка стоїть перед кожною пошуково-рекомендаційною системою. Мета пошуку інформації полягає не лише в наданні відповіді на конкретний запит, а й у виборі релевантних документів із величезної колекції, що може містити мільйони одиниць.

Основні завдання систем пошуку:

1) ***індексація.*** У більшості сучасних систем індексація здійснюється за допомогою моделі векторного простору, що дає можливість створювати ефективне подання тексту.

2) ***зіставлення*** виконується з використанням показників подібності або відстані, щоб знайти документи, які найбільше відповідають введеному запиту.

Новітні підходи. *Google* об'єднав функцію пошуку інформації з мультимодальною моделлю, яка працює з текстовими, графічними та відеоданими. На рис. 14.3 зображено приклад двобаштової мережі, яка створює представлення вхідного запиту та групи документів (або елементів) за допомогою двох

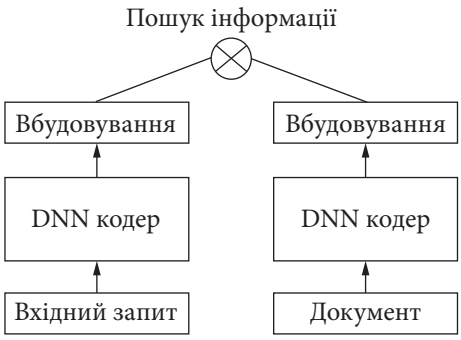


Рис. 14.3. Двобаштова мережа для подання вхідного запиту та групи документів (або елементів) за допомогою двох окремих мереж [14.1]

окремих мереж. Ця архітектура порівнює зображення запиту з поданням релевантних результатів пошуку [14.1].

Важливу роль у системах пошуку інформації, як і в багатьох інших задачах оброблення природної мови, відіграють *DNN-кодери (Deep Neural Network Encoder). DNN*-кодери є компонентами глибоких нейронних мереж, які використовують для перетворення вхідних текстових даних на числові зображення (вектори), зберігаючи інформацію про контекст і значення. Завдяки *DNN*-кодерам системи оброблення природної мови можуть краще розуміти й генерувати людське мовлення, що забезпечує більш природну та ефективну взаємодію між людиною і машиною.

Особливості роботи DNN-кодера

1. ***Перетворення тексту на вектори.*** *DNN*-кодер використовує нейронні мережі для перетворення слів або речень на багатовимірні вектори. Ці вектори містять семантичну інформацію, що дає змогу моделі краще розуміти значення слів у контексті.

2. ***Урахування контексту.*** Кодер обробляє вхідні дані з урахуванням контексту, даючи можливість розрізняти значення одного й того самого слова залежно від його місця в реченні.

3. ***Навчання на великих даних.*** *DNN*-кодери навчаються на великих обсягах текстових даних, що допомагає їм виявляти складні патерни у мовленні й створювати точні векторні зображення.

4. ***Використання в різних завданнях.*** Кодери є ключовими компонентами у завданнях пошуку інформації, машинного перекладу, розпізнавання мови, текстової класифікації, генерації тексту тощо.

5. ***Архітектура.*** *DNN*-кодери зазвичай використовують рекурентні нейронні мережі *(RNN),* довготривалу короткочасну пам'ять *(LSTM)* або трансформери (як-от *BERT* чи *GPT*). Ці архітектури забезпечують ефективне оброблення послідовностей даних і збереження контекстної інформації.

Резюмування тексту

Резюмування — це завдання скорочення тексту з виділенням найбільш релевантної інформації. Компанія *Salesforce* розробила підсумовувач [14.8], який являє собою інструмент або модель для автоматичного створення ре-

зюме текстів. Цей підсумовувач забезпечує ефективне і точне резюмування завдяки передовим технологіям оброблення природної мови.

Функції підсумовувача:

1) ***скорочення тексту.*** Інструмент аналізує великі тексти й скорочує їх, виділяючи основну інформацію.

2) ***фактична узгодженість.*** Підсумовувач оцінює точність і відповідність фактів у створеному резюме, щоб уникнути помилок і спотворень інформації.

Методи резюмування

1. ***Екстрактивне узагальнення.*** Цей підхід полягає у вилученні найбільш важливих речень із тексту та їх об'єднанні для формування резюме. Зазвичай екстрактивне узагальнення оцінює кожне речення у вхідному тексті, а потім обирає кілька речень, що найкраще передають суть документа.

2. ***Абстрактне узагальнення.*** У цьому методі резюме створюється за допомогою перефразування тексту. Це нагадує написання анотації, яка може містити слова або речення, яких немає в оригінальному тексті. Абстрактне узагальнення ґрунтується на моделюванні завдання «послідовність у послідовність», де вхідними даними є розлогий текст, а вихідними — стисле резюме.

Відповіді на запитання

Моделі відповіді на запитання (*Question Answering, QA)* здатні розуміти запити користувачів і знаходити відповіді в заданому тексті або базі знань. Їх використовують у різних додатках, таких як пошукові системи, чатботи, віртуальні асистенти та системи підтримки клієнтів [14.9].

Основні етапи роботи моделі *QA*

1. ***Аналіз запитання.*** Модель *QA* спочатку аналізує запитання користувача, щоб зрозуміти його структуру та семантику. Це полягає у:

- розпізнаванні ключових слів і фраз;
- визначенні типу запитання (наприклад, хто? що? коли? де? чому? як?);
- ідентифікації основної сутності запитання.

2. ***Пошук інформації.*** Після аналізу модель виконує пошук відповідної інформації в тексті або базі даних. До нього входить:

- пошук документів, які потенційно містять відповідь;
- пошук абзаців або речень, що, ймовірно, містять відповідь на запитання.

3. ***Екстрагування відповіді.*** Модель витягує відповідь зі знайденого тексту за допомогою одного з підходів:

- **екстрактивний підхід** — витягування фрагментів тексту без змін;
- **абстрактивний підхід** — перефразування або генерування відповіді на основі знайденої інформації.

4. ***Оцінка відповідності.*** Модель оцінює релевантність знайденої відповіді, включно з перевіркою:

• фактів;

• узгодженості;

• відповідності суті запитання.

5. ***Видача відповіді.*** Після оброблення та оцінки модель надає користувачеві відповідь.

Типи завдань щодо відповідей на запитання:

1) ***множинний вибір.*** Завдання містить запитання та кілька варіантів відповідей. Модель має обрати найбільш точну відповідь із наданих варіантів;

2) ***відкритий домен.*** У цьому типі модель надає відповідь природною мовою без будь-яких попередньо визначених варіантів. Відповіді можуть ґрунтуватися на аналізі великої кількості текстів.

Навчання моделей *QA*. Моделі *QA* навчаються на великих наборах даних із прикладами запитань і відповідей. Вони використовують різні архітектури глибинного навчання, зокрема: трансформери (наприклад, *BERT, GPT*), що дає моделі можливість розуміти контекст і генерувати точні відповіді.

Підсумовуючи розгляд практичних застосувань *NLP*, наведемо їх стислий порівняльний аналіз з оцінкою популярності (масштабів поширення) за шкалою Міллера (8 — найбільш поширені застосування; 1 — найменш поширені) та переваг і обмежень у використанні цих застосувань (табл. 14.1).

Аналіз 13-ти основних застосувань *NLP* (див. табл. 14.1) ілюструє їх високу актуальність, широке впровадження в практику та популярність, яка розподіляється за трьома рівнями рейтингу застосувань за шкалою Міллера: 8 — високий, 6—7 — середній та 5 — низький.

До категорії з високим рівнем рейтингу належать найбільш затребувані застосування, які широко використовуються в комерційних і суспільних сферах, зокрема:

• *аналіз настроїв* — забезпечує моніторинг громадської думки та аналіз емоцій у соціальних медіа й маркетингу;

• *машинний переклад* — ключове застосування для глобалізації бізнесу та міжнародної співпраці;

• *чатботи* — використовуються для автоматизації діалогів і покращення клієнтського досвіду;

• *відповіді на запитання* (QA) — сприяють поліпшенню користувацького досвіду через інтеграцію у віртуальних асистентах.

Ці застосування мають високу ефективність і універсальність у різних сферах. Разом з тим, їм притаманні складнощі з якістю даних і розумінням контексту.

Група з середнім рівнем застосувань, які мають значну цінність, але менш поширені через специфічні обмеження, містить такі застосування:

• *класифікація токсичності тексту* — ефективний інструмент для гарантування безпеки користувачів у соціальних мережах;

• *виявлення спаму* — запобігає поширенню небажаних повідомлень у електронній пошті та онлайн-комунікаціях;

• *моделі виправлення граматичних помилок* — поліпшують якість текстів у сфері освіти та текстових редакторів;

• *генерування тексту* — важливий інструмент для автоматизації створення контенту;

• *автозаповнення* — спрощує і прискорює введення тексту в мобільних додатках і редакторах;

• *пошук інформації* — ефективне рішення для роботи з великими базами даних у пошукових системах.

Ця група застосувань характеризується значною практичною зручністю та забезпеченням безпеки, але має певні обмеження в обробленні складних сценаріїв і контекстів.

До групи з низьким рівнем популярності належать вузькоспеціалізовані застосування, які характеризуються високою складністю впровадження. До цієї групи належать:

• *тематичне моделювання* — використовується для виявлення тем у великих корпусах текстів;

• *резюмування тексту* — скорочує обсяг тексту, але може втратити важливу інформацію.

Ці застосування корисні для спеціалізованих завдань, таких як аналітика текстів і документообіг, але вони мають низький рівень популярності через складність інтерпретації результатів та обмежену універсальність.

Отже, застосування *NLP* з високим рівнем популярності є критично важливими для сучасного штучного інтелекту, вони домінують у практичному використанні завдяки їхній універсальності та затребуваності. Застосування середнього рівня популярності є цінними інструментами для підвищення зручності, безпеки й ефективності текстової обробки. Застосування низького рівня популярності мають перспективи зростання за умов удосконалення алгоритмів і розширення практичних сценаріїв використання.

Разом з тим спостерігається стрімкий загальний розвиток *NLP*, спрямований на вдосконалення моделей з огляду на їхні точність, контекстуальність та можливість інтеграції з іншими технологіями. Головними тенденціями цього розвитку є такі:

• прогресування розвитку великих мовних моделей *(Large Language Models, LLMs*). Розроблення складних *LLM*, як-от *GPT-4* від *OpenAI* і *Gemini* від *Google* [14.6], значно розширила можливості *NLP*. Ці моделі демонструють

високі результати в розумінні та генерації мови, наближеної до людської, що забезпечує більш тонку та контекстно залежну взаємодію;

• усе більше уваги приділяється моделям, які обробляють кілька типів даних, зокрема текст, зображення та аудіо. Такий мультимодальний підхід дає можливість краще розуміти й генерувати контент, розширюючи застосування *NLP* у різних сферах;

• прогрес у сфері аналізу емоцій і виявлення настроїв дає змогу точніше визначати емоції та настрої в текстах. Це особливо важливо для аналізу відгуків клієнтів та моніторингу соціальних медіа;

Перелік та порівняльний аналіз

Назва застосування	Сфера застосування
Аналіз настроїв	Соціальні медіа, маркетинг, аналітика емоцій
Класифікація токсичності	Модерація контенту, захист користувачів у соціальних мережах
Машинний переклад	Локалізація контенту, міжмовна комунікація
Розпізнавання іменованих сутностей	Інформаційний пошук, бізнес-аналітика, *OSINT*
Виявлення спаму	Електронна пошта, онлайн-комунікації
Моделі виправлення граматичних помилок	Освіта, текстові редактори
Тематичне моделювання	Аналітика текстів, дослідження
Генерування тексту	Створення креативних текстів, контенту
Автозаповнення	Мобільні додатки, текстові редактори
Чатботи	Сервісна підтримка, автоматизація діалогів
Пошук інформації	Пошукові системи, архіви
Резюмування тексту	Публікації, документообіг
Відповіді на запитання (*QA*)	Чатботи, віртуальні асистенти

• спостерігається активна інтеграція *NLP* у бізнес-додатки. Компанії все частіше використовують *NLP* для покращення обслуговування клієнтів за допомогою чатботів, автоматизації створення контенту та вдосконалення процесів аналізу даних. Така інтеграція оптимізує робочі процеси і поліпшує досвід користувачів;

• зі зростанням складності моделей *NLP* все більшої ваги набуває потреба зробити ці моделі зрозумілими. Розуміння того, як моделі досягають конкретних результатів, є важливим для забезпечення довіри та етичного використання ШІ;

Таблиця 14.1

головних застосувань NLP

Переваги	Обмеження	Рейтинг за шкалою Міллера (Версія *ChatGPT*)
Висока популярність; придатний для моніторингу громадської думки, досліджень ринку	Значна залежність від якості навчальних даних	8
Ефективний інструмент для забезпечення безпечного онлайн-середовища	Складнощі коректного розпізнавання контексту та унеможливлення хибних спрацьовувань	7
Критично важливий для глобалізації бізнесу та міжнародної співпраці	Складнощі перекладу специфічних, культурно або контекстуально забарвлених фраз	8
Спрощує структурування великих текстових даних	Суттєва залежність від налаштування категорій	6
Ефективно запобігає спаму та небажаним повідомленням	Можливі помилкові спрацьовування	7
Покращує якість текстів; корисний для освітніх цілей	Можливі помилкові спрацьовування	6
Виявлення тем у великих корпусах текстів	Складнощі в інтерпретації результатів	5
Важливий інструмент для автоматизації	Ризик генерації неякісного контенту	7
Покращує зручність і швидкість набору тексту	Можливі неточності у складних сценаріях	6
Поліпшує взаємодію з клієнтами	Обмежена глибина розуміння контексту	8
Ефективність під час роботи з великими базами даних	Залежність від якісної індексації	7
Скорочує час читання	Ризик втрати важливої інформації	5
Поліпшує користувацький досвід	Складнощі роботи у відкритому домені	8

• усунення упереджень та врахування етичних аспектів у моделях *NLP* залишається критично важливою проблемою. Тому постійно триває розроблення методів виявлення і зменшення упереджень з метою забезпечення справедливого та рівноправного застосування ШІ.

Ці тенденції підкреслюють динамічний і швидкий розвиток *NLP,* нові досягнення в цьому напрямі розширюють можливості та застосування ШІ в різних сферах людської діяльності.

14.3. Як працюють моделі обробки природної мови?

Моделі *NLP* працюють за допомогою встановлення зв'язків між складовими частинами мови, такими як літери, слова та речення, що містяться у текстових наборах даних. Архітектури *NLP* ґрунтуються на різних процесах оброблення даних, вилучення ознак і моделювання (див. рис. 14.1). Розглянемо основні етапи цих процесів.

Попереднє оброблення даних

Перш ніж модель зможе виконувати завдання оброблення тексту, його зазвичай потрібно попередньо обробити. Це дає змогу підвищити продуктивність моделі або перетворити слова й символи на формат, зрозумілий для моделі. Для вирішення таких завдань застосовується підхід, відомий як штучний інтелект, орієнтований на дані. Цей підхід передбачає пріоритетну увагу до збирання, оброблення й аналізу великих обсягів даних, що є критично важливим для вдосконалення моделей машинного навчання.

Основні етапи такого підходу:

• *збирання даних* — формування великого набору різноманітних даних за визначеною тематикою із численних джерел;

• *попереднє оброблення* — очищення даних, усунення шуму, заповнення пропусків і нормалізація;

• *аналіз даних* — використання статистичних методів і алгоритмів машинного навчання для виявлення шаблонів та взаємозв'язків;

• *моделювання* — створення і навчання моделей для вирішення завдань прогнозування чи класифікації;

• *оцінка і оптимізація* — перевірка точності моделей, їх оптимізація та застосування до нових даних.

Зведення слів до їх базових, або кореневих, форм (стемінг)

Стемінг є ключовим етапом попереднього оброблення даних у завданнях *NLP.* Він передбачає зведення слів до базових, або кореневих, форм, що зменшує кількість унікальних слів у тексті і спрощує аналіз даних.

Основні аспекти стемінгу:

• скорочення розмірності — зменшення кількості унікальних слів за допомогою об'єднання різних форм одного слова в базову форму;

• покращення узагальнення — модель краще узагальнює інформацію, оскільки різні форми слова розглядаються як єдине поняття (наприклад, «*running*», «*runs*» і «*ran*» зводяться до «*run*»);

• евристичний підхід — використання простих правил для відсікання закінчень і префіксів;

• інструменти — як інструменти можна розглядати бібліотеки *NLTK (Natural Language Toolkit)* або *Snowball Stemmer* та інші, що підтримують стемінг для різних мов. Зокрема, *Snowball Stemmer* — це бібліотека для стемінгу в обробленні природної мови, яка є частиною бібліотеки *NLTK*. Вона ґрунтується на алгоритмі *Snowball* (https://snowballstem.org/). Однак стемінг може бути менш точним порівняно з лематизацією, яка враховує морфологічну структуру та контекст.

Лематизація

Лематизація є ще одним важливим етапом попереднього оброблення тексту. Вона полягає у перетворенні слів на їх основну форму (лему), враховуючи контекст і частину мови.

Основні кроки лематизації:

• *токенізація* — розділення тексту на окремі слова, або токени;

• *POS-аналіз* — визначення частини мови для кожного токена;

• *лематизація* — перетворення токенів на їх леми залежно від контексту. Наприклад, слова «бігав», «біжу» і «бігти» зводяться до леми «бігти»; слова «машина», «машини», «машиною» — до «машина».

Переваги лематизації:

• зменшення розмірності даних;

• зменшення кількості унікальних слів, що підвищує ефективність алгоритмів *NLP*;

• більша точність завдяки врахуванню контексту й частини мови.

Для лематизації популярною є мова програмування «*Python*» з бібліотеками:

• *NLTK (Natural Language Toolkit*) — високофункціональний інструмент для *NLP*;

• *Pymorphy2* — морфологічний аналізатор для російської та української мов;

• *spaCy* — потужна бібліотека для швидкого й точного оброблення текстів.

Наведемо приклади використання бібліотек *NLTK, Pymorphy2* та *spaCy* для лематизації українського тексту [14.10]:

Приклад 1. **Використання бібліотек *NLTK* та *Pymorphy2***

Спочатку встановимо необхідні бібліотеки:

```
bash
Копіювати код
pip install nltk pymorphy2
```

Після встановлення використаємо бібліотеку Pymorphy2 для лематизації українського тексту:

```
python
Копіювати код
import nltk
import pymorphy2

# Ініціалізація аналізатора pymorphy2
morph = pymorphy2.MorphAnalyzer(lang='uk')

# Функція для лематизації тексту
def lemmatize_text(text):
    words = nltk.word_tokenize(text) # Токенізація тексту
    lemmas = [morph.parse(word)[0].normal_form for word in words] # Отримання лем для кожного слова
    return ' '.join(lemmas)

# Приклад тексту українською мовою
text = «Це приклад використання лематизації для українського тексту.»

# Лематизація тексту
lemmatized_text = lemmatize_text(text)
print(lemmatized_text)
```

Висновок. З використанням бібліотеки *Pymorphy2*, яка підтримує українську мову, та функціональності токенізації від *NLTK* виконано лематизацію тексту. У прикладі текст спочатку токенізується, щоб розділити його на окремі слова, після чого кожне слово аналізується й приводиться до базової форми. Отриманий результат показує, як легко можна інтегрувати ці бібліотеки для попереднього оброблення тексту. Це відкриває можливості для подальших досліджень і застосувань, таких як створення пошукових систем, аналіз настроїв чи систем машинного перекладу для української мови. Отже, комбінація *NLTK* і *Pymorphy2* є ефективним вирішенням для завдань оброблення тексту, орієнтованих на українську мову.

Приклад 2. **Використання бібліотеки «spaCy»**

Для використання spaCy із підтримкою української мови спочатку встановимо відповідну модель:

```
bash
Копіювати код
pip install spacy
```

Завантажимо модель для української мови:

```
python
Копіювати код
import spacy

# Завантаження моделі для української мови (перевірте наявність моделі uk_core_news_sm)
nlp = spacy.load('uk_core_news_sm')

# Приклад тексту
text = «Він бігав дуже швидко.»

# Обробка тексту
doc = nlp(text)

# Виведення лем для кожного слова в тексті
for token in doc:
    print(f'{token.text} -> {token.lemma_}')
```

Отримуємо результат:

```
Rust
Копіювати код
Він -> він
бігав -> бігти
дуже -> дуже
швидко -> швидко
```

Висновок. Лематизація за допомогою *spaCy* виконується швидко і зручно завдяки інтеграції моделей, які підтримують українську мову. У наведеному прикладі кожне слово тексту аналізується, і *spaCy* автоматично надає його лему — базову форму слова. Результати демонструють високу точність оброблення тексту, де слова збережено у своїй граматичній формі, але зведено до їхніх базових значень, що є критично важливим для багатьох *NLP*-завдань. Приклад свідчить, що використання *spaCy* для лематизації українського тексту має низку переваг, зокрема швидкість, легкість інтеграції та доступність готових моделей. Це робить *spaCy* ефективним інструментом

для розробників, які працюють над завданнями аналізу текстів, пошуку інформації або створення чатботів для української мови.

Сегментація речень

Сегментація речень *(sentence segmentation)* в обробленні природної мови (*NLP*) є процесом розділення тексту на окремі речення для структурування тексту з метою подальшого оброблення та аналізу. Це базовий і важливий етап у багатьох завданнях *NLP*, оскільки розгляд тексту на рівні речень значною мірою сприяє аналізу й обробленню. Для сегментації речень використовують такі основні підходи:

1) ***на основі правил.*** Використання регулярних виразів та шаблонів для виявлення кінцевих знаків пунктуації (крапки, знаки запитання та оклику). Враховується контекст, щоб уникнути неправильного розділення, наприклад, коли крапка використовується у скороченнях або числах;

2) ***на основі машинного навчання***. Використання класифікаторів, які навчені розпізнавати кінець речення на основі контексту та ознак. Для цього застосовуються моделі послідовностей, як-от *CRF (Conditional Random Fields),* для маркування та сегментації послідовностей [14.11]. Також використовуються нейронні мережі для виявлення закінчень речень.

Розглянемо приклади використання бібліотек *spaCy* та *NLTK* для сегментації речень [14.12].

Приклад 1. **Використання *spaCy* для сегментації речень.** Бібліотека *spaCy* має вбудовану підтримку сегментації речень. Сегментація здійснюється в такий спосіб:

```
python
Копіювати код
import spacy

# Завантаження англійської моделі
nlp = spacy.load(«en_core_web_sm»)

# Приклад тексту
text = «This is the first sentence. This is the second sentence! Is this the third sentence? Yes, it is.»

# Обробка тексту
doc = nlp(text)

# Виведення сегментованих речень
for sent in doc.sents:
    print(sent.text)
```

Результат:

kotlin
Копіювати код
This is the first sentence.
This is the second sentence!
Is this the third sentence?
Yes, it is.

Висновок. Завдяки вбудованим можливостям spaCy процес сегментації здійснюється автоматично, під час оброблення тексту за допомогою мовної моделі. Це забезпечує високу точність і правильне розділення тексту на речення навіть за наявності складної пунктуації. Перевага *spa* методу із застосуванням *Cy* полягає у його продуктивності та простоті використання. Завдяки оптимізації та підтримці великої кількості мов ця бібліотека ідеально придатна для завднь, які потребують швидкого й точного оброблення тексту. Застосування *spaCy* до сегментації речень може бути корисним у завданнях, пов'язаних із машинним перекладом, аналізом тексту чи підготовкою даних для подальшого використання.

Приклад 2. **Використання *NLTK* для сегментації речень** [14.13]:

python
Копіювати код
import nltk
nltk.download('punkt')

from nltk.tokenize import sent_tokenize

# Приклад тексту
text = «This is the first sentence. This is the second sentence! Is this the third sentence? Yes, it is.»

# Сегментація речень
sentences = sent_tokenize(text)

# Виведення сегментованих речень
for sentence in sentences:
    print(sentence)

Результат:

kotlin
Копіювати код
This is the first sentence.
This is the second sentence!
Is this the third sentence?
Yes, it is.

Висновок. Приклад демонструє високу якість розділення тексту, яку забезпечує *NLTK*, завдяки попередньо навченим алгоритмам, що враховують особливості пунктуації та структури мови. Перевага *NLTK* полягає у широкій доступності та можливості працювати з текстами багатьма мовами, що робить його універсальним інструментом для початкового оброблення тексту. *NLTK* є легким у використанні й ідеально придатний для освітніх цілей та прототипування *NLP*-систем. Його гнучкість і здатність працювати з різними мовами дають змогу використовувати *NLTK* у проєктах з обмеженими вимогами до швидкості, але з високими вимогами до точності аналізу.

Загальний висновок. Обидва інструменти — *spaCy* і *NLTK* — забезпечують ефективну сегментацію речень, проте їх вибір залежить від вимог проєкту. SpaCy придатний для швидких і високопродуктивних систем із сучасними мовними моделями, тоді як *NLTK* є універсальним вибором для простих і навчальних завдань.

Видалення стоп-слів

Видалення стоп-слів є важливим етапом попереднього оброблення тексту в завданнях оброблення природної мови. ***Стоп-слова*** — це загальновживані слова, які часто не мають значущого змісту для аналізу тексту, наприклад «і», «або», «на», «в», «the», «a», «an» тощо. Видалення таких слів допомагає:

- зменшити обсяг даних;
- знизити рівень шуму в текстових даних;
- підвищити ефективність і точність моделей оброблення природної мови.

Для видалення стоп-слів зазвичай використовують такі бібліотеки, як *NLTK, spaCy, sklearn, razdel* [14.14]. Розглянемо приклад використання *sklearn* для видалення стоп-слів [14.14].

Приклад:

```
python
Копіювати код
from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS

# Приклад тексту
text = «This is a sample sentence, showing off the stop words filtration.»

# Токенізація тексту
words = text.split()

# Видалення стоп-слів
filtered_words = [word for word in words if word.lower() not in ENGLISH_
STOP_WORDS]

print(filtered_words)
```

Результат:

```
css
Копіювати код
['sample', 'sentence,, 'showing', 'stop', 'words', 'filtration.']
```

Висновок. У прикладі застосовується вбудований список англійських стоп-слів ENGLISH_STOP_WORDS. Текст токенізується розділенням на окремі слова, після чого слова зі списку стоп-слів видаляються. Як результат, отримується очищений текст, у якому зберігаються лише змістовно важливі слова. Цей підхід є простим і ефективним для попереднього оброблення тексту, наприклад, у завданнях класифікації тексту, побудови векторних зображень або аналізу настроїв. Використання вбудованих інструментів *sklearn* дає змогу зекономити час і забезпечує стандартизований підхід до видалення стоп-слів. Це робить *sklearn* корисним інструментом для завдань початкового оброблення текстових даних у багатьох NLP-проєктах.

Токенізація

Токенізація — це процес розділення тексту на окремі слова чи фрагменти. У результаті текст набуває вигляду покажчика слів і токенізованого тексту, де кожне слово або фрагмент можуть бути перетворені на числові жетони для подальшого використання в різних методах глибокого навчання (рис. 14.4) [14.1]. Токенізація є важливим етапом в обробленні природної мови, з огляду на підготовку тексту для подальших етапів аналізу, таких як частиномовний аналіз, розпізнавання іменованих сутностей, синтаксичний аналіз тощо. В обробленні природної мови токенізація є першочерговим кроком, оскільки більшість алгоритмів і моделей машинного навчання потребують текстових даних у вигляді окремих токенів.

Основні методи токенізації

1. ***Токенізація за словами:***
- розділення тексту на окремі слова;
- врахування пунктуації та спеціальних символів;
- застосування для загальних завдань *NLP.*

2. ***Токенізація за реченнями:***
- розділення тексту на окремі речення;
- використання кінцевих знаків пунктуації;
- використання як підготовчий етап перед іншими завданнями *NLP.*

3. ***Символьна токенізація:***
- розділення тексту на окремі символи.

Цей тип токенізації використовується рідше, але він є ефективним для специфічних завдань, таких як аналіз геномів чи оброблення паролів [14.15].

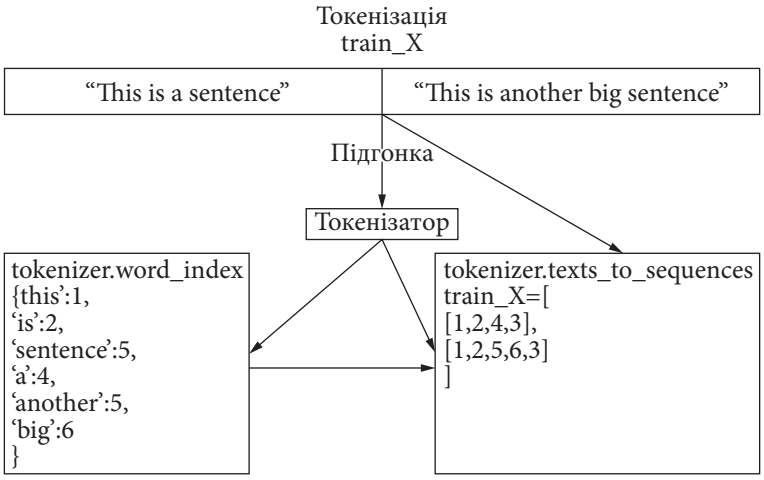


Рис. 14.4. Зображення кожного слова послідовністю чисел (токенізація) для будь-якого документа [14.1]

Для виконання токенізації в мові *Python* найчастіше використовують бібліотеки *NLTK, spaCy, razdel*. Наведемо приклад використання бібліотеки *razdel* для токенізації та сегментації україномовного тексту [14.16].

Приклад:

```
python
Копіювати код
from razdel import tokenize, sentenize

# Приклад тексту українською мовою
text = «Це приклад речення. Ось ще одне.»

# Токенізація за словами
words = [token.text for token in tokenize(text)]
print(words)

# Токенізація за реченнями
sentences = [sentence.text for sentence in sentenize(text)]
print(sentences)
```

Результат:

```
python
Копіювати код
[‘Це’, ‘приклад’, ‘речення’, ‘.’, ‘Ось’, ‘ще’, ‘одне’, ‘.’]
[‘Це приклад речення.’, ‘Ось ще одне.’]
```

Висновок. У наведеному прикладі текст спочатку токенізується на окремі слова, включно зі знаками пунктуації, що важливо для збереження контексту.

Далі застосовується сегментація на речення, що дає змогу отримати окремі завершені фрагменти тексту. Результат свідчить про здатність *razdel* коректно обробляти текст українською мовою, виділяючи як слова, так і речення. Такий підхід є важливим етапом попереднього оброблення тексту в *NLP*-завданнях, зокрема в аналізі текстів, класифікації, машинному перекладі або генеруванні тексту. Приклад підтверджує, що бібліотека *razdel* вирізняється швидкістю і точністю, що робить її корисним інструментом для роботи з україномовними текстами в рамках проєктів, орієнтованих на оброблення природної мови.

Виявлення ознак *(Feature extraction)* в обробленні природної мови — це процес перетворення текстових даних на числові вирази, які можуть бути використані алгоритмами машинного навчання. Це ключовий етап у побудові *NLP*-моделей, оскільки текстові дані у їх початковій формі не можуть бути безпосередньо оброблені більшістю алгоритмів.

Основні методи виявлення ознак

1. ***Метод «Мішок слів» (Bag of Words, BoW)*** [14.1]:

• текст подається як набір (або «мішок») слів без урахування порядку їх появи;

• кожному слову надається унікальний індекс, а текст зображається у вигляді вектора частот слів (рис. 14.5).

2. ***Метод «TF-IDF» (Term Frequency-Inverse Document Frequency)*** [14.17]:

• визначає міру важливості слова в документі відносно всіх документів у колекції;

• знижує вагу часто вживаних слів і підвищує вагу рідкісних слів.

3. ***Метод «Word Embeddings»:***

• зображує слова у вигляді векторів у багатовимірному просторі;

• здійснює навчання таких моделей, як: *Word2Vec* (*Word to Vector*) [14.18], *GloVe* (*Global Vectors for Word Representation*) [14.19], *FastText* (*Fast Text Processing*) [14.20]. Ці моделі створюють векторні зображення, що враховують семантичну подібність між словами. На відміну від традиційних методів, таких як *Bag of Words* або *TF-IDF*, що не враховують контекст, *Word Embeddings* дає можливість алгоритмам розуміти семантичні та синтаксичні зв'язки між словами.

4. ***Метод «Document Embeddings»*** [14.21]:

• зображує цілі документи у вигляді векторів;

• використовується в моделях *Doc2Vec* (*Document to Vector*) та *BERT*. Цей підхід враховує контекст тексту на більш високому рівні, що дає можливість глибшого розуміння змісту документів.

Наведемо приклад використання методу *Bag of Words* (*BoW*) за допомогою інструмента *CountVectorizer (Count Vectorizer)*, який є одним з основних елементів бібліотеки *Scikit-learn (Science Kit Learn)* [14.22]). Він використо-

Рис. 14.5. «Мішок слів» (*Bag of Words, BoW*) кодує загальну кількість використання кожного слова у відповідному корпусі [14.1]

вується для перетворення колекції текстових документів на матрицю чисел, де кожне число відповідає кількості разів появи певного слова у документі.

***Приклад*:**

```
from sklearn.feature_extraction.text import CountVectorizer

# Приклад текстових даних
texts = [«I love programming.», «Python is great!», «I hate bugs.»]
# Ініціалізація та налаштування CountVectorizer
vectorizer = CountVectorizer()

# Перетворення тексту на матрицю частот термінів
X = vectorizer.fit_transform(texts)

# Виведення результату
print(vectorizer.get_feature_names_out())
print(X.toarray())
```

Висновок. У наведеному прикладі *Count Vectorizer* виконує кілька важливих етапів. Спочатку відбувається ***токенізація*** тексту, тобто розділення текстових даних на окремі слова. Потім створюється ***словник***, який містить усі унікальні слова з текстів. На завершальному етапі будується ***матриця частот***, де для кожного документа визначається, скільки разів кожне слово зі словника трапляється в тексті. Результатом є структуроване числове зображення, яке можна легко використовувати для алгоритмів машинного навчання.

Оцінки важливості терміна в документі відносно корпусу документів

Ця операція здійснюється з застосуванням методу *TF-IDF (Term Frequency-Inverse Document Frequency)*. Як зазначалося в параграфі 14.2, *TF-IDF* — це статистичний метод, який використовують в обробленні природної мови та інформаційних системах для оцінювання важливості терміна в документі відносно корпусу документів [14.17]. Цей метод допомагає зрозуміти, які терміни є найбільш значущими в конкретному документі в контексті всього набору документів. На відміну від методу *Bag-of-Words*, який підраховує частоту появи кожного слова або *N*-грами в документі, метод *TF-IDF* дає можливість оцінити кожне слово за його важливістю. Щоб оцінити значення слова**, враховуються дві складові** (рис. 14.6) [14.1]:

1) ***TF (Term Frequency)*** — частота появи терміна *t* в документі *d*, яка зазвичай обчислюється як відношення кількості разів появи терміна *t* в документі *d* до загальної кількості термінів у документі *d*;

2) ***IDF (Inverse Document Frequency)*** — зворотна частота документів, яка вказує на те, наскільки рідкісним є термін *t* у всьому корпусі документів *D*. Якщо термін трапляється в багатьох документах, його значення буде меншим. *IDF* вимірюється як логарифм відношення кількості документів у корпусі *D* до кількості документів, що містять термін *t*. Для обчислення цього показника використовують загальну формулу: $TF\text{-}IDF(t, d, D) = TF(t, d) \times IDF(t, D)$.

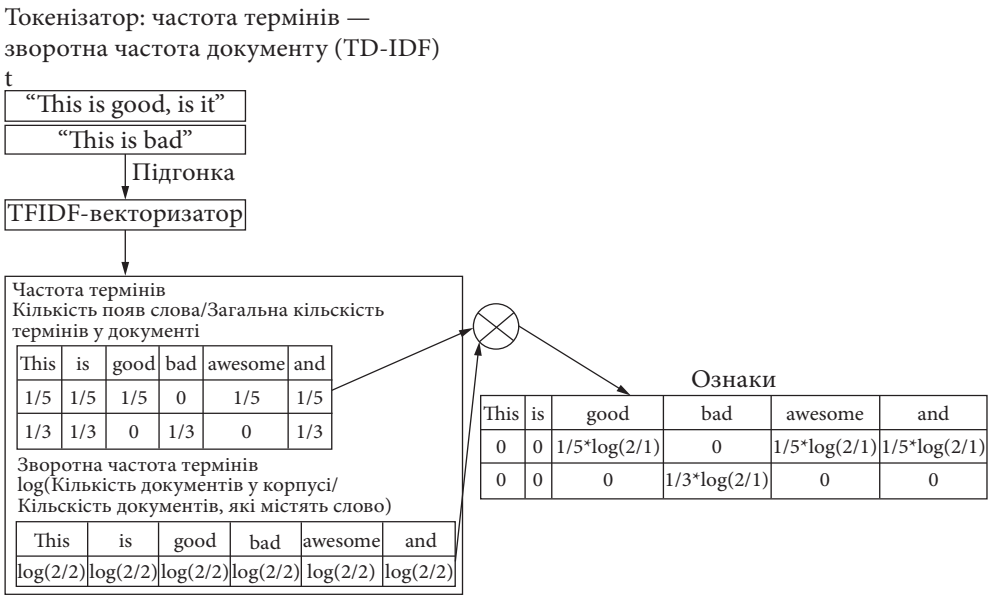


Рис. 14.6. Створення ознак документа на основі порівняння частоти появи кожного слова в документі і в усьому корпусі документів [14.1]

Приклад. Розглянемо приклад застосування методу *TF-IDF.* Нехай потрібно знайти ключові слова в кожному наборі документів. Алгоритм виконання цього завдання такий:

1) для кожного терміна в документі обчислюють його *TF*;

2) обчислюють *IDF* для кожного терміна по всьому корпусу документів;

3) отримані значення *TF* і *IDF* перемножують, щоб отримати *TF-IDF* для кожного терміна.

Наприклад, корпус складається з трьох документів:

Документ 1. «Оброблення тексту в обробці природної мови».

Документ 2. «Мова і текст: важливі концепти в обробці природної мови».

Документ 3. «Як обробляти природну мову для покращення тексту?».

Термін «обробка» має високе значення *TF* у Документі 1, але середнє значення *IDF*, оскільки він трапляється в кількох документах. Отже, його *TF-IDF*-значення може бути меншим, ніж у терміна «покращення», який може мати низьке значення *TF* у документі, але високе значення *IDF*, якщо він трапляється рідше в корпусі.

Перетворення слова на багатовимірні вектори

Таке перетворення здійснюється із застосуванням методу *Word2Vec* для створення векторних зображень слів. Цей метод розроблено командою *Google* під керівництвом Томаша Міколова в 2013 р. [14.23]. Метод дає змогу перетворювати слова на багатовимірні вектори, які розміщуються в просторі так, щоб близькі за змістом слова мали б подібні векторні зображення. Метод *Word2Vec* є потужним інструментом для розуміння та аналізу текстових даних і широко використовується в багатьох додатках оброблення природної мови, таких як класифікація тексту, кластеризація, пошук інформації та багато інших.

Для оптимізації векторних зображень для великих обсягів тексту є *два основних підходи на основі використання нейронних мереж.*

1. ***Skip-gram*** *(Skip* — пропускати, gram — *N*-грам) передбачає контекстні слова, використовуючи поточне слово. Це означає, що у разі застосування цього методу пропускаються деякі слова (згідно з певним «вікном» контексту) і вибирається поточне слово для передбачення контекстних слів, тобто слів, які перебувають у певному радіусі навколо поточного слова. Основна ідея полягає в тому, що поточне (центральне) слово використовується для передбачення інших слів (контексту), які трапляються поруч із ним у тексті. Цей метод дає можливість максимізувати ймовірність передбачення контекстних слів на основі певного центрального слова.

2. ***CBOW (Continuous Bag of Words)*** — це підхід до навчання *Word2Vec*, за якого модель намагається передбачити центральне слово, беручи певні слова з його контексту [14.24]. Це означає, що для кожного слова в тек-

сті модель застосовує навколишні слова для передбачення самого слова. У моделі *CBOW* слова з контексту (тобто слова поблизу центрального слова) використовуються для передбачення центрального слова. Це протилежний до *Skip-gram* підхід, де центральне слово береться для передбачення контекстних слів.

Наведемо приклад застосування ***Word2Vec.*** Припустімо, є текстовий корпус, який містить речення «The quick brown fox jumps over the lazy dog». Розглянемо приклад застосування *Word2Vec* для створення векторних зображень кожного слова в цьому реченні на основі *Python* з використанням бібліотеки *Gensim* [14.25].

Приклад використання Word2Vec:

```
python
Копіювати код
from gensim.models import Word2Vec

# Приклад текстового корпусу
sentences = [
    [«the», «quick», «brown», «fox», «jumps», «over», «the», «lazy», «dog»],
    [«I», «love», «machine», «learning»],
    [«word2vec», «is», «a», «great», «tool»]
]

# Навчання моделі Word2Vec
model = Word2Vec(sentences, vector_size=100, window=5, min_count=1, workers=4)

# Отримання векторного представлення слова «fox»
vector = model.wv[‘fox’]
print(vector)
```

Висновок. Приклад ілюструє, що завдяки *Word2Vec* кожне слово в корпусі перетворюється на багатовимірний вектор, який відображає контекстуальне значення цього слова у реченнях. Результат, отриманий у вигляді вектора для слова «fox», демонструє здатність моделі навчатися семантичних і синтаксичних зв’язків між словами. Такі векторні зображення є основою багатьох сучасних *NLP*-завдань, таких як класифікація тексту, пошук семантичної близькості слів, кластеризація документів тощо.

Цей підхід ефективний для завдань, де потрібне збереження та використання семантичної інформації, а також може бути інтегрований у складніші моделі, такі як трансформери або рекурентні нейронні мережі. Отже, *Word2Vec* є потужним інструментом в арсеналі *NLP*-розробників, що значно спрощує і вдосконалює роботу з текстовими даними.

Висновки

Розділ 14 присвячений одному з найактуальніших напрямів розвитку штучного інтелекту — обробці природної мови (*NLP*), який відіграє ключову роль у забезпеченні взаємодії між людиною та машиною. Стрімкий прогрес у цій галузі обумовлений низкою технологічних проривів, що кардинально змінюють підходи до розуміння, генерації та адаптації мовних моделей. До основних рушійних сил цих змін належать поява великих трансформерних моделей, оптимізація алгоритмів навчання, зростання обчислювальних потужностей, впровадження інтерактивного навчання та розширення функціональності моделей. Завдяки цьому сучасні моделі *NLP* використовуються не лише для текстових запитів, а й для аналізу зображень, аудіо та відео, що відкриває нові можливості для мультифункціональних систем штучного інтелекту.

Сучасні мовні моделі стають енергоефективнішими, швидшими та більш адаптивними до потреб користувачів. Це досягається завдяки новим методам навчання, удосконаленню архітектури та стрімкому розвитку апаратного забезпечення, що дає змогу значно підвищити їхню продуктивність при скороченні витрат обчислювальних ресурсів. Галузь обробки природної мови перебуває на етапі безпрецедентного прогресу, що сприяє створенню високотехнологічних, ефективних і гнучких систем штучного інтелекту, здатних забезпечувати якісну взаємодію між людиною та цифровими платформами.

NLP розглядається як складна, адаптивна система, здатна ефективно реагувати на різноманітні завдання і контексти. Вона об'єднує численні процеси та алгоритми, що дають змогу аналізувати й обробляти природну мову з урахуванням складних зв'язків між словами та їхнім контекстом. Основними тенденціями розвитку *NLP* на сучасному етапі є зростання ролі великих мовних моделей, які демонструють високу здатність до контекстного розуміння тексту та генерування значущих відповідей. Поглиблення контекстного розуміння тексту забезпечується новими моделями, які враховують не лише окремі слова, а й складні взаємозв'язки між ними, що сприяє досягненню точнішого аналізу. Інтеграція *NLP* у повсякденні застосунки набуває популярності, зокрема у таких сферах як чатботи, пошукові системи, системи перекладу, аналіз тексту та розпізнавання мовлення.

Сучасні моделі *NLP* охоплюють кілька ключових етапів оброблення даних, зокрема попереднє оброблення тексту (очищення тексту, видалення шуму, застосування стемінгу та лематизації для приведення слів до їх базових форм), сегментацію і токенізацію (розділення тексту на речення та окремі значущі елементи — слова чи токени), видобування ознак (оцінку важливості термінів та виділення ключових слів), перетворення слів на багатовимірні вектори та створення числових зображень слів для подальшого аналізу та оброблення.

Отже, оброблення природної мови є фундаментальним напрямом сучасного штучного інтелекту. Воно закладає основу для створення інтелектуальних систем, здатних до взаємодії з людиною на новому рівні, забезпечуючи автоматизацію комунікацій, точність аналізу та прийняття рішень. Розвиток *NLP* визначає майбутнє технологій, орієнтованих на взаємодію людини і машини, і є невід'ємною складовою прогресу у сфері штучного інтелекту.

СПИСОК ЛІТЕРАТУРИ

14.1. Deeplearning.ai. Natural Language Processing [Online course]. 2023, January 11. URL: https://www.deeplearning.ai/resources/natural-language-processing/#What_Is_Natural_Language_Processing_(NLP)_Used_for

14.2. Khurana D., Koli A., Khatter K., & Singh S. Natural language processing: State of the art, current trends and challenges. *Multimedia Tools and Applications*. 2023. 82. P. 3713—3744. URL: https://link.springer.com/article/10.1007/s11042-022-13428-4

14.3. Goodfellow I., Bengio Y., & Courville A. Deep learning. MIT Press. 2016. URL: https://www.deeplearningbook.org/

14.4. Робінсон Н. Grammarly Review: є Grammarly Преміум того вартий? 2024. URL: https://www.guru99.com/uk/grammarly-review.html

14.5 Devlin J., Chang M.-W., Lee K., & Toutanova K. BERT: Pre-training of deep bidirectional transformers for language understanding (arXiv:1810.04805). arXiv. 2019. URL: https://arxiv.org/abs/1810.04805

14.6. Thoppilan R., De Freitas D., Hall J., et al. LaMDA: Language Models for Dialog Applications. Cornell University. 2022. URL: https://arxiv.org/abs/2201.08239

14.7. Books Notes. Design a search autocomplete system. 2022. URL: https://books.dwf.dev/docs/system-design/c14

14.8. Deeplearning.ai. Machine learning research: Keeping the facts straight. 2019, December 11. URL: https://www.deeplearning.ai/the-batch/keeping-the-facts-straight/

14.9. Carrigan M., & Noyan M. Question answering with Hugging Face Transformers. 2022, January 13. URL: https://keras.io/examples/nlp/question_answering/

14.10. Stack Overflow. How to make a JS code which allows user to view their page only if they satisfy the 'if' condition? 2020, January 21. URL: https://stackoverflow.com/questions/59845621/how-to-make-a-js-code-which-allows-user-to-view-their-page-only-if-they-satisfy

14.11. Scikit-learn Documentation. Scikit-learn documentation. 2024. URL: https://www.bing.com/search?q=Scikit-learn+Documentation&form=ANNTH1&refig=7c98efa02e6c403d9fa9a146bf8e0bec&pc=U531

14.12. spaCy. Industrial-strength natural language processing in Python. 2024. URL: https://spacy.io/?ref=floydhub.ghost.io

14.13. NLTK Documentation. Natural Language Toolkit. 2023. URL: https://www.nltk.org/#:~:text=Natural%20Language%20Toolkit

14.14. Sklearn-Crfsuite Documentation. Sklearn-crfsuite documentation. 2024. URL: https://sklearn-crfsuite.readthedocs.io/en/latest/

14.15. Neptune.ai. Tokenization in NLP: Types, challenges, examples, tools. 2024. URL: https://neptune.ai/blog/tokenization-in-nlp

14.16. Razdel Library. Razdel. 2025. URL: https://github.com/natasha/razdel

14.17. Manning C. D., Raghavan P., Schütze H. Introduction to information retrieval. Cambridge University Press. 2008. URL: https://nlp.stanford.edu/IR-book/

14.18. Mikolov T., Chen K., Corrado G., Dean J. Efficient estimation of word representations in vector space. Cornell University. 2013. URL: https://arxiv.org/abs/1301.3781

14.19. Pennington J., Socher R., Manning C. D. GloVe: Global vectors for word representation. 2014. URL: https://arxiv.org/abs/1406.2077

14.20. Bojanowski P., Grave E., Joulin A., Mikolov T. Enriching word vectors with subword information. Cornell University. 2016. URL: https://arxiv.org/abs/1607.04606

14.21. Le Q. V., Mikolov T. Distributed representations of sentences and documents. Cornell University. 2014. URL: https://arxiv.org/abs/1405.4053

14.22. Scikit-learn. Scikit-learn: Machine learning in Python. 2022. URL: https://github.com/scikit-learn/scikit-learn

14.23. Mikolov T., Sutskever I., Chen K., Corrado G., Dean J. Distributed representations of words and phrases and their compositionality. In: *Advances in Neural Information Processing Systems (NIPS '13)*, 26, 3111—3119. Curran Associates Inc. 2013. URL: https://dl.acm.org/doi/10.5555/2999792.2999959

РОЗДІЛ 15

МЕТОДИ І МОДЕЛІ ОБРОБКИ ПРИРОДНОЇ МОВИ

15.1. Традиційні методи і моделі машинного навчання

15.2. Методи і моделі глибокого навчання
в обробці природної мови

15.3. Методи глибокого навчання з підкріпленням

Висновки

Список літератури

Більшість задач *NLP*, розглянутих у розділі 14, можна вирішити за допомогою низки загальних методів і моделей. Корисно розглядати методи і моделі обробки природної мови у двох категоріях: *традиційні методи і моделі машинного навчання* та *методи і моделі глибокого навчання*.

Машинне навчання (*Machine Learning, ML*) і **глибоке навчання (*Deep Learning, DL*)** у контексті методів і моделей обробки природної мови мають чіткі визначення, адаптовані до цього напряму.

Машинне навчання (ML) — це розділ штучного інтелекту, що займається розробленням алгоритмів, здатних вчитися на основі даних і покращувати свою продуктивність без явного програмування всіх правил. У контексті *NLP* машинне навчання дає можливість розробляти моделі, які здатні аналізувати і генерувати природну мову, використовуючи дані для автоматичного покращення своєї продуктивності. Машинне навчання в *NLP* дає моделям змогу виконувати завдання без явного програмування правил, орієнтуючись на навчання на основі текстових корпусів.

Глибоке навчання (DL) — це підмножина машинного навчання, яка використовує штучні нейронні мережі з багатьма шарами для автоматичного виділення ознак із даних і побудови високоточних моделей, здатних працювати зі складними і нерегулярними структурами тексту.

Еволюція технологій оброблення природної мови розкриває суттєві відмінності між машинним навчанням і глибоким навчанням, кожне з яких має певні унікальні характеристики, переваги та обмеження [15.1—15.5]. Ці дві методології часто розглядають як різні підходи до вирішення завдань *NLP*, що мають значний вплив на якість і ефективність обробки текстових даних.

Машинне навчання зазвичай ґрунтується на ручному виділенні ключових ознак тексту, що потребує участі експертів. Ці ознаки створюються через правила або алгоритми, що дають можливість моделі виявляти патерни в даних. Глибоке навчання, натомість, пропонує автоматизований підхід: нейронні мережі самостійно виявляють релевантні ознаки з тексту, що особливо ефективно для роботи зі складними структурами даних. Ця автоматизація зменшує залежність від втручання людини і дає змогу глибоким моделям виявляти складні взаємозв'язки в тексті, які могли би залишитися поза увагою за ручного проєктування.

Моделі, які використовують у машинному навчанні, зазвичай є простішими і містять такі алгоритми як логістична регресія, метод опорних векторів (*SVM*), дерева рішень чи наївний Байєс. Глибоке навчання, натомість, використовує складніші архітектури багатошарових нейронних мереж, як-от рекурентні мережі (*RNN*), згорткові мережі (*CNN*) та трансформери (наприклад, *GPT, BERT*). Ці моделі сприяють досягненню високих результатів глибокого навчання завдяки здатності вивчати контекстуальні залежності та багаторівневі подання тексту.

Обсяг даних є ще одним важливим фактором, що розрізняє ці підходи. Машинне навчання може ефективно працювати з меншими наборами даних за умови якісного проєктування ознак. Глибоке навчання потребує великих обсягів даних, оскільки працює з більшою кількістю параметрів і багатими на інформацію датасетами для належного навчання. Саме тому використання глибокого навчання є виправданим для задач, які передбачають доступ до великих обсягів текстових даних.

Крім того, машинне навчання зазвичай застосовується для простіших завдань *NLP*, таких як аналіз тональності тексту або базова текстова класифікація. Водночас глибоке навчання дає змогу вирішувати складніші завдання, зокрема контекстне розуміння тексту, генерацію тексту чи машинний переклад. Завдяки здатності глибоких моделей обробляти контекстуальні залежності та генерувати текст на основі складної семантики вони стали ключовими інструментами у сучасних *NLP*-додатках.

З точки зору ресурсів машинне навчання є менш вимогливим до обчислювальної інфраструктури, адже його моделі можна ефективно тренувати на стандартному обладнанні, наприклад на центральних процесорах (*Central Processing Unit, CPU*). Глибоке навчання, навпаки, потребує потужних графічних процесорів (*Graphics Processing Unit, GPU*) і великих обсягів пам'яті, що робить його застосування ресурсозатратним. Це обмежує доступність глибокого навчання для організацій із недостатньою технічною базою.

Отже, машинне навчання залишається важливим інструментом для розв'язання простіших задач *NLP*, особливо за обмежених ресурсів чи даних. Однак для складних задач, які потребують глибокого контекстного розуміння та роботи з великими обсягами тексту, глибоке навчання є значно ефективнішим підходом, хоча й потребує значних обчислювальних потужностей і ресурсів. Ці дві методології часто доповнюють одна одну, забезпечуючи гнучкість і різноманіття у вирішенні завдань *NLP*.

15.1. Традиційні методи і моделі машинного навчання

До традиційних методів і моделей машинного навчання в обробці природної мови належить низка підходів та алгоритмів, які були популярними до широкого впровадження глибокого навчання.

Особливості роботи традиційних методів машинного навчання

До головних традиційних методів машинного навчання можна віднести такі [15.1—15.5]:

1. ***Методи, основані на правилах:***

• *методи, побудовані на регулярних виразах.* Використовуються для шаблонного пошуку та заміни слів у текстах;

• *методи, побудовані на граматичних правилах.* Ручне створення набору граматичних правил для аналізу тексту.

2. ***Статистичні методи:***

• *метод логістичної регресії.* Ґрунтується на концепції логістичної (сигмоїдної) функції, добре придатний для завдань, де потрібно класифікувати об'єкти на два або більше класів;

• *метод наївного Байєса.* Використовується для класифікації текстів на основі аналізу ймовірностей;

• *прихований розподіл Діріхле (Latent Dirichlet Allocation, LDA).* Зручний метод для тематичного моделювання з метою знаходження прихованих тем у документах;

• *приховані марковські моделі (Hidden Markov Models, HMM).* Ймовірнісні моделі, які визначають наступний стан системи на основі поточного стану, даючи можливість ефективно моделювати послідовності даних.

3. ***Методи на основі частотного аналізу:***

• *TF-IDF (Term Frequency-Inverse Document Frequency).* Метод, що оцінює важливість слова в контексті документа або набору документів.

4. ***Методи кластеризації та зниження розмірності:***

• дерево рішень. Використовується для розбиття даних на підмножини, щоб максимізувати приріст інформації й однорідність підмножин на кожному рівні;

• *K-means кластеризація*. Використовується для групування подібних документів;

• *Principal Component Analysis (PCA)*. Зручний метод для зменшення розмірності даних з метою полегшення їх аналізу.

5. ***Методи на основі матриць:***

• *Singular Value Decomposition (SVD)*. Використовується для зменшення розмірності та виявлення латентної структури в даних.

6. ***Методи на основі векторних зображень:***

• *Bag of Words (BoW)*. Подання тексту у вигляді вектора частот слів;

• *Word2Vec*. Метод для отримання векторних зображень слів (ембединг).

Таблиця 15.1

Порівняльні характеристики головних груп традиційних методів машинного навчання

Група методів	Переваги	Недоліки	Масштаби застосування	Оцінка масштабів застосувань за версією *ChatGPT* — шкала Міллера (1—8)
Методи, основані на правилах	Простота реалізації; висока точність для вузькоспеціалізованих завдань	Трудомістке ручне створення правил; слабка масштабованість	Невеликі або вузькі завдання	2
Статистичні методи	Добре придатні для аналізу текстів; ефективні для невеликих наборів даних	Обмеженість для складних завдань; недостатньо враховують контекст	Середні за складністю задачі	5
Методи на основі частотного аналізу	Простота впровадження; зручність для оброблення великих корпусів текстів	Ігнорування порядку слів і контексту	Масштабні завдання в аналітиці	6
Методи кластеризації та зниження розмірності	Допомагають виділяти приховані структури в даних; зменшують розмірність для візуалізації	Чутливість до початкових параметрів (наприклад, кількості кластерів)	Великий аналіз текстових даних	4
Методи на основі матриць	Ефективність для виявлення латентних структур; застосовність для зменшення розмірності	Не завжди інтерпретовані результати; потребують значних ресурсів для великих даних	Завдання з об'ємними даними	3
Методи на основі векторних зображень	Здатність зберігати семантичні зв'язки між словами; масштабованість для великих корпусів текстів	Обмеженість у відображенні контексту (*BoW*); залежність від якості даних під час навчання (*Word2Vec*)	Складні завдання NLP	8

Порівняльні характеристики головних груп традиційних методів машинного навчання наведено в табл. 15.1. Оцінку масштабів застосувань виконано за шкалою Міллера (8 — найбільший, 1 — найменший масштаб застосувань).

Цей порівняльний аналіз свідчить, що методи, основані на правилах, мають вузьку спеціалізацію, їх застосування обмежене завданнями з чітко структурованими вимогами. Статистичні методи ширше використовуються для базових завдань *NLP*, але поступаються більш сучасним підходам. Методи частотного аналізу (наприклад, *TF-IDF*) є популярними завдяки своїй простоті й ефективності у масштабних завданнях аналізу тексту.

Разом з тим методи кластеризації та зниження розмірності актуальні для аналізу великих текстових корпусів і візуалізації даних, але менш універсальні. Методи на основі матриць використовуються для специфічних завдань, таких як обробка латентних структур у великих даних. Методи на основі векторних зображень (особливо *Word2Vec*) забезпечують гнучкість і ефективність для складних завдань *NLP*, що робить їх найбільш масштабованими.

Окремі приклади традиційних методів машинного навчання

Розглянемо роботу деяких традиційних методів машинного навчання.

1. **Метод логістичної регресії** (група статистичних методів) [15.6]. Цей метод є потужним і часто використовуваним інструментом для завдань класифікації даних у машинному навчанні. Він ґрунтується на концепції логістичної функції (сигмоїдної функції) і добре придатний для завдань, де потрібно класифікувати об'єкти на два або більше класів. Логістична функція інтерпретується як імовірність належності до певного класу. Це інструмент, що дає змогу передбачати ймовірність належності об'єкта до певної категорії, ґрунтуючись на його характеристиках. Його можна уявити як місток між математичним моделюванням і практичним вирішенням завдань аналізу даних.

Основна ідея логістичної регресії полягає в тому, щоб знайти спосіб, який показує, як набір характеристик впливає на прийняття рішення про належність до тієї чи іншої групи. Наприклад, у медицині логістичну регресію можна використати для передбачення ймовірності захворювання на основі результатів аналізів. Вона допомагає не просто класифікувати, а оцінити впевненість у рішенні, надаючи ймовірності для кожної категорії.

Одна з ключових переваг методу логістичної регресії — це його простота. Логістична регресія добре працює у разі, коли залежність між характеристиками об'єкта і результатом є досить чіткою і простою. Вона не потре-

бує великих обчислювальних ресурсів і легко інтерпретується, що робить її популярним вибором для аналізу даних у багатьох галузях. Крім того, метод дає змогу враховувати зайві або надлишкові характеристики, завдяки чому можна уникати перенавчання. Це досягається через спеціальні механізми, які унеможливлюють надмірне ускладнення моделі.

Проте метод має і певні обмеження. Логістична регресія менш ефективна для складних завдань, де залежності між даними не є лінійними. Для таких випадків її можливості можна розширити через додаткові перетворення даних або використання інших, складніших алгоритмів.

Реалізація логістичної регресії у бібліотеці *scikit-learn* робить її ще більш доступною для широкого використання. Ця бібліотека надає простий у використанні інструмент для роботи з логістичною регресією. Вона дає можливість швидко налаштувати модель, підібрати оптимальні параметри і отримати точний результат. Особливо корисним є те, що *scikit-learn* дає змогу інтегрувати логістичну регресію в складніші обчислювальні процеси, що робить її універсальним рішенням для аналізу даних.

Отже, *логістична регресія — це не лише метод, а й філософія пошуку балансу між простотою і точністю.* Метод наочно демонструє, як можна використовувати математичні моделі для вирішення реальних завдань, де важлива не тільки відповідь, а й розуміння, наскільки впевнена модель у своєму рішенні. Це робить її невід'ємною частиною сучасного інструментарію машинного навчання.

Наведемо приклад реалізації методу логістичної регресії з використанням бібліотеки *scikit-learn* та мови програмування *Python* у конкретному завданні класифікації даних [15.7]. Зазначимо, що *Python* є високорівневою, об'єктно орієнтованою мовою програмування, відомою своєю простотою, читабельністю і широкими можливостями. Вона була створена нідерландським програмістом Гвідо ван Россумом (Guido van Rossum) у 1991 р. і на сьогодні є однією з найпопулярніших мов програмування у світі.

Приклад. Запишемо код у *Python* і проілюструємо різні його частини.

1. ***Розділення даних на тренувальні та тестові вибірки:***

python

Копіювати код

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
state=42)
```

• Цей рядок ілюструє процес підготовки даних. Набір даних X (характеристики) і Y (мітки або класи) розбивають на тренувальну (80 %) і тестову (20 %) частини. Це потрібно для перевірки якості моделі на нових, не відомих їй даних;

• параметр random_state=42 забезпечує відтворюваність результатів.

2. ***Створення і налаштування моделі логістичної регресії:***

```
python
Копіювати код
model = LogisticRegression(penalty='l2', solver='lbfgs', C=1.0, random_state=42)
```

• Визначається модель логістичної регресії з регуляризацією L2 (для запобігання перенавчанню) та методом оптимізації lbfgs;

• параметр C=1.0 визначає рівень регуляризації: менші значення збільшують силу регуляризації.

3. ***Навчання моделі:***

```
python
Копіювати код
model.fit(X_train, y_train)
```

• Модель тренується на тренувальних даних. Вона «вчиться» визначати залежність між характеристиками Xtrain та мітками Ytrain.

4. ***Прогнозування на тестовій вибірці:***

```
python
Копіювати код
y_pred = model.predict(X_test)
```

• Модель застосовується до тестових даних, щоб передбачити, до якого класу належить кожен об'єкт тестової вибірки.

5. ***Оцінка точності моделі:***

```
python
Копіювати код
accuracy = accuracy_score(y_test, y_pred)
print(f»Accuracy: {accuracy:.2f}»)
```

• Використовується метрика точності (*accuracy*) для оцінки якості моделі, тобто частки правильних передбачень на тестовій вибірці;

• значення точності виводиться у зручному форматі.

Цей приклад ілюструє повний цикл машинного навчання і зокрема всі етапи вирішення завдання класифікації, від підготовки даних до оцінки якості моделі. Він показує, наскільки легко можна налаштувати, навчити і застосувати модель логістичної регресії за допомогою бібліотеки *scikit-learn*.

2. **Метод наївного Байєса** в завданнях обробки природної мови (група статистичних методів) ґрунтується на теоремі Байєса та припущенні про незалежність ознак [15.8]. Основна ідея методу полягає в оцінці ймовірності того, що певний текст належить до конкретної категорії, враховуючи розподіл слів у тексті. Метод використовує алгоритм контрольованої класифікації, який знаходить умовний розподіл імовірностей P(*label* / *text*) за

допомогою формули Байєса

$$P(C|X) = P(X|C) \cdot P(C)/P(X),$$

де P(C|X) — ймовірність класу C для тексту X; P(X|C) — ймовірність спостереження тексту X за умови класу C; P(C) — ймовірність класу C (апріорна ймовірність); P(X) — загальна ймовірність тексту X.

При цьому робиться *припущення про незалежність ознак*, тобто слова в тексті розглядаються як незалежні одне від одного, що дає змогу спростити обчислення P(X|C) у вигляді добутку ймовірностей кожного слова:

$$P(X|C) = P(w_1|C) \cdot P(w_2|C) \cdot \ldots \cdot P(w_n|C),$$

де w_1, w_2, …, w_n — слова у тексті.

До найпоширеніших застосувань методу наївного Байєса в задачах *NLP* належать завдання класифікації тексту (наприклад, аналізу тональності, категоризації документів), завдання фільтрації спаму (визначення, чи є електронний лист спамом), розпізнавання мов, пошуку помилок у програмному коді [15.9].

Робота методу наївного Байєса дуже проста і зводиться до розрахунку P(C) як частки документів у класі «C» з наступним обчисленням P(w|C) як частоти слова «w» у текстах класу «C» (з використанням згладжування Лапласа для уникнення нульових значень).

Перевагами методу є висока швидкість роботи та ефективність, але на невеликих даних. До недоліків можна віднести необхідність робити значне припущення про незалежність слів, що не завжди відповідає реальним даним. Разом з тим метод наївного Байєса залишається популярним у завданнях *NLP* завдяки своїй ефективності, особливо для задач із великою кількістю класів і лінійно роздільними даними.

Наведемо приклад реалізації методу наївного Байєса для класифікації текстів з використанням мови *Python* та бібліотеки *sklearn* (для автоматизації розрахунків).

Приклад. **Класифікація тексту (спам / не спам)**

```
python
Копіювати код
fromsklearn.feature_extraction.textimportCountVectorizer
fromsklearn.model_selectionimporttrain_test_split
fromsklearn.naive_bayesimportMultinomialNB
fromsklearn.metricsimportaccuracy_score

# Дані: повідомлення та їхні мітки (1 — спам, 0 — не спам)
texts = [
    «Win a freelotterynow», «Callmetomorrow»,
```

```
    «Congratulations, youwon a prize!», «Let'smeetforlunch»,
    «Limitedoffer, buynow», «Canwediscusstheproject?»,
    «Exclusivedealjustforyou», «Seeyouatthemeeting»
]
labels = [1, 0, 1, 0, 1, 0, 1, 0]

# Перетворення тексту на числові ознаки
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(texts)

# Розподіл даних на навчальний і тестовий набори
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.25,
random_state=42)

# Ініціалізація та навчання моделі наївного Байєса
model = MultinomialNB()
model.fit(X_train, y_train)

# Прогноз на тестових даних
y_pred = model.predict(X_test)

# Оцінка точності
accuracy = accuracy_score(y_test, y_pred)
print(«Точність моделі:», accuracy)

# Тестування нових повідомлень
new_texts = [«Freegiftforyou», «Canwemeettomorrow?»]
new_X = vectorizer.transform(new_texts)
predictions = model.predict(new_X)

fortext, labelinzip(new_texts, predictions):
print(f»Повідомлення: '{text}' -> {'Спам' iflabel == 1 else 'Не спам'}»)
```

Результати. Цей приклад ілюструє базовий підхід до використання методу наївного Байєса у завданнях *NLP*, зокрема класифікацію текстових повідомлень як «Спам» або «Не спам».

Пояснення деяких елементів коду:

1. CountVectorizer — перетворює текст на матрицю частоти термінів (кількість слів у тексті).

2. MultinomialNB — реалізація наївного Байєса для текстових даних.

3. train_test_split: Розподіляє дані на навчальний і тестовий набори.

4. Мітки (*labels*) — позначають, чи є повідомлення спамом.

Висновок. У прикладі використовується *CountVectorizer*, що перетворює текстові дані на числову форму за допомогою побудови матриці частоти тер-

мінів, яка є основою для подальшої класифікації. Далі модель *MultinomialNB*, яка реалізує наївний Байєс для дискретних даних, навчається на навчальному наборі даних, що забезпечує спроможність розрізняти спам і не спам; *train_test_split* використовується для розподілу даних на навчальний і тестовий набори, що дає змогу оцінити точність моделі на даних, які вона раніше не бачила. Модель продемонструвала свою ефективність, забезпечивши точність класифікації (наприклад, точність 0,75 залежно від тестового набору), що свідчить про її здатність коректно класифікувати більшість повідомлень. Для додаткового тестування модель було використано на нових повідомленнях, де вона з точністю визначила, чи є вони спамом, чи ні.

3. **Дерево рішень** (група методів кластеризації та зниження розмірності) є одним із основних алгоритмів традиційного машинного навчання, який часто використовують для завдань обробки природної мови [15.10]. Основна ідея методу полягає в розподілі даних на підмножини на основі критеріїв, визначених у вузлах дерева, так, щоб максимізувати збільшення інформації і однорідність підмножин у цих розбиттях на кожному рівні.

У задачах обробки природної мови *дерево рішень застосовується для*:

• класифікації тексту. Дерево рішень можна використовувати для класифікації тексту за різними категоріями, наприклад, спам / не спам, позитивний / негативний відгук тощо;

• розпізнавання частин мови (Part-of-Speech *Tagging, POS tagging).* Дерево рішень можна використовувати для визначення частин мови для кожного слова в реченні;

• аналізу сентименту. Використовуючи ключові слова та інші текстові характеристики, дерево рішень може допомогти у визначенні настрою тексту (позитивний, негативний, нейтральний);

• розпізнавання іменованих сутностей (*Named Entity Recognition, NER).* Дерево рішень можна використати для класифікації слів або фраз у тексті як іменованих сутностей (наприклад, імена, місця, організації).

Процес побудови дерева рішень

1. *Вибір ознак.* Спочатку обирають ознаки, за якими буде виконуватись розподіл. Це можуть бути окремі слова, фрази або статистичні показники, як-от *TF-IDF*.

2. *Побудова дерева.* Дерево будують за допомогою рекурсивного розподілу даних на підмножини, виходячи з обраних ознак, доти, доки не досягається заданий критерій зупинки (наприклад, максимальна глибина дерева або мінімальна кількість даних у листі).

3. *Оцінка якості розбиття.* Використовують метрики, такі як інформаційний приріст (*information gain*) або коефіцієнт Джині (*Gini index*), для оцінки якості розподілу на кожному кроці.

Переваги методу дерева рішень. Результати легко інтерпретувати й візуалізувати; метод не потребує значної попередньої обробки даних; може працювати як з числовими, так і з категорійними даними.

Недоліки. Схильність до перенавчання (*overfitting*), особливо на великих і складних наборах даних; метод може створювати дуже глибокі дерева, які важко інтерпретувати.

Приклад. Розглянемо приклад використання методу дерева рішень для класифікації текстів (наприклад, визначення, чи є текст позитивним, чи негативним за тональністю) [15.11]. Виконаємо класифікацію тональності тексту (позитивний / негативний) з використанням методу дерева рішень і його реалізації мовою *Python*.

Запишемо код мовою *Python* для вирішення цього завдання:

```
python
Копіювати код
# Імпортуємо необхідні бібліотеки
fromsklearn.feature_extraction.textimportCountVectorizer
fromsklearn.model_selectionimporttrain_test_split
fromsklearn.treeimportDecisionTreeClassifier
fromsklearn.metricsimportaccuracy_score, classification_report

# Дані
texts = [
    «I lovethisproduct, it'samazing!»,
    «ThisistheworstthingI'veeverbought.»,
    «Absolutelyfantastic, I willbuyitagain!»,
    «Terribleexperience, notworththemoney.»,
    «I'mveryhappywiththispurchase.»,
    «I hatethis, itbrokeafteroneuse.»
]
labels = [1, 0, 1, 0, 1, 0] # 1 - позитивний, 0 - негативний

# Перетворення текстів у числовий вигляд за допомогою BagofWords
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(texts)

# Розбиваємо дані на тренувальний і тестовий набори
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.3,
random_state=42)

# Створення та навчання моделі дерева рішень
classifier = DecisionTreeClassifier(random_state=42)
classifier.fit(X_train, y_train)
```

```
# Прогнозування
y_pred = classifier.predict(X_test)

# Оцінка моделі
accuracy = accuracy_score(y_test, y_pred)
print(f»Accuracy: {accuracy:.2f}»)
print(«ClassificationReport:»)
print(classification_report(y_test, y_pred))

# Перевірка на нових текстах
new_texts = [
    «I reallyenjoyedthisproduct, greatquality!»,
    «TheworstpurchaseI'veevermade, awful!»
]
new_X = vectorizer.transform(new_texts)
new_predictions = classifier.predict(new_X)
fortext, predinzip(new_texts, new_predictions):
sentiment = «Positive» ifpred == 1 else «Negative»
print(f»Text: '{text}' =>Sentiment: {sentiment}»)
```

Отже, реалізацію методу дерева рішень алгоритмічною мовою *Python* виконано за такими етапами:

1. ***Підготовка даних.*** Використовують простий набір текстів та їх мітки (позитивна чи негативна тональність).

2. ***Перетворення тексту.*** Застосовують CountVectorizer для переведення тексту в числовий вигляд (*Bag of Words*).

3. ***Модель дерева рішень.*** Виконують навчання дерева рішень для класифікації текстів.

4. ***Оцінка моделі.*** Обчислюють точність і створюють класифікаційний звіт.

5. ***Тестування на нових текстах.*** Прогнозують тональність для нових текстів.

Висновок. У цьому прикладі текстові дані спочатку перетворюються на числове зображення за допомогою *CountVectorizer*, що створює матрицю частоти термінів (*Bag of Words*). Потім дані розподіляють на тренувальний і тестовий набори для забезпечення об'єктивної оцінки моделі. Використовуючи *DecisionTreeClassifier* із бібліотеки *Scikit-learn*, модель навчається класифікувати текстові дані, а її точність оцінюється за допомогою метрик точності та класифікаційного звіту. Результати навчання моделі показали адекватну продуктивність на тестових даних, що свідчить про її здатність правильно розрізняти позитивну і негативну тональність текстів. У прикладі нові текстові дані також було успішно класифіковано, що підтверджує коректність роботи моделі.

4. **Прихований розподіл Діріхле (*Latent Dirichlet Allocation, LDA*)** (група статистичних методів) — це потужний статистичний метод для автома-

тичного виявлення тем у великих колекціях текстових даних та документів. *LDA* припускає, що кожен документ є сумішшю декількох тем, а кожна тема характеризується розподілом слів [15.12]. Він допомагає зрозуміти структуру текстів і виділити ключові теми, що можуть бути корисними для різноманітних задач — аналіз контенту, рекомендаційні системи та ін.

Головні визначення LDA

1. *Розподіл Діріхле.* Використовується для моделювання розподілу тем у документах і слів у темах; задається двома параметрами: альфа (α) для розподілу тем у документах і бета (β) для розподілу слів у темах.

2. *Теми (Topics).* Тема — це розподіл слів, де кожне слово має певну ймовірність бути вибраним; кількість тем задається як параметр моделі.

3. *Слова (Words).* Кожне слово в документі стосується певної теми.

4. *Імпорт бібліотек. Gensim* — популярна бібліотека для обробки природної мови, яка містить реалізацію *LDA*.

5. *Документи.* Набір текстових документів, які будуть використані для побудови моделі.

6. *Попередня обробка тексту.* Токенізація документів (розподіл на слова та перетворення в нижній регістр); створення словника та корпусу.

7. *Словник.* Перетворення унікальних слів на числові ідентифікатори.

8. *Корпус.* Перетворення кожного документа на вектор частоти слів (*bag-of-words*).

9. *Створення моделі LDA.* Визначаються кількість тем (*num_topics*), ідентифікатори слів (*id2word*) та кількість проходжень алгоритму над даними (*passes*).

10. *Виведення тем.* Виводяться знайдені теми з відповідними словами та їх імовірностями.

Принципи роботи LDA

1. *Робляться припущення.* Кожний документ складається з декількох тем; кожна тема характеризується розподілом слів.

2. *Здійснюється процес генерації документів (згідно з LDA).* Для кожного документа вибирається розподіл тем, специфічний для цього документа; для кожного слова в документі вибирається тема згідно з розподілом тем документа та вибирається слово згідно з розподілом слів для цієї теми.

Процес навчання LDA полягає у знаходженні таких розподілів тем для кожного документа і розподілів слів для кожної теми, які найкраще пояснюють спостережувані дані. Навчання здійснюється за таким алгоритмом:

1) *ініціалізація.* Випадково призначаються початкові теми для кожного слова в кожному документі;

2) *ітеративне оновлення.* Для кожного слова в кожному документі оновлюється ймовірність належності цього слова до кожної теми; врахо-

вуючи поточний стан моделі, перепризначається тема відповідно до цього слова на основі оновлених імовірностей;

3) *досягнення збіжності*. Процес повторюється до досягнення збіжності (тобто коли розподіли більше суттєво не змінюються на наступних ітераціях).

Розглянемо приклад використання *LDA* для виявлення прихованих тем у текстових даних. Основна мета — визначити набір тем, які найкраще описують дані, та розподілити документи за цими темами.

Приклад. Для виконання цього завдання необхідно провести тематичне моделювання (*Topic Modeling*), за якого визначаються умови роботи.

1. ***Вхідні дані:***

• набір текстів (документів), де кожний документ складається зі слів.

2. ***Вихідні дані:***

• набір тем, що характеризують сукупність текстів;

• ключові слова для кожної теми, які визначають її зміст;

• розподіл імовірностей, які показують, до яких тем належать документи.

3. ***Завдання:***

• у наборі текстів виявити речення, що стосуються футболу та баскетболу;

• знайти дві основні теми, які об'єднують ці тексти.

4. ***Робота LDA у цьому випадку:***

4.1. *Попереднє оброблення тексту:*

• тексти розподіляються на слоти і перетворюються на числовий вигляд (*Bag of Words*).

4.2. *Виконується побудова словника та корпусу:*

• створюється словник, який містить усі унікальні слова в текстах;

• кожен документ зображується як вектор частот слів.

5. ***Застосовується LDA-модель:***

• модель визначає дві теми (задано num_topics=2);

• для кожної теми знаходяться слова, які мають найбільшу вагу (найбільшу ймовірність бути частиною цієї теми).

6. ***Отримуємо результат:***

• ключові слова для кожної з двох тем. Наприклад, одна тема може бути пов'язана з футболом, інша — з баскетболом.

Приклад виводу:

plaintext
Копіювати код
Topic: 0
Words: 0.1*football + 0.08*sport + 0.07*playing + ...
Topic: 1
Words: 0.09*basketball + 0.07*games + 0.06*watching + ...

Висновок. У цьому завданні на основі вхідного набору текстів було побудовано модель, що визначила дві основні теми: одна пов'язана з футболом, інша — з баскетболом. Процес роботи *LDA* містить кілька важливих етапів: попереднє оброблення тексту (розподіл на слова, приведення до числового вигляду через *Bag of Words*), створення словника та корпусу текстів, а також навчання моделі з параметром num_topics=2, який визначає кількість тем. Результати роботи *LDA* відображають ключові слова, які найкраще характеризують кожну з тем. Наприклад, для першої теми основними словами є football, sport, playing, тоді як для другої — basketball, games, watching. Це дає змогу легко інтерпретувати зміст тем і їх зв'язок із документами.

5. **Метод прихованих марковських моделей (*Hidden Marcov Models, HMM*)** (група статистичних методів) — потужний інструмент у багатьох аспектах обробки природної мови. Це ймовірнісні моделі, які визначають наступний стан системи на основі поточного стану, оскільки вони дають можливість ефективно моделювати послідовності даних.

Наприклад, у задачах *NLP* часто необхідно запропонувати наступне слово на основі попереднього слова. Цей процес можна змоделювати за допомогою марковської моделі, де потрібно знайти ймовірності переходу від word1 до word2, тобто $P(\text{слово}_1/\text{слово}_2)$. Тоді можна використати добуток цих імовірностей переходів, щоб знайти ймовірність словосполучення. Тобто прихована марковська модель — це імовірнісна техніка моделювання, яка вводить прихований стан у марковську модель. Прихований стан — це властивість даних, яка безпосередньо не спостерігається.

Зокрема, *HMM* можна використовувати для тегування частин мови (*Part-of-Speech tagging*, або *POS tagging*), тобто автоматичного визначення та позначення частин мови для кожного слова в тексті, де слова в реченні є спостережуваними станами, а *POS*-теги — прихованими станами. Частини мови — це іменники, дієслова, прикметники, прислівники та інші граматичні категорії.

Тегування частин мови є важливим кроком у багатьох завданнях обробки природної мови, таких як синтаксичний аналіз, розпізнавання іменованих сутностей, машинний переклад та ін. При цьому *HMM* ґрунтується на поняттях «Імовірність викидів» та «Імовірність спостереження» в прихованому стані. На практиці це завдання вирішується за допомогою алгоритму *Вітербі* (*Viterbi Algorithm*), який використовується для пошуку найімовірнішої послідовності прихованих станів у прихованих марковських моделях, виходячи з наявної послідовності спостережень [15.13].

Приховані марковські моделі є потужним інструментом для різних аспектів *NLP*, як-от розпізнавання мови, розпізнавання іменованих сутностей, корекції орфографії та машинного перекладу. Наведемо приклади декількох ключових застосувань *HMM*.

1. ***Розпізнавання мовлення.*** Методи *HMM* використовують для автоматичного розпізнавання мови (*Automatic Speech Recognition, ASR*), де вони допомагають моделювати акустичні сигнали та їх перетворення на текст. Зокрема, розширені версії *HMM*, такі як факторні *HMM*, застосовуються для поліпшення точності розпізнавання мови в різних умовах навколишнього середовища.

2. ***Токенізація і тегування частин мови.*** *HMM* можуть допомагати у визначенні частини мови (іменник, дієслово тощо) для кожного слова в тексті, що є критично важливим кроком у багатьох задачах *NLP*.

3. ***Аналіз почуттів і тональності тексту.*** *HMM* можна використовувати для моделювання імовірностей того, що певні слова або фрази виражають позитивні чи негативні емоції.

4. ***Розпізнавання сутностей.*** Це завдання полягає у визначенні імен, місць, організацій та інших важливих сутностей у тексті. *HMM* можна використати для передбачення ймовірності того, що певні послідовності слів є сутностями.

Ключові застосування методу

Застосування *HMM*	Головні сфери застосування	Переваги застосування	
Розпізнавання мовлення	Акустика та лінгвістика	Висока точність під час моделювання послідовностей акустичних сигналів	
Токенізація і тегування частин мови	Обробка тексту	Простота реалізації, ефективність для коротких текстів	
Аналіз почуттів та тональності тексту	Обробка тексту	Ефективне моделювання послідовностей емоційних маркерів	
Розпізнавання сутностей	Обробка тексту	Висока ефективність для текстів з формальними структурами	
Автоматичний переклад	Машинний переклад	Покращення якості вирівнювання мовних пар	
Корекція орфографії та граматики	Лінгвістика	Добре придатний для простих випадків з відомими правилами	
Сегментація тексту	Обробка тексту	Швидкість і точність у стандартних текстах	
Розпізнавання іменованих сутностей (*NER*)	Обробка тексту	Висока точність у стандартних наборах даних	
Корекція орфографії	Лінгвістика	Можливість контекстного виправлення помилок	
Машинний переклад	Машинний переклад	Покращення узгодженості перекладу завдяки моделям послідовності	

5. ***Автоматичний переклад.*** *HMM* можна використовувати для моделювання послідовності слів у різних мовах, допомагаючи системам перекладу визначати найбільш точні переклади.

6. ***Корекція орфографії та граматики.*** *HMM* можуть допомагати визначати найімовірніші варіанти корекції для слів або фраз, що містять помилки.

7. ***Сегментація тексту.*** В задачах, де потрібно розподілити текст на речення або абзаци, *HMM* можна використовувати для визначення імовірностей розділення в певних місцях тексту.

8. ***Розпізнавання іменованих сутностей*** *(Named Entity Recognition, NER). HMM* ефективно використовують для ідентифікації іменованих сутностей у тексті, як-от імена людей, місця, організації тощо.

9. ***Корекція орфографії.*** *HMM* можна використати для автоматичної корекції орфографічних помилок, де моделі допомагають передбачити правильне написання слів на основі контексту та ймовірності переходу між різними станами (наприклад, правильним та неправильним написанням слів).

Таблиця 15.2

прихованих марковських моделей

	Недоліки застосування	Масштаби застосування за версією *ChatGPT,* шкала Міллера (1—8)
	Обмеження в разі роботи з великими шумами та складними мовами	**8**
	Обмежена точність для рідкісних слів або складних граматичних конструкцій	**7**
	Низька точність у неоднозначних текстах	**6**
	Складність оброблення неструктурованих текстів	**7**
	Неефективність для багатомовного середовища	**6**
	Обмежена здатність роботи з креативними текстами або помилками типу «словник»	**6**
	Може припускати помилки в текстах зі специфічною пунктуацією	**5**
	Низька продуктивність у складних або міжмовних текстах	**7**
	Складність оброблення нових або невідомих слів	**5**
	Обмеження для складних мов з багатозначністю	**6**

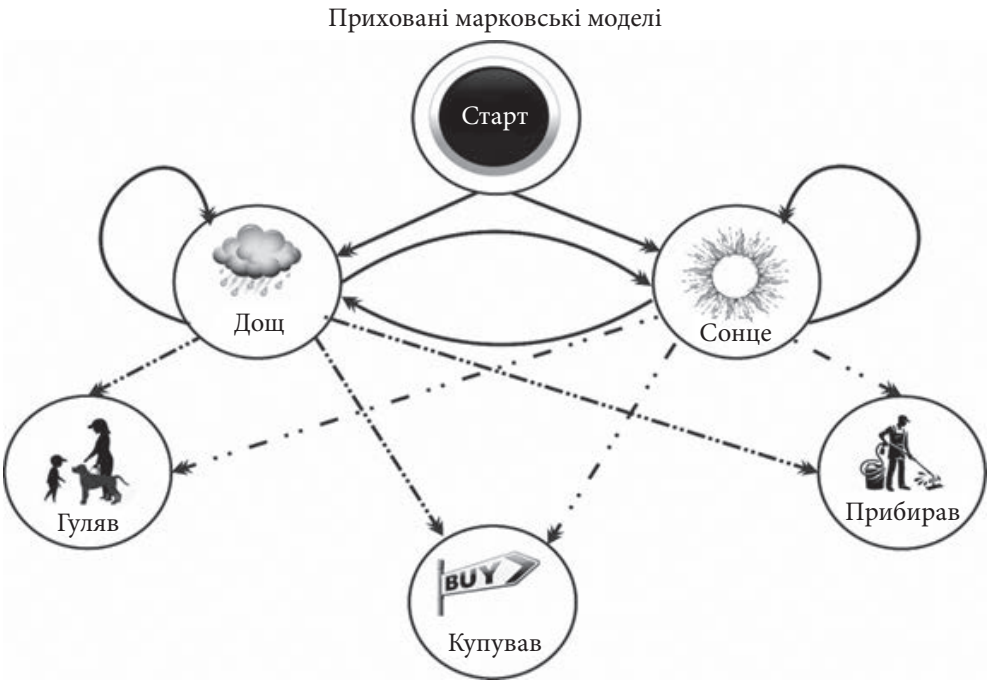


Рис. 15.1. Приклад застосування *HMM* для визначення прихованих станів (погоди) на певному відрізку часу в минулому [15.1]

10. ***Машинний переклад.*** В обробці природної мови *HMM* використовують для вирівнювання слів і фраз під час машинного перекладу. Це допомагає покращити якість перекладу, моделюючи ймовірність переходів між мовними парами.

Основні характеристики застосування *HMM* та їхні переваги й недоліки, що визначають їх ефективність у конкретних задачах, наведено в табл. 15.2.

Наведемо приклад застосування *HMM* (рис. 15.1) [15.1]. Припустімо, що людина намагається пригадати погоду минулого тижня на основі того, що вона робила в той чи інший час. Її дії є спостережуваними станами, а різні типи погоди є прихованими станами. Лінії між станами погоди показують імовірності переходу, а лінії від станів погоди до дій — імовірності того, що людина виконала певну дію з урахуванням погоди.

Приклад. Метод прихованих марковських моделей добре придатний для задач, в яких потрібно моделювати послідовності з прихованими (наприклад, тип погоди) і спостережуваними (наприклад, дії людини) станами [15.13]. Наведемо приклад реалізації такого підходу на *Python*.

Завдання. Моделювання погоди на основі дій людини за допомогою *HMM*.

Реалізація на Python:

python
Копіювати код

```
importnumpyasnp
fromhmmlearnimporthmm

# Визначення моделі
# Припустимо, приховані стани (погода): [‘Sunny’, ‘Rainy’, ‘Cloudy’]
# Спостереження (дії): [‘Walk’, ‘Shop’, ‘Clean’]

# Визначимо імена прихованих станів і спостережень
states = [‘Sunny’, ‘Rainy’, ‘Cloudy’]
observations = [‘Walk’, ‘Shop’, ‘Clean’]
n_states = len(states)
n_observations = len(observations)

# Імовірності початкових станів
start_probability = np.array([0.5, 0.3, 0.2]) # ймовірності, що тиждень почався із цих станів


# Матриця ймовірностей переходу між прихованими станами (погоди)
transition_probability = np.array([
    [0.6, 0.3, 0.1], # Sunny -> [Sunny, Rainy, Cloudy]
    [0.2, 0.6, 0.2], # Rainy -> [Sunny, Rainy, Cloudy]
    [0.3, 0.3, 0.4] # Cloudy -> [Sunny, Rainy, Cloudy]
])


# Матриця ймовірностей спостереження (що людина робила за певної погоди)
emission_probability = np.array([
    [0.8, 0.1, 0.1], # Sunny -> [Walk, Shop, Clean]
    [0.2, 0.6, 0.2], # Rainy -> [Walk, Shop, Clean]
    [0.4, 0.3, 0.3] # Cloudy -> [Walk, Shop, Clean]
])

# Створення HMM моделі
model = hmm.MultinomialHMM(n_components=n_states, n_iter=100, tol=0.01, random_state=42)
model.startprob_ = start_probability
model.transmat_ = transition_probability
model.emissionprob_ = emission_probability
```

```
# Спостережувана послідовність дій (індекси дій у списку observations)
# Наприклад, людина робила: ['Walk', 'Shop', 'Walk', 'Clean', 'Walk']
observed_sequence = [0, 1, 0, 2, 0] # Індекси дій
observed_sequence = np.array(observed_sequence).reshape(-1, 1)

# Використання моделі для передбачення прихованих станів (погоди)
log_prob, hidden_states = model.decode(observed_sequence,
algorithm=»viterbi»)

# Результати
decoded_weather = [states[state] forstateinhidden_states]
print(«ObservedActions:», [observations[i] for i inobserved_sequence.
flatten()])
print(«PredictedWeather:», decoded_weather)
```

Пояснення

Вхідні дані:

• *приховані стани (типи погоди)* ['Sunny', 'Rainy', 'Cloudy'];

• *спостереження (дії)* ['Walk', 'Shop', 'Clean'].

Параметри HMM:

• start_probability — ймовірність того, що тиждень починається з певного типу погоди;

• transition_probability — ймовірності переходів між станами погоди;

• emission_probability — ймовірності, що людина виконує певну дію залежно від погоди.

Застосування алгоритму Viterbi:

• використовується для визначення найімовірнішої послідовності прихованих станів на основі спостережуваних дій.

Виведення результату:

• *Observed Actions* — дії людини (['Walk', 'Shop', 'Walk', 'Clean', 'Walk']);

• *Predicted Weather* — найімовірніша послідовність погоди (['Sunny', 'Rainy', 'Sunny', 'Cloudy', 'Sunny']).

Висновок. Отримані результати свідчать про те, що за заданими діями людини (наприклад, прогулянки, покупки, прибирання) модель змогла визначити найімовірнішу послідовність погодних умов. Для цього використовувалися ймовірнісні матриця переходів станів та матриця спостережень (або емісійна матриця), які відображають, відповідно, ймовірності переходу між прихованими станами (типами погоди) та ймовірності спостереження певної дії за кожного типу погоди. Перевага прихованої марковської моделі (*HMM*) полягає у здатності моделювати послідовності даних із невизначеностями та враховувати залежності між прихованими станами. У цьому прикладі модель дала змогу реконструювати ймовірну картину змі-

ни погодних умов, враховуючи лише зовнішні дії людини. Подібні підходи широко застосовуються у задачах розпізнавання мовлення, біоінформатики, аналізу текстів та інших сферах, де необхідно враховувати приховані залежності у часових або послідовних даних.

Отже, цей підхід дає змогу визначити, яка погода могла бути кожного дня, виходячи зі спостережуваних дій людини (див. рис. 15.1). Ви можете змінювати вхідні дані, параметри ймовірностей або послідовності спостережень для тестування інших сценаріїв.

15.2. Методи і моделі глибокого навчання в обробці природної мови

Глибоке навчання є основою сучасних досягнень у сфері обробки природної мови, даючи змогу моделювати складні залежності в текстових даних і забезпечувати високий рівень автоматизації та точності у виконанні завдань [15.14]. Багатошарові нейронні мережі, покладені в основу цього підходу, відкривають можливості для аналізу тексту на різних рівнях абстракції — від розпізнавання слів і фраз до розуміння контексту та семантики тексту.

У завданнях *NLP* глибоке навчання демонструє виняткову ефективність завдяки своїй здатності автоматично виділяти релевантні ознаки тексту. Замість того, щоб покладатися на вручну створені ознаки, моделі глибокого навчання аналізують текст, виявляючи приховані структури та залежності. Це особливо важливо для таких завдань як класифікація тексту, машинний переклад, генерація тексту, аналіз настроїв і відповіді на запитання, де врахування контексту та розуміння багатозначності слів є критичними.

Один із найважливіших елементів глибокого навчання в *NLP* — це можливість працювати з великими обсягами текстових даних. Завдяки використанню графічних процесорів (*Graphics Processing Unit, GPU*) та сучасних алгоритмів оптимізації глибокі нейронні мережі можуть навчатися на масивних текстових корпусах, що забезпечує високу якість моделей. Наприклад, великі мовні моделі (*Large Language Models, LLM*), такі як *BERT (Bidirectional Encoder Representations from Transformers*) і *GPT (Generative Pre-Trained Transformer)*, використовують трансформерні архітектури, що сприяють врахуванню глобального контексту текстового масиву та моделюванню довготривалих залежностей [15.15].

Найбільший прорив у *NLP* стався з появою великих мовних моделей *(LLM)*, які здатні генерувати зв'язний і контекстуально релевантний текст. Вони використовують глибокі трансформерні архітектури, навчені на величезних корпусах даних, що дає їм змогу адаптуватися до широкого спектра завдань: від написання статей і коду до ведення діалогів із користувачами.

Отже, глибоке навчання стало основою сучасного *NLP*, забезпечуючи автоматизацію та підвищення якості виконання навіть найскладніших завдань. Його здатність аналізувати текст на різних рівнях, враховувати контекст і інтегрувати різні моделі уможливлює створення інноваційних рішень, які трансформують традиційне уявлення про обробку природної мови. Завдяки глибокому навчанню системи *NLP* наближаються до розуміння та використання мови на рівні, близькому до рівня людини.

Особливості роботи і використання методів та моделей глибокого навчання

Еволюція нейронних мереж стала ключовим етапом у розвитку сучасного штучного інтелекту, завдяки чому глибоке навчання зумовило істотний стрибок технологій обробки природної мови. Головні відмінності цих технологій порівняно з традиційним навчанням нейронних мереж полягають у п'яти основних характерних ознаках [15.14]: архітектурі, масштабуванні, особливостях навчання, автоматизації витягування ознак та продуктивності.

Так, традиційне навчання нейронних мереж ґрунтується на простих архітектурах, таких як одно- або двошарові перцептрони. Ці моделі обмежуються здатністю обробляти лише базові залежності в даних. Зі свого боку глибоке навчання використовує багатошарові нейронні мережі (*Deep Neural Networks, DNN*), які мають десятки, а іноді й сотні прихованих шарів. Ця багатошаровість дає змогу мережам вивчати складніші патерни та абстракції, уможливлюючи обробку таких складних завдань як розпізнавання мовлення, комп'ютерний зір та автоматичний переклад.

Одним із фундаментальних обмежень традиційних нейронних мереж є їхня неспроможність працювати з великими обсягами даних через обмеженість обчислювальних ресурсів. Глибоке навчання вирішує цю проблему завдяки застосуванню графічних процесорів (*GPU*), які ефективно виконують паралельні обчислення. Це дає глибоким мережам можливість обробляти масиви даних величезного розміру (*BigData*) і створювати моделі високої розмірності, що неможливо для традиційних мереж.

Традиційні нейронні мережі зазвичай навчалися на менших наборах даних, використовуючи базові методи оптимізації. Наприклад, для таких моделей часто застосовували прості варіанти градієнтного спуску. Глибоке навчання, навпаки, інтегрує сучасні алгоритми оптимізації, які забезпечують швидке і стабільне навчання на великих наборах даних. Крім того, сучасні методи регуляризації знижують ризик перенавчання і підвищують узагальнювальну здатність моделей.

Одним із ключових недоліків традиційних нейронних мереж є залежність від ручного проєктування ознак (*Feature Engineering*). Це надає про-

цесу складності та залежності від експертизи інженера. Глибоке навчання радикально змінює цей підхід, даючи моделям можливість автоматично виділяти релевантні ознаки з даних. Така автоматизація значно скорочує потребу в ручній роботі, що є особливо цінним під час оброблення високорівневих і неструктурованих даних, як-от зображення або аудіо.

Обмеження продуктивності традиційних нейронних мереж яскраво проявляються у складних завданнях, де виникає необхідість виявлення глибоких патернів. Глибоке навчання усунуло ці обмеження, демонструючи вражаючі результати у розпізнаванні образів, класифікації мовлення та аналізі тексту. Така перевага зумовлена здатністю глибоких мереж створювати багаторівневі подання даних, що значно перевершують за складністю традиційні підходи.

Бачимо, що глибоке навчання є еволюційним проривом, який дає змогу штучному інтелекту вийти на новий рівень складності й адаптивності. Здатність технологій глибокого навчання працювати з великими обсягами даних, автоматично вилучати ознаки та забезпечувати високу продуктивність у складних завданнях робить їх незамінними на сучасному етапі розвитку ШІ. Тим часом традиційні нейронні мережі залишаються важливими для базових завдань, проте вони поступово втрачають свою актуальність в умовах зростання вимог до точності, ефективності та гнучкості моделей [15.14].

Порівняння традиційного та глибокого навчання в задачах *NLP* за п'ятьма базовими характеристиками наведено в табл. 15.3. Це порівняння демонструє ключові відмінності між традиційним і глибоким навчанням, а також їхній вплив на вирішення завдань *NLP.*

На основі наведеного порівняння можна зазначити, що традиційне навчання нейронних мереж залишається актуальним для простих завдань, які не потребують великих обсягів даних чи складної обчислювальної архітектури. Разом з тим глибоке навчання є незамінним інструментом для сучасних завдань *NLP* — машинному перекладі, розпізнавання сутностей та аналізі настроїв тексту. Основна перевага глибокого навчання — здатність обробляти складні патерни та великі обсяги даних — компенсує його вимоги до обчислювальних ресурсів та складності реалізації.

Приклад. Наведемо дані кількісного порівняння відмінностей між глибоким навчанням і традиційним навчанням нейронних мереж у завданнях обробки природної мови. Розглянемо популярне завдання аналізу тональності (*Sentiment Analysis*) або завдання класифікації тексту. Це дає змогу порівняти базові характеристики глибокого і традиційного навчання нейронних мереж, такі як точність, ефективність використання ресурсів та обчислювальні потреби [15.16].

Завдання. Виконати аналіз тональності на наборі даних *IMDB (Internet Movie Database)* і класифікацію рецензій на позитивні та негативні.

Таблиця 15.3

Порівняння традиційного і глибокого навчання в завданнях NLP

Базова характеристика	Традиційне навчання	Глибоке навчання
Архітектура	Простіші моделі (одно- або двошарові перцептрони) *Переваги:* легка інтерпретація, швидка підготовка *Недоліки:* низька здатність до моделювання складних залежностей	Багатошарові нейронні мережі (*DNN*), рекурентні мережі (*RNN*), трансформери (наприклад, *BERT, GPT*) *Переваги:* можливість вивчати складні й абстрактні патерни *Недоліки:* високі вимоги до обчислювальних ресурсів
Масштабування	Ефективні на малих наборах даних *Переваги:* малі витрати ресурсів *Недоліки:* не масштабуються для великих наборів даних	Працюють з великими обсягами даних (*BigData*) завдяки *GPU* та сучасним обчислювальним платформам *Переваги:* здатність обробляти величезні набори текстових даних *Недоліки:* залежність від доступності великих обсягів даних та апаратних ресурсів
Особливості навчання	Просте навчання на невеликих наборах даних із базовими алгоритмами оптимізації (наприклад, градієнтний спуск) *Переваги:* менша складність навчання, швидкість реалізації *Недоліки:* схильність до перенавчання, низька ефективність у складних задачах	Навчання на великих наборах даних із сучасними алгоритмами (*Adam, RMSprop*), використання регуляризації (*Dropout, Batch Norm*) *Переваги:* висока узагальнювальна здатність, уникнення перенавчання завдяки регуляризації *Недоліки:* велика складність навчання, значні часові та обчислювальні витрати
Автоматизація вилучення ознак	Потребує ручного проєктування ознак, залежного від експертизи інженера *Переваги:* Контрольованість процесу *Недоліки:* часомісткість, залежність від людського фактора	Автоматичне вилучення ознак (наприклад, у глибоких мережах — *CNN* або трансформери) *Переваги:* мінімізація ручної роботи, вища ефективність у складних завданнях *Недоліки:* висока складність моделі може утруднити інтерпретацію
Продуктивність	Ефективні лише для простих задач (наприклад, базовий аналіз тексту, токенізація) *Переваги:* достатньо для задач низької складності *Недоліки:* низька ефективність для складних задач із великою кількістю змінних	Висока продуктивність у складних завданнях (розпізнавання мовлення, переклад, розпізнавання сутностей) *Переваги:* показують хороші результати у складних *NLP* завданнях *Недоліки:* значний вплив на ресурси (обчислювальні та енергетичні)

1. ***Традиційна нейронна мережа (NLP з TF-IDF-ознаками):***
• архітектура — одношаровий перцептрон з 1000 нейронів;
• ручні ознаки — *TF-IDF* на основі уніграм;
• точність — ~85 %;
• кількість параметрів — ~1 млн;
• час тренування — ~1 година на *CPU* (для середнього набору даних);
• обчислювальні ресурси — *CPU.*
2. ***Глибока нейронна мережа (BERT):***
• архітектура — попередньо навчена модель *BERT*, донавчена на завданні аналізу тональності;
• автоматичні ознаки — контекстуальні векторні зображення слів;
• точність — ~94 %;
• кількість параметрів — ~110 млн (*BERT base*);
• час тренування — ~1 година на *GPU* (для середнього набору даних);
• обчислювальні ресурси — *GPU.*

Результати порівняння базових характеристик традиційної нейронної мережі (*NLP* з *TF-IDF*-ознаками) і глибокої нейронної мережі (*BERT*) на

Таблиця 15.4

Порівняння базових характеристик традиційного і глибокого навчання на прикладі вирішення завдання класифікації тексту

Базова характеристика	Традиційна нейронна мережа (*NLP* з *TF-IDF*-ознаками)	Глибока нейронна мережа (*BERT*)
Архітектура	Одношаровий перцептрон з 1000 нейронів	Попередньо навчена модель *BERT*, донавчена на завданні аналізу тональності
Ознаки	Ручні (*TF-IDF* на основі уніграм)	Автоматичні (контекстуальні векторні подання слів)
Точність	~85 %	~94 %
Кількість параметрів	~1 млн	~110 млн (*BERT base*)
Час тренування	~1 година на *CPU* (для середнього набору даних)	~1 година на *GPU* (для середнього набору даних)
Обчислювальні ресурси	*CPU*	*GPU*
Ключові переваги	Простота реалізації; менші обчислювальні вимоги	Висока точність; автоматичне виділення контекстуальних ознак
Ключові недоліки	Менша точність; залежність від ручного проєктування ознак; неможливість урахування контексту	Великі обчислювальні ресурси; більша складність моделі; залежність від великої кількості пам'яті та потужного апаратного забезпечення

прикладі вирішення завдання аналізу тональності на наборі даних *IMDB* наведено в табл. 15.4.

Висновок. Отже, глибока модель *BERT* демонструє вищу продуктивність завдяки своїй здатності вивчати контекстуальні залежності в тексті. Водночас традиційна модель з *TF-IDF*-ознаками є ефективнішою у використанні ресурсів і простіша у реалізації, але поступається у точності та гнучкості. Вибір підходу залежить від доступності апаратного забезпечення та вимог до точності вирішення завдання.

Класифікація методів і моделей глибокого навчання

Методи і моделі глибокого навчання посідають центральне місце у сучасних дослідженнях і практичному застосуванні обробки природної мови (*NLP*). З огляду на архітектурний і функціональний підходи, враховуючи принципи роботи, їх можна розподілити на чотири основні класи: ***послідовні моделі, локальні моделі, моделі з механізмом уваги та генеративні моделі*** (табл. 15.5) [15.16—15.25].

Послідовні моделі зосереджені на обробленні даних, які мають часову чи текстову послідовність. Вони враховують залежність кожного елемента від попередніх, що робить їх ефективними для завдань, де контекст є критично важливим. Наприклад, у машинному перекладі чи аналізі настроїв послідовність слів відіграє ключову роль. Серед методів цього класу особливе місце посідають рекурентні нейронні мережі (*Recurrent Neural Networks, RNN*) та їх поліпшені версії — *Long Short-Term Memory (LSTM)* і *Gated Recurrent Unit (GRU)* [15.16—15.18]. Ці моделі дають можливість працювати з довгими послідовностями, хоча мають певні обмеження, зокрема високу обчислювальну складність та труднощі із захопленням дуже довготривалих залежностей.

Локальні (згорткові) моделі представлені насамперед згортковими нейронними мережами (*Convolutional Neural Network, CNN*). Їхня перевага полягає у виділенні локальних залежностей, що робить їх надзвичайно ефективними для класифікації тексту чи виявлення ключових фраз. Використовуючи операцію згортки, *CNN* аналізують фрагменти тексту, сприяючи швидкому й ефективному вирішенню завдання, де локальний контекст є достатнім. Однак обмеження цього класу полягають у неможливості врахування глобальних залежностей, що ускладнює їх використання в більш комплексних застосуваннях, як-от генерація тексту або переклад.

Моделі з механізмом уваги (*Attention Mechanism*), як-от *Transformers (Bidirectional Encoder Representations from Transformers BERT* і *Generative Pre-Trained Transformer GPT)*, стали революційними у світі *NLP* завдяки своїй здатності враховувати глобальний контекст [15.19—15.21]. Використовуючи механізм уваги, ці моделі одночасно аналізують усі елементи послідовності, що дає

Таблиця 15.5

Головні класи методів і моделей глибокого навчання у завданнях *NLP*

Клас	Методи і моделі	Загальна характеристика класу	Сфера застосування класу	Недоліки	Переваги
Послідовні моделі	*RNN, LSTM, GRU, Seq2Seq*	Обробляють послідовності даних з урахуванням контексту, зберігаючи інформацію про попередні елементи	Машинний переклад, мовне моделювання, аналіз настроїв, синтез тексту	Проблеми з довготривалими залежностями (*RNN*), висока обчислювальна складність (*LSTM, GRU*)	Добре захоплюють контекст у послідовностях, придатні для задач із часовою залежністю
Локальні (згорткові) моделі	*CNN*	Використовують операцію згортки для виділення локальних патернів у тексті	Класифікація тексту, виявлення ключових фраз, аналіз локальних властивостей тексту	Обмеження у захопленні довготривалих залежностей, не враховують глобальний контекст	Висока швидкість і ефективність у завданнях, де важливі локальні залежності
Моделі з механізмом уваги	*Transformer, BERT, GPT, Attention Mechanism*	Використовують механізм уваги для обробки послідовностей, враховують глобальний контекст через механізм уваги	Переклад, класифікація тексту, генерація тексту, відповіді на запитання	Висока обчислювальна вартість, особливо для довгих послідовностей	Успішне захоплення довготривалих залежностей, паралельне оброблення даних, універсальність у завданнях *NLP*
Генеративні моделі	*Auto-encoder, GPT*	Генерують нові дані, здійснюють стиснення інформації	Генерація тексту, виявлення аномалій, стиснення даних	Обмеження складністю вхідних даних	Потужна генерація даних

можливість ефективно працювати навіть із довгими текстами. Універсальність цих моделей робить їх придатними для широкого спектра задач — від класифікації до генерації тексту і відповідей на запитання. Хоча їхньою основною перевагою є потужність і гнучкість, обчислювальна вартість цих моделей є високою, що може обмежувати їх застосування в умовах обмежених ресурсів.

Окрему категорію становлять ***генеративні моделі,*** спрямовані на створення нових даних або стиснення інформації. До них належать автоенкодери та *GPT* [15.22—15.25]. Автоенкодери використовують для створення компактних подань тексту або виявлення аномалій, натомість *GPT* спеціалізується на створенні тексту, який важко відрізнити від тексту, написаного людиною. Генеративні моделі мають потужний потенціал у завданнях, пов'язаних із генерацією даних, проте їх застосування може бути обмежене складністю вхідної інформації.

Запропонована класифікація (табл. 15.5) дає змогу інтегрувати технічні аспекти роботи моделей із функціональними характеристиками, забезпечуючи баланс між теоретичною і практичною складовими та створюючи зручну основу для розуміння ролі й місця сучасних моделей і методів глибокого навчання у завданнях *NLP* та їх застосування.

У завданнях обробки природної мови методи і моделі глибокого навчання відіграють вирішальну роль у сучасних технологіях, даючи можливість вирішувати широкий спектр завдань: від класифікації тексту до генерації природного тексту [15.16—15.25]. Різні архітектури мають свої унікальні особливості, що визначають сфери їх застосування, їхні переваги й недоліки (табл. 15.6).

Рекурентні нейронні мережі (RNN) стали базовими для оброблення послідовних даних, як-от текст чи відео. Вони можуть працювати з послідовностями довільної довжини, що робить їх незамінними для мовного моделювання, аналізу настроїв та машинного перекладу. Проте проблеми з довготривалими залежностями через загасання градієнта обмежують їхню ефективність у складних завданнях.

Автоенкодери є окремим підходом до оброблення тексту, з орієнтуванням на зниження розмірності, виявлення аномалій та створення ембедингів тексту. Завдяки здатності навчатися без нагляду вони корисні для завдань стиснення даних, але обмеженість у захопленні складної семантики зменшує їхню ефективність у завданнях, які потребують глибшого розуміння контексту.

Для подолання цих обмежень було розроблено ***LSTM (Long Short-Term Memory) і GRU (Gated Recurrent Unit)***, які ефективно обробляють довготривалі залежності. *LSTM* забезпечує успішне моделювання складних взаємозв'язків у даних, хоча й потребує значних обчислювальних ресурсів. *GRU* пропонує спрощену архітектуру з подібною ефективністю, але мен-

Таблиця 15.6

Класифікація моделей і методів глибокого навчання у завданнях *NLP*

Метод / Модель	Сфера застосування	Переваги	Недоліки	Масштаби застосування за версією *ChatGPT*, шкала Міллера (1—8)
RNN	Обробка послідовностей	Гнучкість у роботі з послідовностями	Проблеми з довготривалими залежностями	5
Autoencoders	Зниження розмірності, ембедінги	Навчання без нагляду	Обмеженість у складній семантиці	4
LSTM	Довготривалі залежності	Подолання загасання градієнта	Висока обчислювальна складність	7
GRU	Довготривалі залежності	Менша складність порівняно з *LSTM*	Менш потужні у складних залежностях	6
CNN	Локальні патерни у тексті	Ефективність, менша складність	Обмеження у довготривалих залежностях	5
Seq2Seq	Переклад, генерація тексту	Гнучкість	Висока обчислювальна складність	6
Transformers	Сучасні *NLP*-задачі	Довготривалі залежності, паралельне оброблення	Висока обчислювальна вартість	8
BERT	Класифікація, пошук	Двоспрямований контекст	Висока обчислювальна вартість	8
GPT	Генерація тексту	Потужні генеративні можливості	Нерівномірна увага до контексту	8
Attention Mechanism	Універсальний компонент *NLP*	Захоплення глобального контексту	Велика обчислювальна складність	8

шою складністю. Обидві моделі широко застосовують для завдань машинного перекладу, генерації тексту та синтезу мови.

Згорткові нейронні мережі (CNN) також застосовують у *NLP* завдяки їхній здатності виділяти локальні патерни в тексті. Їх часто використовують для класифікації тексту та пошуку ключових фраз. Попри високу ефективність і швидкість роботи, *CNN* обмежені у здатності моделювати довготривалі залежності, що звужує їх застосування до локальних завдань.

Seq2Seq (Sequence-to-Sequence)-моделі ґрунтуються на ідеї перетворення однієї послідовності на іншу, що робить їх придатними для таких завдань як машинний переклад і підсумовування текстів. Вони демонструють гнучкість у роботі з послідовностями довільної довжини, але можуть стикатися з проблемами у моделюванні довготривалих залежностей, особливо без механізму уваги.

Революційний прорив у *NLP* забезпечили ***Transformers,*** які використовують ***Механізм уваги (Attention Mechanism)*** для врахування глобального контексту тексту. Вони можуть обробляти великі обсяги даних паралельно, що робить їх незамінними для сучасних завдань, включно з перекладом, класифікацією тексту та відповідями на запитання. Їхній успіх забезпечується механізмом уваги, який став ключовим компонентом моделей *BERT* і *GPT.*

Модель BERT (Bidirectional Encoder Representations from Transformers) розроблено для вирішення завдань класифікації тексту, пошуку інформації та відповіді на запитання. Її головна перевага — двоспрямований контекст, що дає змогу враховувати як попередній, так і наступний контекст у тексті. Високі результати *BERT* у завданнях *NLP* забезпечили цій моделі провідне місце серед сучасних моделей, хоча вона потребує досить великих обчислень.

Модель GPT (Generative Pre-trained Transformer) зосереджена на генерації тексту, демонструючи потужні можливості створення зв'язного і контекстуально точного тексту. Її застосовують для створення чатботів, автоматичного написання текстів і доповнення контенту. Однак нерівномірна увага до деталей контексту може бути певним обмеженням цієї архітектури.

Механізм уваги (***Attention Mechanism),*** який лежить в основі ***Transformers,*** є універсальним інструментом, що успішно моделює глобальні залежності в тексті. Цей механізм забезпечує високу якість у завданнях перекладу та узагальнення тексту, хоча його обчислювальна складність для довгих послідовностей може бути викликом.

Отже, різноманітність методів глибокого навчання (див. табл. 15.6) [15.16—15.25] забезпечує широкий вибір інструментів для вирішення завдань *NLP.* Від моделей, що фокусуються на роботі з послідовностями, до трансформерів, які революціонізували цю галузь, кожний метод має свої сильні сторони, обмеження та ідеальну сферу застосування. Це дає можливість створювати гнучкі й ефективні системи для аналізу, розуміння та генерації тексту.

15.3. Методи глибокого навчання з підкріпленням

Представлені вище чотири групи методів глибокого навчання в обробці природної мови мають широкий спектр застосувань і формують основу сучасних рішень у цій галузі. Окремо від цих груп стоїть метод глибокого навчання з підкріпленням (*Deep Reinforcement Learning, DRL*), який, хоча і менш поширений для вирішення завдань *NLP*, усе ж посідає важливе місце завдяки своїм особливим можливостям. Він є додатковим, дещо особливим підходом, що доповнює можливості основних методів.

Глибоке навчання з підкріпленням (*Reinforcement Learning, RL*) є підходом до навчання агентів через взаємодію із середовищем для досягнення визначеної мети [15.26]. У цьому процесі агент отримує зворотний зв'язок у вигляді винагороди за правильні дії або покарання за неправильні. Основною метою агента є максимізація отриманої винагороди протягом часу. *RL* ґрунтується на фундаментальних компонентах, як-от агент, середовище, стани, дії, винагороди, політика і функція цінності. Агент обирає дії, базуючись на своїй політиці, середовище реагує, змінюючи свій стан і надаючи винагороду, а агент оновлює свою стратегію на основі цієї інформації.

Поєднання* RL *із глибокими нейронними мережами дає змогу розвивати глибоке навчання з підкріпленням *(Deep Reinforcement Learning, DRL)*. Це злиття дає можливість вирішувати складні завдання, де традиційні методи *RL* неефективні через високий рівень складності середовища або великий простір можливих станів. *DRL* використовує нейронні мережі для апроксимації функцій цінності чи політики, що дає змогу обробляти дані великого масштабу, наприклад зображення або звукові сигнали. Основні алгоритми *DRL — Deep Q-Learning (DQN), Policy Gradient Methods ma Actor-Critic,* забезпечують ефективне навчання агента за різними сценаріями.

DRL застосовують у багатьох сферах. В іграх, наприклад, алгоритми *DRL* допомагають агентам досягати рівня професіоналів (як у випадку *AlphaGo*). У робототехніці цей підхід оптимізує рухи роботів у складних і динамічних середовищах. Автономні системи, як-от автопілоти для автомобілів, також активно використовують *DRL* для навчання прийняттю послідовних рішень у реальному часі. Крім того, *DRL* застосовують у завданнях управління ресурсами, наприклад, в оптимізації енергоспоживання або керуванні хмарними обчисленнями.

Попри високу ефективність, *DRL* має і свої надоліки. Його навчання є обчислювально інтенсивним і потребує великих обсягів даних, що значно підвищує вимоги до апаратного забезпечення. Додатково, поведінка агентів у *DRL* може бути нестабільною через високу складність середовищ, з

якими вони працюють. Однак навіть з урахуванням цих недоліків, *DRL* демонструє вражаючі результати в умовах невизначеності, де потрібно приймати складні послідовні рішення. Завдяки постійному розвитку методів і алгоритмів глибоке навчання з підкріпленням продовжує залишатися однією з ключових технологій сучасного машинного навчання.

Приклад. Застосування глибокого навчання з підкріпленням на прикладі алгоритму *Deep Q-Learning* для комп'ютерної гри *Atari Breakout*. Алгоритм ***Deep Q-Learning (DQN)*** був розроблений компанією *DeepMind* і продемонстрований на класичних ком'ютерних іграх *Atari,* зокрема *Breakout* [15.27]. Мета гри полягає в тому, щоб керувати платформою, яка відбиває м'яч, руйнуючи блоки у верхній частині екрана. Завдання агента полягає у максимізації кількості очок, набраних у грі.

Кількісні параметри

1. ***Стан (State):***

• вхідні дані — це кадри гри (зображення 84 × 84 пікселів у відтінках сірого);

• агент отримує послідовність із чотирьох останніх кадрів для врахування динаміки гри.

2. ***Дії (Actions):***

• можливі дії — пересунути платформу ліворуч, праворуч або залишити її на місці;

• кількість дій — три.

3. ***Винагорода (Reward):***

• «+1» за успішне руйнування блока;

• «0» за відсутність взаємодії з блоками;

• «−1» за втрату м'яча.

4. ***Алгоритм:***

• використовується нейронна мережа з трьома згортковими шарами (*Convolutional Layers*) для оброблення зображень і двома щільними шарами (*Fully Connected Layers*) для передбачення функції Q;

• функція Q оцінює довгострокову вигоду від кожної можливої дії.

5. ***Навчання:***

• підхід — метод *Replay Buffer* для збереження досвіду агента та вибірки мінібатчів для навчання;

• використовуються алгоритми оптимізації — *RMSProp* або *Adam;*

• кількість епох — 200 000 епізодів гри.

6. ***Результати:***

• **Початковий рівень агента —** випадковий рух платформи, середня кількість балів за гру — близько 5;

• **рівень після навчання —** агент демонструє стратегічну гру, руйнуючи максимальну кількість блоків, середня кількість балів за гру — понад 400.

Код програми на Python для гри Atari Breakout

```
# Встановіть необхідні бібліотеки, якщо вони ще не встановлені
# pipinstallstable-baselines3[extra] gym[atari]

importgym
fromstable_baselines3importDQN
fromstable_baselines3.common.evaluationimportevaluate_policy

# Завантаження середовища AtariBreakout
env = gym.make(«Breakout-v0»)

# Ініціалізація моделі DQN
model = DQN(
    «CnnPolicy»,            # Політика для роботи із зображеннями
env,                        # Середовище гри
learning_rate=0.0001,       # Швидкість навчання
buffer_size=50000,          # Розмір ReplayBuffer
learning_starts=1000,       # Кількість початкових кроків для збору даних
batch_size=32,              # Розмір міні-батчів
gamma=0.99,                 # Дисконтуючий фактор
target_update_interval=500,         # Оновлення TargetNetwork
verbose=1,                  # Рівень виводу інформації
)

# Навчання моделі
print(«Навчання моделі...»)
model.learn(total_timesteps=100000)    # Кількість кроків навчання

# Оцінка ефективності моделі
print(«Оцінка ефективності...»)
mean_reward, std_reward = evaluate_policy(model, env, n_eval_episodes=10)
print(f»Середня винагорода: {mean_reward}, Стандартне відхилення:
{std_reward}»)

# Збереження моделі
model.save(«dqn_breakout_model»)

# Демонстрація гри
print(«Демонстрація гри...»)
obs = env.reset()
for _ inrange(5000):
action, _states = model.predict(obs, deterministic=True)
obs, reward, done, info = env.step(action)
env.render()
```

```
ifdone:
obs = env.reset()

env.close()
```

Пояснення. Алгоритм *DQN* використовує нейронну мережу для апроксимації функції Q, яка оцінює важливість кожної дії в поточному стані. Агент навчається через експерименти у середовищі: він виконує дії, отримує винагороду й оновлює свою стратегію. Завдяки *Replay Buffer* алгоритм зберігає досвід взаємодій, що дає змогу зменшити кореляцію між послідовними станами і покращити стабільність навчання. Також методику зменшення ε (*epsilon-greedy*) використовують для балансування між дослідженням (*exploration*) і використанням накопиченого досвіду (*exploitation*).

Висновок. Алгоритм *Deep Q-Learning* показав, як глибоке навчання з підкріпленням може забезпечити високу продуктивність навіть у складних середовищах, як-от комп'ютерні ігри *Atari.* У прикладі з *Breakout* агент навчився грати краще за середньостатистичну людину, демонструючи здатність формувати оптимальні стратегії. Цей підхід може бути адаптований для широкого спектра завдань, від робототехніки до оптимізації в реальних системах, де потрібно приймати складні послідовні рішення.

Інтеграція методів глибокого навчання для вирішення комплексних завдань *NLP*

Інтеграція методів глибокого навчання є ключовою стратегією у вирішенні комплексних завдань обробки природної мови *(NLP).* Комплексні завдання *NLP* — машинний переклад, генерація тексту, класифікація, аналіз настроїв та отримання відповідей на запитання, потребують поєднання різних моделей і підходів, кожен із яких має певні унікальні переваги. Розглянемо концептуальні аспекти інтеграції на прикладі методів *RNN, Autoencoders, LSTM, GRU, CNN, Seq2Seq, Transformers, BERT, GPT* та *Attention Mechanism* (табл. 15.6).

Методи послідовної обробки (RNN, LSTM і GRU) надають потужний інструмент для оброблення послідовностей тексту. Вони забезпечують можливість моделювання залежностей між словами, враховуючи контекст попередніх елементів. Однак обмеження цих методів у роботі з довготривалими залежностями та висока обчислювальна складність створюють необхідність їх інтеграції з іншими підходами. Наприклад, комбінація *RNN* з *Attention Mechanism* дає можливість зосереджуватись на релевантних частинах тексту, що значно покращує результати у завданнях машинного перекладу.

Autoencoders забезпечують ефективне зниження розмірності та створення латентних подань тексту, що можуть бути інтегровані з іншими

моделями, як-от *LSTM* або *Transformers.* Наприклад, автоенкодери можна використовувати для попереднього оброблення тексту перед подальшим його аналізом із застосуванням складніших моделей. Це дає змогу знизити обчислювальні витрати і забезпечити якісну роботу зі стиснутими даними.

Згорткові нейронні мережі (CNN), завдяки своїй здатності виділяти локальні патерни, ефективно працюють у завданнях класифікації тексту або пошуку ключових фраз. Їх можна інтегрувати з *LSTM* або *GRU*, створюючи гібридні моделі, що одночасно враховують як локальні, так і глобальні залежності у тексті. Це особливо корисно для аналізу довгих текстів, де важливо зберегти баланс між локальною та глобальною інформацією.

Seq2Seq-моделі традиційно використовуються для перетворення однієї послідовності на іншу, наприклад, у завданнях машинного перекладу. Інтеграція *Attention Mechanism* з *Seq2Seq* значно підвищує їхню ефективність, даючи змогу моделям враховувати важливість кожного елемента послідовності під час декодування.

Transformers та їхні похідні — *BERT* і *GPT*, забезпечили революцію в *NLP* завдяки механізму уваги, що дає можливість обробляти текст глобально, а не послідовно. Інтеграція *Transformers* з іншими методами, наприклад автоенкодерами для створення попередньо навчених ембедингів або з *CNN* для виявлення локальних особливостей тексту, сприяє створенню потужних і універсальних систем. *GPT*, наприклад, може генерувати текст на основі інформації, попередньо отриманої через механізми уваги, інтегровані в його трансформерну архітектуру.

BERT завдяки двоспрямованій природі використовують для створення контекстно-залежних ембедингів тексту, які можна передати до інших моделей для подальшого оброблення. Наприклад, *BERT* можна інтегрувати в системи *Seq2Seq* для поліпшення розуміння тексту перед виконанням перекладу або узагальнення.

Attention Mechanism є ключовим компонентом сучасних підходів — *Transformers, BERT і GPT.* Його інтеграція з іншими архітектурами, наприклад *RNN* або *Seq2Seq*, допомагає створювати системи, що успішно враховують як локальний, так і глобальний контекст. Це особливо корисно для вирішення складних завдань, де потрібно працювати з довгими послідовностями, зберігаючи їхню семантичну цілісність.

Інтеграція методу глибокого навчання з підкріпленням (DRL) з провідними моделями глибокого навчання — *RNN, Autoencoders, LSTM, GRU, CNN, Seq2Seq, Transformers, BERT, GPT* і *Attention Mechanism,* є доцільним у тому разі, якщо *DRL* додає унікальну властивість — здатність адаптивно приймати рішення на основі взаємодії із середовищем, що відкриває нові можливості для вдосконалення традиційних методів. Так, *RNN, LSTM* і *GRU,* відомі своєю здатністю моделювати часові залежності у текстових

даних, стають ще ефективнішими, коли до них додається *DRL.* Наприклад, у діалогових системах *DRL* допомагає агенту оптимізувати відповіді з урахуванням довгострокових взаємодій із користувачем. Такий підхід дає змогу враховувати не лише поточний контекст, а й вплив кожної репліки на подальшу бесіду. У випадку *Autoencoders i*нтеграція з *DRL* стає корисною для завдань, де важлива адаптація до змін у даних, наприклад, для адаптивної реконструкції або очищення шуму. *DRL* дає агенту можливість обирати найкращу стратегію для зменшення шуму, оптимізуючи процес реконструкції залежно від контексту.

З модифікацією *CNN* інтеграція *DRL* є доцільною в завданнях аналізу тексту або зображень у динамічному середовищі. Наприклад, у системах, що працюють із потоковими даними або інтерактивними сценаріями, *CNN* забезпечує якісний аналіз локальних особливостей, тоді як *DRL* допомагає агенту адаптувати свої дії відповідно до змінних умов. *Seq2Seq*-моделі, які широко використовують у завданнях машинного перекладу та автоматичного резюмування, також отримують допомогу від *DRL.* Додавання механізму підкріплення дає можливість оптимізувати переклад або структуру резюме, орієнтуючись на цільові критерії, наприклад точність чи стиль.

Особливий інтерес становить інтеграція *DRL* із моделями *Transformers*, як-от *BERT*, *GPT*, і механізмами уваги. Ці моделі вже демонструють виняткову здатність моделювати контекст і залежності у текстах, але *DRL* дає можливість удосконалювати їх через адаптивне налаштування. Зокрема, *DRL* використовують для навчання моделей *GPT* за допомогою *RLHF (Reinforcement Learning with Human Feedback)*, що забезпечує оптимальну релевантність відповідей і кращу взаємодію з користувачами. У пошукових системах *DRL* допомагає налаштувати результати пошуку залежно від зворотного зв'язку користувачів, а також оптимізувати адаптацію моделей до змінних умов середовища.

Генеративні моделі — *GPT* і *Autoencoders,* набувають значних переваг завдяки *DRL* у завданнях генерації тексту. Так, *DRL* може забезпечити створення текстів, які відповідають конкретним стилям чи контекстам, наприклад, у творчих додатках чи чатботах. У поєднанні з традиційними методами *DRL* може вирішувати завдання довгострокового планування — оптимізацію пошукових запитів чи рекомендаційних систем. У цих сценаріях *DRL* додає можливість враховувати вплив поточних рішень на остаточний результат, що підвищує ефективність моделі.

Отже, інтеграція методів глибокого навчання дає можливість ефективно комбінувати їхні сильні сторони, вирішуючи комплексні завдання *NLP.* Ці інтегровані моделі можуть забезпечити як локальний, так і глобальний аналіз тексту, враховувати контекст різного рівня і забезпечувати високу продуктивність у завданнях машинного перекладу, класифікації, генерації

тексту та відповіді на запитання. Це підхід, який відкриває нові горизонти для розвитку обробки природної мови.

Приклад. Сучасним прикладом інтеграції методів глибокого навчання для вирішення комплексних завдань *NLP* є поєднання великих мовних моделей (*Large Language Models, LLM*), як-от *GPT*, з методами навчання з підкріпленням (*Reinforcement Learning, RL*) для створення діалогових систем, здатних генерувати природні та релевантні відповіді. *RL* — це підхід до машинного навчання, за якого агент навчається взаємодіяти з середовищем, виконуючи дії і отримуючи винагороди за кожну з них. Мета — максимізувати сумарну винагороду, яка залежить від виконання певного завдання.

У листопаді 2022 р. дослідники *OpenAI* запропонували модель *ChatGPT*, що поєднує *GPT-3.5* з навчанням з підкріпленням від людської оцінки (*Reinforcement Learning with Human Feedback, RLHF*). *RLHF* — це вдосконалена форма навчання з підкріпленням, у якій оцінка людини використовується для вдосконалення моделі [15.28]. Люди надають зворотний зв'язок, указуючи на бажану поведінку моделі або якість її відповідей. Завдяки застосуванню такого підходу модель може не лише генерувати текст на основі статистичних закономірностей, а й адаптувати свої відповіді з урахуванням уподобань та етичних норм людини. Кількісні результати показали значне поліпшення якості відповідей. Зокрема, за оцінками користувачів, *ChatGPT* перевершив попередні моделі *GPT-3* за такими показниками: релевантність, зв'язність та природність відповідей. Якісно модель продемонструвала здатність вести більш осмислені та контекстуально відповідні діалоги, що робить її корисною в різних застосуваннях, від підтримки клієнтів до освітніх платформ.

Висновок. Наведений приклад свідчить, що інтеграція методів глибокого навчання під час вирішення комплексних завдань *NLP* має суттєві переваги: підвищення точності, гнучкість та універсальність моделей. Це дає можливість ефективно поєднувати локальні та глобальні підходи до аналізу тексту, адаптувати системи до специфіки завдання та забезпечувати їх застосування в широкому спектрі завдань. Саме тому сучасні NLP-технології ґрунтуються на синтезі різних методів глибокого навчання, що уможливлює вирішення завдань, які ще кілька років тому здавалися недосяжними.

Висновки

У сучасних системах обробки природної мови (*NLP*) ключову роль відіграють два підходи: традиційні методи машинного навчання та моделі глибокого навчання. Докладно розглянуто їхні особливості, функціональні характеристики та сфери застосування. Зокрема, методи глибокого навчання було класифіковано за архітектурними та функціональними підхо-

дами на чотири основні класи: послідовні моделі, локальні моделі, моделі з механізмом уваги та генеративні моделі. Такий підхід дав можливість інтегрувати технічні аспекти роботи моделей із функціональними характеристиками, створюючи зручну основу для розуміння ролі й місця цих моделей у сучасному *NLP.*

Однією з основних переваг глибокого навчання є його здатність обробляти складні патерни та великі обсяги даних, що робить його незамінним у вирішенні комплексних завдань *NLP.* Від моделей, що працюють із послідовностями, до трансформерів, які революціонізували галузь, кожен підхід має свої сильні сторони, обмеження та ідеальні сфери застосування. Наприклад, рекурентні нейронні мережі (*RNN*), автоенкодери, довготривала короткострокова пам'ять (*LSTM*) та згорткові нейронні мережі (*CNN*) добре придатні для аналізу структурованих даних, натомість моделі з механізмом уваги, трансформери, *BERT* і *GPT* забезпечують гнучкість у роботі з неструктурованими текстовими даними.

Інтеграція методів глибокого навчання є ключовою стратегією для вирішення складних завдань *NLP* — машинного перекладу, генерації тексту, класифікації, аналізу настроїв та отримання відповідей на запитання. Наприклад, *Seq2Seq*-моделі та моделі з механізмом уваги допомагають ефективно обробляти послідовності даних, тоді як трансформери забезпечують високоточну генерацію тексту та аналіз змісту. Завдяки цьому стає можливим поєднання локальних і глобальних підходів до аналізу тексту, адаптація систем до специфіки завдань та їх широке застосування у різних сферах.

Інтеграція методу глибокого навчання з підкріпленням (*Deep Reinforcement Learning, DRL*) із провідними моделями глибокого навчання як-от *RNN, Autoencoders, LSTM, GRU, CNN, Seq2Seq, Transformers, BERT, GPT* і *Attention Mechanism,* є перспективним напрямом для вирішення складних завдань у галузі обробки природної мови. *DRL* додає унікальну властивість — здатність адаптивно приймати рішення на основі взаємодії із середовищем, що відкриває нові можливості для вдосконалення традиційних методів.

Отже, важливою перевагою сучасних *NLP*-технологій є здатність комбінувати різні методи глибокого навчання. Синтез методів глибокого навчання не лише відповідає потребам сучасного *NLP*, але й прокладає шлях до подальших інновацій у цій сфері.

СПИСОК ЛІТЕРАТУРИ

15.1 *Deeplearning.ai. Natural Language Processing.* 2023. URL: https://www.deeplearning.ai/resources/natural-language-processing/#What_Is_Natural_Language_Processing_(NLP)_Used_for

15.2 Manning C. D., Raghavan P., Schütze H. Introduction to Information Retrieval. Cambridge University Press. 2008. 506 p. URL: https://nlp.stanford.edu/IR-book/information-retrieval-book.html

15.3 Bird S., Klein E., Loper E. Natural Language Processing with Python. O'Reilly Media. 2009. 504 p. URL: https://www.researchgate.net/publication/220691633_Natural_Language_Processing_with_Python

15.4 Goldberg Y. Neural Network Methods for Natural Language Processing. Morgan & Claypool Publishers. 2009. 310 p. URL: https://link.springer.com/book/10.1007/978-3-031-02165-7

15.5 Alpaydin E. Introduction to Machine Learning (14 th edition). MIT Press. 2020. 712 p. URL: https://mitpress.mit.edu/9780262043793/introduction-to-machine-learning/

15.6 Bishop C. M. Pattern Recognition and Machine Learning/ Springer. 2006. 738 p. URL: https://www.cs.uoi.gr/~arly/courses/ml/tmp/Bishop_book.pdf

15.7. Pedregosa F., Varoquaux G., Gramfort A., Michel V., Thirion B., Grisel O. et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*/ 2011. No. 12. P. 2825—2830. URL: http://jmlr.org/papers/v12/pedregosa11a.html

15.8 Scikit Learn. 1.9. Naive Bayes. 2024. URL: https://scikit-learn.org/stable/modules/naive_bayes.html

15.9 Bug Squasher: AI helps security researchers fix the most severe bugs first. 2020. URL: https://www.deeplearning.ai/the-batch/bug-squasher/

15.10 Huang Z., Xu W., Yu K. Bidirectional LSTM-CRF Models for Sequence Tagging, 2015. URL: https://arxiv.org/abs/1508.01991

15.11 Arya A. Text Classification Using Decision Tree. Kaggle. URL: https://www.kaggle.com/code/aryaadithyan/text-classification-using-decision-tree

15.12 Blei D. M., Ng A. Y., Jordan M. I. Latent Dirichlet Allocation. *Journal of Machine Learning Research*. 2003. Vol. 3. P. 993—1022. URL: https://dl.acm.org/doi/10.5555/944919.944937

15.13 Zucchini W., MacDonald I. L. Hidden Markov Models for Time Series. An Introduction Using R. 2009. 288 p. URL: https://www.taylorfrancis.com/books/mono/10.1201/9781420010893/hidden-markov-models-time-series-walter-zucchini-iain-macdonald

15.14 LeCun Y., Bengio Y., Hinton G. Deep learning. *Nature*. 2015. 521(7553). P. 436—444. URL: https://www.nature.com/articles/nature14539

15.15. Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A.N. et al. Attention is All You Need. *Advances in Neural Information Processing Systems*. 2017. Vol. 30. URL: https://papers.nips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

15.16 Kowsari K., Meimandi K. J., Heidarysafa M., Mendu S., Barnes L. E., Brown D. E. Text classification algorithms: A survey. *Information*. 2019. 10(4), 150. URL: https://arxiv.org/abs/1904.08067

15.17 Hochreiter S., Schmidhuber J. Long Short-Term Memory. *Neural Computation*. 1997. No. 9(8). P. 1735—1780. URL: https://dl.acm.org/doi/10.1162/neco.1997.9.8.1735

15.18 Cho K., Van Merriënboer B., Gulcehre C., Bahdanau D., Bougares F., Schwenk, H., et al. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2014. URL: https://arxiv.org/abs/1406.1078

15.19 Devlin J., Chang M. W., Lee K., Toutanova K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*. 2018. URL: https://aclanthology.org/N19-1423/

15.20 Liu Y., Ott M., Goyal N., Du J., Joshi M., Chen D. et al. *RoBERTa:* A Robustly Optimized BERT Pretraining Approach. 2019. URL: https://arxiv.org/abs/1907.11692

15.21 Radford A., Narasimhan K., Salimans T., Sutskever I. Improving Language Understanding by Generative Pre-Training. *OpenAI.* 2018. URL: https://www.semanticscholar.org/paper/Improving-Language-Understanding-by-Generative-Radford-Narasimhan/cd18800a0fe0b668a1cc19f2ec95b5003d0a5035

15.22 Le Q. V., Mikolov T. Distributed representations of sentences and documents. *International Conference on Machine Learning. PMLR*. 2014. URL: https://arxiv.org/abs/1405.4053

15.23 Peters M. E., Neumann M., Iyyer M., Gardner M., Clark C., Lee K. et al. Deep contextualized word representations. *In Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT).* 2018. URL: https://aclanthology.org/N18-1202/

15.24 Howard J., Ruder S. Universal Language Model Fine-tuning for Text Classification. *In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL).* 2018. URL: https://aclanthology.org/P18-1031/

15.25 Mikolov T., Chen K., Corrado G., Dean J. Efficient Estimation of Word Representations in Vector Space. 2013. arXiv preprint:1301.3781. URL: https://arxiv.org/abs/1301.3781

15.26. Clark E. Reinforcement Learning: Whatisit, Algorithms, Typesand Examples. 2024. URL: https://www.guru99.com/uk/reinforcement-learning-tutorial.html

15.27. Chapman J., Lechner M. *Keras.io.* Deep Q-Learning for Atari Breakout. 2024. URL: https://keras.io/examples/rl/deep_q_network_breakout/?utm_source=chatgpt.com

15.28. *OpenAI*. ChatGPT: Optimizing Language Models for Dialogue. November 30 2022. URL: https://openai.com/blog/chatgpt/

РОЗДІЛ 16

ДЕЯКІ ПРАКТИЧНІ АСПЕКТИ ЗАСТОСУВАННЯ СИСТЕМ ОБРОБКИ ПРИРОДНОЇ МОВИ

16.1. Еволюція систем обробки природної мови у період 1964—2024 рр.

16.2. Найпоширеніші класи мовних моделей

16.3. Нове покоління *NLP*-моделей, придатних для вирішення складних і ресурсомістких завдань

16.4. Апаратне і програмне забезпечення машинного навчання мовних моделей

16.5. Деякі суперечки і дискусії навколо проблеми обробки природної мови

Висновки

Список літератури

Системи обробки природної мови *(NLP)* є конкретними реалізаціями методів і моделей, навчених на певних наборах даних для виконання різноманітних завдань. Вони втілюють алгоритми в конкретні архітектури, адаптуючи їх до практичних потреб. Роль *NLP*-систем у розвитку штучного інтелекту (ШІ) є надзвичайно важливою: від початкових алгоритмів, що ґрунтувалися на жорстких правилах, до сучасних глибоких нейронних мереж, які забезпечують високоточну обробку та генерацію тексту з урахуванням контексту.

Еволюція цих систем нерозривно пов'язана з прогресом штучного інтелекту загалом: від ручного створення правил і шаблонів до використання підходів глибокого навчання, що дають можливість моделювати складні мовні закономірності на основі великих текстових корпусів. Сучасні *NLP*-технології стали ключовим інструментом для вирішення таких завдань, як машинний переклад, розпізнавання настроїв, класифікація тексту та автоматичне написання текстів, забезпечуючи їх ефективність і універсальність у широкому спектрі застосувань.

16.1. Еволюція систем обробки природної мови у період 1964—2024 рр.

Еволюція систем обробки природної мови (*NLP*) умовно поділяється на чотири етапи, кожен із яких зробив вагомий внесок у розвиток технологій ШІ.

Перший етап (1960—1970-ті рр.) ознаменував початкові дослідження, спрямовані на розроблення концептуальних засад ШІ. Перші мовні моделі — *Eliza* [16.1] і *SHRDLU (A Model for Understanding and Generating Language)* [16.2] заклали основу для розвитку *NLP. Eliza*, розроблена Джозефом Вайценбаумом (Joseph Weizenbaum), імітувала інтелектуальну взаємодію людини з машиною за допомогою використання шаблонних відповідей, створюючи ілюзію розуміння тексту. Ця модель стала важливим кроком у розробленні чатботів. *SHRDLU*, створена Террі Віноградом (Terry Winograd), інтегрувала обробку природної мови, логіку та планування дій у «світі блоків», демонструючи розуміння складніших мовних конструкцій [16.2]. Хоча ці моделі мали вузький контекст застосування, вони стали прототипами для майбутніх інтерактивних систем.

Другий етап (1980-ті—2000-ні рр.) характеризується впровадженням у *NLP* статистичних підходів. Моделі цього періоду, як-то *TF-IDF (Term*

Frequency-Inverse Document Frequency) [16.3], *LSA (Latent Semantic Analysis)* [16.4, 16.5] і *LDA (Latent Dirichlet Allocation)* [16.6], дали змогу аналізувати великі текстові корпуси та видобувати значущу інформацію на основі статистичних закономірностей. *TF-IDF* забезпечила оцінку важливості термінів у документах, *LSA* виявила приховані семантичні зв'язки через сингулярне розкладання матриць, а *LDA* запровадила тематичне моделювання, де документи зображались як суміші тем. Ці підходи заклали фундамент для масштабованих і гнучких систем обробки текстів, що стало можливим завдяки зростанню обчислювальних потужностей.

Третій етап (2010-ті рр.) ознаменував перехід до глибокого навчання та семантичного зображення природної мови. Моделі *Word2Vec (Word to Vector)* [16.7] і *GloVe (Global Vectors for Word Representation)* [16.8] стали революційними, створюючи векторні зображення слів у багатовимірному просторі. *Word2Vec*, розроблена командою *Google,* використовувала архітектури *CBOW (Continuous Bag of Words)* і *Skip-gram* для визначення контекстуальних зв'язків між словами, тоді як *GloVe* запропонувала аналіз матриць спільної поширеності для врахування глобальних залежностей. Ці моделі стали основою для складніших систем, як-от трансформери.

Четвертий етап (2017 — по сьогодні) пов'язаний із революцією трансформерів. Механізм самоуваги *(self-attention)* дав можливість трансформерам, як-от *BERT* (*Bidirectional Encoder Representations from Transformers*) [16.9], *GPT (Generative Pre-Trained Transformer)* [16.10], *T5* [16.11], *Claude (Claude Shannon)* [16.12], *LLaMA (Large Language Model Meta AI)* [16.13], *RoBERTa (Robustly Optimized BERT Approach)* [16.14], OPT *(Optimized Runtime for Transformers)* [16.15] досягти значного прогресу в розумінні тексту. Завдяки паралельному обробленню тексту трансформери стали основою сучасних *NLP*-систем, демонструючи високі результати в генерації тексту, машинному перекладі та аналізі настроїв. Ці моделі також знайшли застосування в мультимодальних системах, що працюють із текстом, зображеннями та аудіо.

Моделі, побудовані на трансформерах, не лише підвищили ефективність обробки природної мови, а й заклали основу для нових технологій, що наблизили *NLP* до рівня людського розуміння. Це відображає еволюцію ШІ від правил і шаблонів до глибокого навчання та інтеграції з іншими типами даних.

Група найновіших мовних моделей, створених у 2023—2024 рр.: *Gemini 1.5* (2024 р.) [16.16]; *Claude 3* (2024) [16.17]; *LLaMA* 3 (2024) [16.18]; *Mistral* 7B/12B (2023) [16.19]; *GPT-4* (2023) [16.20] революціонізували *NLP.*

Відомі моделі обробки природної мови *(NLP)*, створені у період 1960—2024 рр., наведено в табл. 16.1. Ці моделі стали ключовими віхами у розвитку ШІ, від простих програм, що імітують діалог людини з машиною, до сучасних глибоких моделей, які можуть виконувати широкий спектр завдань природного спілкування людини з машиною.

Таблиця 16.1

Широко застосовувані моделі обробки природної мови (*NLP*) у період 1964—2024 рр.

Модель	Рік створення	Концептуальні засади	Кількість параметрів	Контекстне вікно (токени)	Продуктивність: точність, % F1-міра, % перплексія, ум. од.
ELIZA	1966	Проста модель, що імітує роботу психотерапевта, використовуючи зіставлення зі зразком для оброблення введених користувачем текстів	Не застосовується	Немає	Не застосовується
SHRDLU	1968—1970	Модель для взаємодії з віртуальним середовищем, що розуміє природну мову в певних рамках	Не застосовується	Немає	Не застосовується
TF-IDF	1972	Метод визначення важливості слів на основі їх частоти	Не застосовується	Немає	Не застосовується
LSA	1988	Модель виявлення прихованих смислових відносин між словами	Не застосовується	Немає	Не застосовується
LDA	2003	Генеративна статистична модель для виявлення прихованих тем у текстах	220 тис. — 50 млн	Немає	Не застосовується
Word2Vec	2013	Векторне зображення слів з урахуванням семантичних зв'язків	6 млн	Немає	Не застосовується
GloVe	2014	Векторні зображення слів із використанням глобальної статистики тексту	100 млн	Немає	Не застосовується
Transformer	2017	Архітектура з механізмом уваги для ефективного оброблення послідовностей	65 млн	~512	Не застосовується
BERT	2018	Модель із двоспрямованим розумінням контексту	110 млн	~512	~85 % ~82 % ~30

Закінчення табл. 16.1

Модель	Рік створення	Концептуальні засади	Кількість параметрів	Контекстне вікно (токени)	Продуктивність: точність, % F1-міра, % перплексія, ум. од.
GPT	2018	Авторегресивна модель для генерування текстів	110 млн	~2 048	~86 % ~83 % ~25
RoBERTa	2019	Покращена версія *BERT* із тривалішим навчанням	125—355 млн	~512	~88 % ~85 % ~20
GPT-2	2019	Покращена версія *GPT* із більшою кількістю параметрів	1,5 млрд	~2 048	~90 % ~88 % ~15
T5	2019	Універсальна текстова модель для уніфікації всіх завдань *NLP*	11 млрд	~1 024	~92 % ~89 % ~12
GPT-3	2020	Суттєво промасштабована модель із високим рівнем універсальності	175 млрд	~2 048	~93 % ~91 % ~10
OPT	2022	Відкрита мовна модель з акцентом на доступність	125 млн — 66 млрд	~2 048	~91 % ~89 % ~14
GPT-4	2023	Мультимодальна модель для оброблення тексту, зображень та аудіо	~1 трильйон	~32 000	~95 % ~93 % ~8
Claude 3	2023	Модель з акцентом на етичність та безпеку	~100 — 200 млрд	~200 000	~95 % ~94 % ~10

Claude 3.5	2024	Вдосконалена версія *Claude 3* із поліпшеними швидкістю та продуктивністю	~200 млрд	~200 000	~96 % ~95 % ~8
Gemini 1.5	2024	Багатомодальна модель із контекстним вікном до 2 млн токенів	~200 млрд	~2 000 000	~97 % ~96 % ~7
LLaMA 3	2024	Модель з довжиною контексту до 128 тис. токенів для багатомовної підтримки	~200—300 млрд	~128 000	~94 % ~92 % ~11
Mistral 7B/12B	2024	Компактні моделі з високою ефективністю для завдань багатомовного оброблення текстів	7—12 млрд	~16 000	~91 % ~89 % ~14

Примітки. 1. Кількість параметрів у моделі визначає її здатність вивчати складні патерни в даних та відображає її потужність.

2. Контекстне вікно *(токени)* визначає кількість токенів (словесних одиниць або символів), які модель може враховувати одночасно під час оброблення тексту.

3. Продуктивність моделей (умовні показники) оцінюється за такими основними метриками: точність *(accuracy)* — відсоток правильних передбачень моделі. Чим вищий цей показник, тим краще модель вирішує завдання; *F1-міра (F1-Score)* — гармонійне середнє між точністю і повнотою, що показує, наскільки модель добре працює в умовах, де важливе врахування як коректних відповідей, так і відсутності помилкових передбачень; перплексія *(perplexity)* — здатність моделі передбачати наступне слово в тексті. Нижче значення свідчить про краще передбачення.

Отже, три ключові показники — кількість параметрів, контекстне вікно і продуктивність — визначають можливості моделі та її застосування. Більша кількість параметрів і розширене контекстне вікно дають моделям можливість працювати з великими обсягами даних і складними завданнями, натомість високі показники точності, F1-міри та низька перплексія є показниками високого рівня розвитку сучасних мовних моделей.

Для оцінки ефективності мовних моделей вводять поняття кількісних характеристик (показників) цих моделей, які визначають їхні потужність, продуктивність, складність та здатність вирішувати різні завдання.

Основними кількісними характеристиками мовних моделей можуть бути такі.

1. ***Кількість параметрів (Number of Parameters)*** є основним показником розміру та складності моделі. Тобто параметри мовної моделі — це числові значення (ваги) в нейронній мережі, що визначають, як модель перетворює вхідні дані на вихідні. У процесі навчання ці параметри налаштовуються (оновлюються) через алгоритми оптимізації (наприклад, алгоритм градієнтного спуску), для того щоб модель могла навчитися правильно обробляти текст і виконувати завдання, такі як класифікація, переклад або генерування тексту. Кількість параметрів визначає, наскільки детально модель може захоплювати інформацію і вирішувати складні завдання. Наприклад, *GPT-3* має 175 млрд параметрів, що забезпечує виконання широкого кола завдань з високою точністю.

2. ***Розмір тренувального набору (Training Data Size)*** — це обсяг текстових даних, на яких було навчено модель. Чим більший і різноманітніший набір даних, тим більше інформації модель може вивчити, що поліпшує її здатність до генерування тексту та розуміння контексту.

3. ***Ширина шарів (Layer Width)*** — це кількість нейронів у кожному шарі нейронної мережі. Більша ширина дає змогу моделі краще захоплювати складні залежності в даних.

4. ***Глибина моделі (Model Depth)*** — це кількість шарів у нейронній мережі (зазвичай у трансформерах), визначає глибину моделі. Більш глибокі моделі здатні захоплювати складніші патерни та ієрархії в даних.

5. ***Контекстне вікно (Context Window, або Sequence Length)*** — максимальна кількість токенів (символів, слів або підслів), які модель може обробити одночасно. Чим довше контекстне вікно, тим більше контексту модель може врахувати під час оброблення або генерування тексту.

6. ***Час навчання (Training Time)*** — кількість часу, який було витрачено на навчання моделі. Час навчання залежить від обсягу даних, кількості параметрів та обчислювальних ресурсів.

7. ***Продуктивність на тестових наборах (Performance on Benchmark Datasets)*** — оцінка точності, F1-міри, перплексії або інших метрик на стандартних наборах даних, які використовують для оцінки якості моделі.

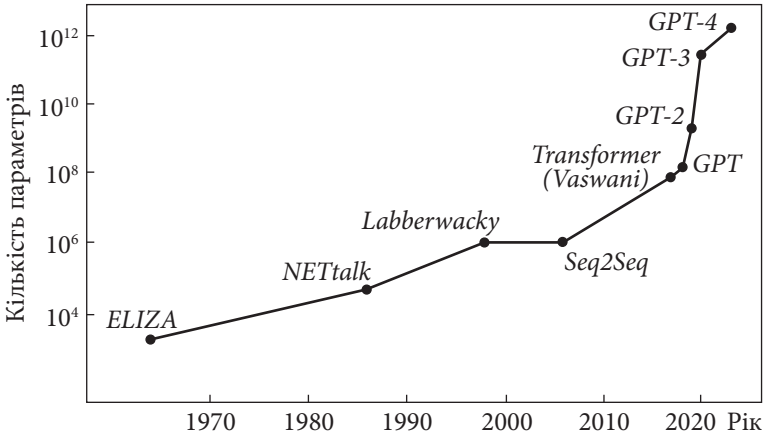


Рис. 16.1. Зростання кількості параметрів у найпопулярніших мовних моделях відрізку за період з 1964 по 2024 р.

Графік зростання кількості параметрів найпопулярніших мовних моделей у період з 1964 по 2024 рр. наведено на рис. 16.1. Цей графік ілюструє, як від моделей з невеликою кількістю параметрів, таких як *ELIZA* у 1964 р., людство дійшло до сучасних моделей — GPT-4, що мають трильйони параметрів. Графік демонструє величезний стрибок експоненціального характеру в можливостях і складності мовних моделей, який стався за останні десятиліття, і окреслює непередбачувані перспективи розвитку ШІ впродовж близького майбутнього.

Важливим показником мовних моделей є їхня продуктивність як узагальнена оцінка точності. *Продуктивність* мовних моделей має такі складові:

1) відсоток правильних передбачень моделі з усіх зроблених передбачень;

2) F1-міри (гармонійного середнього значення між точністю і повнотою передбачень);

3) перплексії (оцінки точності передбачення вибраного слова у реченні).

Еволюція продуктивності мовних моделей у період з 1964 по 2024 р. відбувалася в кілька етапів, де кожен етап відзначався істотним підвищенням можливостей обробки природної мови, зокрема продуктивністю в узагальнених умовних одиницях (табл. 16.2).

Графік зростання продуктивності мовних моделей у період 1964—2024 рр. наведено на рис. 16.2. Бачимо, що за останні 70 років їх продуктивність зросла на кілька порядків, що свідчить про стрімкий розвиток технологій обробки природної мови.

Еволюція мовних моделей за період 1964—2024 рр. відображає значний прогрес у підходах до обробки природної мови. Кожне десятиліття було позначене новими концептуальними проривами, які поступово покращували здатність машин розуміти й генерувати мову людини. Ця еволюція демон-

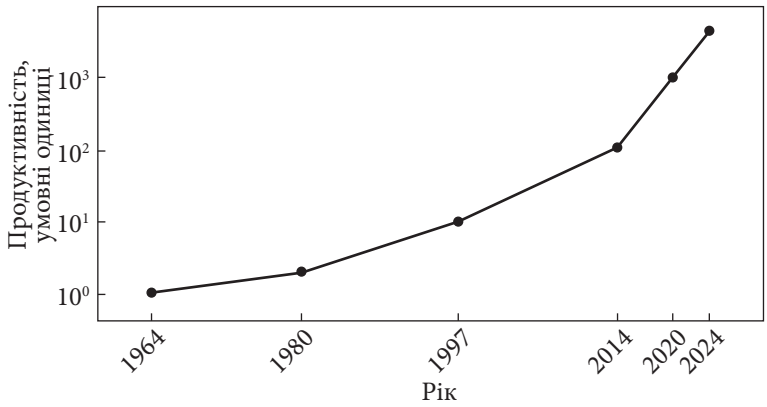


Рис. 16.2. Зростання продуктивності мовних моделей (умовні одиниці) за період 1964—2024 рр.

струє, як технології в *NLP* поступово розвивалися від простих правил до сучасних потужних нейромережевих архітектур, здатних розв'язувати широкий спектр задач з високою точністю. Ось основні етапи цього шляху:

1. Перехід від використання правил і шаблонних моделей до застосування нейронних мереж, що забезпечило здатність машин до навчання і генерування тексту.

2. Створення семантичного розуміння і векторних зображень, тобто розроблення моделей, здатних вловлювати семантичні зв'язки між словами і текстами.

3. Революція мовних моделей завдяки появі трансформерів. Впровадження трансформерів, таких, наприклад, як *Transformer, BERT, GPT* та ін., що забезпечило можливість вирішення складних завдань *NLP* з високою точністю.

Таблиця 16.2

Еволюція продуктивності мовних моделей за період з 1964 по 2024 р.

Номер пор.	Етап	Період (роки)	Продуктивність, ум. од.
1	Ранні роки	1960—1980-ті	1—5
2	Перехід до статистичних методів	1990-ті	5—20
3	Моделі на основі векторних зображень	2000-ні — середина 2010-х	20—100
4	Епоха нейронних мереж та трансформерів	2017—2020-ті	100—1 000
5	Сучасні великі мовні моделі	2020—2024	1 000—10 000

4. Масштабування і уніфікація мовних моделей. У 2020-х рр. моделі стали значно більшими і потужнішими, досягаючи неймовірних успіхів у розумінні та генеруванні тексту.

5. Зосередження на ефективності мовних моделей. Останні моделі фокусуються на оптимізації, зменшенні витрат ресурсів і підвищенні продуктивності на основі глибокого навчання.

6. Стрімкий розвиток мультимодальних та багатомовних моделей. Новітні розробки зосереджуються на мультимодальних моделях (що можуть обробляти текст, зображення, аудіодані тощо) та багатомовних моделях (що розуміють і генерують текст різними мовами).

7. Врахування етичних та соціальних викликів. Паралельно з технічним прогресом постали питання етики, прозорості та відповідальності в *NLP.* Стало важливо, щоб моделі були справедливими, неупередженими і не використовувалися зі шкідливою метою.

Цей еволюційний шлях ілюструє, як швидко розвиваються технології у сфері *NLP* і які можливості вони відкривають у найближчому майбутньому.

16.2. Найпоширеніші класи мовних моделей

Наведемо найбільш типові приклади різних класів мовних моделей, які розроблялися на різних етапах розвитку технологій і методів обробки природної мови.

Перший етап (1960-ті—1970-ті рр.). Мовні моделі, побудовані на правилах і шаблонах

1. **Мовна модель *ELIZA* (1964—1966).** Першу мовну модель для обробки природної мови, відому під назвою *ELIZA*, було розроблено в 1964—1966 рр. американським комп'ютерним науковцем Джозефом Вайценбаумом (Joseph Weizenbaum) у Массачусетському технологічному інституті (MIT) [16.1]. *ELIZA* була однією з перших програм, яка імітувала діалог людини з комп'ютером, використовуючи прості шаблони розпізнавання тексту. Програма могла «спілкуватися» з користувачами, повторюючи їхні твердження у формі запитань або коментарів.

Одним із найбільш цікавих і знакових застосувань моделі *ELIZA* була імітація поведінки психотерапевта у стилі видатного американського психолога Карла Роджерса [16.1]. Модель зосереджувалася на використанні клієнт-центрованого підходу Роджерса, відтворюючи ключові елементи його гуманістичної психотерапії. Вона програмно відтворювала ці принципи за допомогою алгоритмів, які реагували на висловлювання користувача. Програма використовувала шаблонні відповіді, щоб створити ілюзію терапевтичної взаємодії, де «психотерапевт» ніби уважно слухав і підтри-

мував розмову. Це підкреслювало емпатійний стиль Карла Роджерса, хоча насправді програма не мала жодного розуміння змісту висловлювань. Розглянемо приклад взаємодії з програмою *ELIZA* [16.1]:

Приклад:

• Користувач: «*Мене турбує моя робота*».

• ELIZA: «*Чому ви кажете, що вас турбує ваша робота?*»

• Користувач: «*Мені здається, що я не досягаю успіху.*»

• ELIZA: «*Чому вам здається, що ви не досягаєте успіху?*»

• Користувач: «*Мої колеги отримують підвищення, а я ні.*»

• ELIZA: «*Ви відчуваєте заздрість до своїх колег?*»

• Користувач: «*Можливо, так.*»

• ELIZA: «*Чому ви кажете, що, можливо, відчуваєте заздрість?*»

Кількісні параметри:

• *Обсяг пам'яті.* Модель працювала на комп'ютері IBM 7094 з 32K оперативної пам'яті.

• *Алгоритм.* Понад 200 шаблонів для оброблення тексту, зокрема ключових слів, наприклад «мама», «чому».

• *Реакція.* Час оброблення одного введення — менш як 1 секунда.

• *Типові показники взаємодії.* Кількість слів у введенні користувача — 5—15; кількість взаємодій у сеансі — ~20—30 реплік; середній час діалогу — 10—15 хвилин.

Результат. З цього діалогу видно, як модель підхоплює ключові фрази користувача і будує на їх основі подальші запитання. Це створює ілюзію уважного слухання та підтримки, що є центральним елементом клієнт-центрованої терапії Роджерса.Тобто модель *ELIZA* викликала у користувачів відчуття осмисленої розмови, хоча її відповіді ґрунтувалися на простих правилах. Вона продемонструвала, що навіть базові системи можуть бути корисними для психологічної підтримки.

Попри технічну простоту модель *ELIZA* стала важливою віхою в історії розвитку штучного інтелекту та обробки природної мови. Вона показала, що навіть базові алгоритми можуть створювати враження «глибокого діалогу», якщо їх поведінка правильно відтворює комунікативні принципи дюдини. І хоча модель не мала справжнього інтелекту, вона наочно демонструвала, як технології можуть імітувати психотерапевтичний підхід і створювати враження емпатії.

2. **Мовна модель *SHRDLU* (1968—1970).** Мовна модель *SHRDLU (A Model for Understanding and Generating Language)* — одна з перших мовних моделей, розроблена Террі Віноградом (Terry Winograd) у 1968—1970 рр. у Стенфордському університеті [16.2]. Модель *SHRDLU* могла розуміти команди природною мовою і керувати віртуальним середовищем (мікросвітом), яке умовно складалося з різних блоків, а користувач міг маніпулю-

вати цими блоками в обмеженому середовищі. Програма стала важливим етапом розвитку штучного інтелекту та обробки природної мови, оскільки в неї було закладено кілька нових, важливих для розвитку *NLP* та штучного інтелекту концептуальних принципів [16.2].

1. ***Семантичне розуміння контексту.*** *SHRDLU* працювала у спеціально визначеному мікросвіті, де об'єкти та їхні властивості було чітко визначено. Ця модель могла розуміти і реагувати на команди в контексті цього мікросвіту, що давало програмі можливість оперувати значеннями слів у певному контексті, а не лише на основі синтаксису.

2. ***Обробка природної мови з використанням структур даних.*** Програма використовувала структурні підходи для обробки природної мови, де мовні команди інтерпретувалися в програмні процедури. Це давало змогу програмі виконувати складні команди, розуміючи не лише окремі слова, а й взаємозв'язки між ними.

3. ***З'ясування неоднозначностей.*** *SHRDLU* могла працювати з неоднозначними командами. Наприклад, якщо користувач вводив команду, яку можна було зрозуміти по-різному, програма намагалася уточнити її зміст через запитання.

4. ***Діалогова інтерактивність.*** *SHRDLU* підтримувала діалогову взаємодію, що давало змогу користувачам ставити уточнювальні питання, відповідати на питання програми і створювати більш природний та гнучкий процес спілкування.

5. ***Інтеграція логічного висновку.*** *SHRDLU* використовувала логічні правила для прийняття рішень. Це означало, що вона могла робити висновки на основі отриманих команд і виконувати логічні операції для досягнення поставленої мети.

Ці принципи допомогли закласти основу для створення складніших моделей *NLP*, які з часом розвинулися в системи, здатні опрацьовувати природну мову на рівні, наближеному до людського розуміння.

Приклад. На прикладі використання моделі *SHRDLU* проілюструємо її здатність надавати користувачам можливість взаємодіяти з комп'ютером за допомогою англійської мови, керуючи об'єктами у віртуальному «світі блоків» [16.21].

Розглянемо приклад взаємодії людини (користувача) з моделлю *SHRDLU* щодо розуміння природної мови та виконання дій на основі команд користувача.

- Користувач: «*Поклади червоний куб на зелений куб.*»
- SHRDLU: «*Гаразд.*» [Кладе червоний куб на зелений куб]
- Користувач: «*Чи є ще якісь куби на зеленому кубі?*»
- SHRDLU: «*Ні, більше немає кубів на зеленому кубі.*»
- Користувач: «*Перемісти зелений куб на великий синій куб.*»

- SHRDLU: «*Я не можу це зробити, бо червоний куб зараз на зеленому кубі.*»
- Користувач: «*Прибери червоний куб.*»
- SHRDLU: «*Гаразд.*» [Прибирає червоний куб зі зеленого куба]
- Користувач: «*Тепер поклади зелений куб на великий синій куб.*»
- SHRDLU: «*Гаразд.*» [Кладе зелений куб на великий синій куб]
- Користувач: «*Які об'єкти зараз знаходяться на великому синьому кубі?*»
- SHRDLU: «*Зелений куб.*»
- Користувач: «*Поклади червоний куб на зелений куб.*»
- SHRDLU: «*Гаразд.*» [Виконує команду]
- Користувач: «*Що знаходиться під червоним кубом?*»
- SHRDLU: «*Зелений куб.*»
- Користувач: «*Скільки кубів у світі блоків?*»
- SHRDLU: «*Є три куби: червоний, зелений і синій.*»

Особливості роботи SHRDLU на цьому прикладі:

• *розуміння контексту.* Програма пам'ятає поточний стан світу блоків та попередні дії;

• *виявлення обмежень. SHRDLU* може пояснити, чому певну команду не можна виконати (наприклад, «зайнятий куб»);

• *діалогова гнучкість.* Можливість ставити уточнювальні запитання і отримувати відповіді про поточний стан об'єктів;

• *інтерактивне маніпулювання об'єктами.* Виконання інструкцій на основі текстових команд.

Параметри і характеристики роботи моделі *SHRDLU*, отримані в процесі взаємодії людини (користувача) з машиною, наведено в табл. 16.3.

Висновок. Цей приклад демонструє, як за допомогою моделі *SHRDLU* здійснювалося моделювання процесу розуміння природної мови в межах структурованого завдання у віртуальному середовищі.

Модель *SHRDLU* є класичним прикладом системи, що працювала в обмеженому середовищі з чітко визначеними об'єктами та діями. Саме ці обмеження давали програмі можливість ефективно обробляти природну мову і виконувати поставлені завдання на ранньому етапі розвитку штучного інтелекту.

Попри певні обмеження, модель *SHRDLU* продемонструвала потенціал машин у розумінні команд людини та взаємодії з віртуальним середовищем. Для свого часу ця програма стала значним досягненням, наочно показавши, як можна поєднувати обробку мови з візуальною взаємодією та логічними висновками.

Отже, ранні моделі *ELIZA* і *SHRDLU* стали важливими етапами в розвитку *NLP*, забезпечивши основу для створення сучасних мовних систем,

як-от *GPT* і *T5*. Модель *ELIZA*, що слугувала попередником сучасних чатботів, маніфестувала, як шаблонні відповіді можуть створювати ілюзію інтелектуальної взаємодії. Водночас модель *SHRDLU* успішно інтегрувала лінгвістику, планування дій і логіку, наочно демонструючи можливості створення інтерактивних систем.

SHRDLU також мала значний вплив на подальший розвиток інтерактивних мовних систем, показуючи, як можна поєднувати обробку мови з логічними алгоритмами і плануванням. Цю модель використовували для навчання студентів основ штучного інтелекту, вона слугувала прототипом для складніших систем. На прикладі цих ранніх розробок видно, як моделі з обмеженими можливостями та вузьким контекстом поступилися місцем сучасним універсальним і масштабованим підходам, що стали основою сучасного *NLP*.

Таблиця 16.3

Параметри і характеристики роботи моделі *SHRDLU*, зафіксовані в процесі взаємодії людини (користувача) з машиною

Параметр	Опис	Кількісне значення
Розмір словника	Містить команди дій, запитання та назви об'єктів Словник містив команди дій («*поклади*», «*перемісти*»), запитання («*що знаходиться?*», «*скільки?*»), а також назви об'єктів («*куб*», «*піраміда*», «*великий*», «*червоний*» тощо)	~50—100 слів
Кількість об'єктів	Куби, піраміди різних кольорів і розмірів	5—15 об'єктів
Обмеження простору	Об'єкти можна ставити поруч або один на одного Простір був обмежений конкретним «світом блоків», де об'єкти можна було ставити один на одного або поруч	Оброблення одного речення за один раз
Мовна обробка	Обмежене розуміння контексту Глибина розуміння контексту була обмежена поточним станом світу блоків	Оброблення одного речення за один раз
Продуктивність	Працювала на комп'ютері DEC PDP-6; використання мови *Lisp*	~1 млн операцій за секунду
Час оброблення команди	Для простих команд (наприклад, «поклади куб на інший куб»)	Секунди або частки секунди
Кількість рядків коду	Код написаний мовою програмування *Lisp*	15 000—20 000 рядків
Обмеження обсягу пам'яті	Обмеження апаратної пам'яті комп'ютера PDP-6	До 64 КБ пам'яті

Другий етап (1970-ті — 2000-ні рр.). Статистичні мовні моделі

1. **Мовна модель *TF-IDF* (1970-ті рр.).** Мовну модель *TF-IDF (Term Frequency-Inverse Document Frequency)* вперше запропоновали Джерард Салтон (Gerard Salton) та його колеги в 1970-х рр. під час їхньої роботи над системою автоматичного індексування текстів *SMART (System for the Mechanical Analysis and Retrieval of Text)* [16.22]. Ця модель стала класичним прикладом статистичного підходу, оскільки ґрунтувалася на підрахунку частоти слів у документах *(TF)* і рідкісності цих слів у колекції документів *(IDF).* Важливість кожного слова в документі визначалася через його частотні характеристики, що зробило цей підхід суто статистичним. З огляду на це модель мала низку принципових особливостей, які відрізняли її від попередніх мовних моделей [16.22]:

1. ***Комбінація частотного аналізу та інформативності термінів.*** *TF-IDF* поєднує частоту появи терміна в конкретному документі *(TF)* з рідкісністю терміна у всьому корпусі документів *(IDF).* Цей підхід допомагає не лише визначати, які терміни є найпоширенішими в документі, а й враховувати, наскільки вони специфічні в контексті всієї колекції документів. Попередні моделі могли зосереджуватися лише на частотному аналізі слів (наприклад, загальне частотне словникове подання), не враховуючи контекст рідкісності термінів у ширшій колекції документів.

2. ***Оцінка значущості терміна через IDF.*** Компонента *IDF* знижує вагу термінів, які часто трапляються у багатьох документах, вважаючи їх менш інформативними. Це відрізняє *TF-IDF* від моделей, що просто враховували абсолютну частоту термінів, попри те, наскільки ці терміни були поширеними в усьому корпусі. Попередні мовні моделі не мали механізму для зниження ваги часто використовуваних термінів, що могло призводити до хибного акцентування уваги на загальних словах.

3. ***Простота обчислень.*** Модель *TF-IDF* є відносно простою для обчислень та реалізації, що зробило її надзвичайно популярною в інформаційному пошуку та *NLP.* Вона не потребує складних лінгвістичних ресурсів, таких як синтаксичні або семантичні аналізатори, що відрізняє її від складніших моделей. Попередні моделі могли містити більш складні алгоритми для оброблення тексту, але *TF-IDF* запропонувала баланс між ефективністю та простотою.

4. ***Широке застосування в інформаційному пошуку.*** Модель *TF-IDF* швидко стала стандартом у галузі інформаційного пошуку та текстового аналізу завдяки своїй здатності поліпшувати результати пошуку, відфільтровуючи неінформативні терміни. Попередні моделі мали менш точні методи оцінки релевантності документів, що часто призводило до менш якісних результатів у пошукових системах.

І дотепер модель *TF-IDF* залишається базовою для багатьох сучасних методів обробки тексту та аналізу даних, попри розвиток більш складних моделей, як-от нейронні мережі та моделі на основі контекстних векторів. Розглянемо приклад практичного використання моделі *TF-IDF* [16.22].

Приклад. Потрібно виконати аналіз тональності тексту або здійснити пошук релевантних документів у інформаційно-пошукових системах. Припустимо, що є набір текстових документів і ми хочемо знайти найбільш релевантні документи на основі певного запиту або виділити важливі ключові слова. Нехай є мовна модель *TF-IDF* з параметрами, наведеними в табл. 16.4.

Потрібно зробити витяг ключових слів із заданого набору документів:

Документ 1. *«Я люблю штучний інтелект. Штучний інтелект змінює світ.»*

Документ 2. *«Машинне навчання є частиною штучного інтелекту.»*

Документ 3. *«Математика лежить в основі машинного навчання.»*

Результати обчислення *TF-IDF* за формулою для термінів у цих документах (див. п. 14.3.8) наведено в табл. 16.5.

Отже, термін ***«штучний інтелект»*** є найбільш релевантним для першого документа, тоді як термін «машинне навчання» — для другого.

Таблиця 16.4

Кількісні параметри і характеристики моделі TF-IDF

Параметр	Опис	Кількісне значення
Обсяг даних	Підтримка текстових колекцій різного розміру	Від сотень до кількох мільйонів документів; до 1 ГБ тексту
Розмір словника	Кількість унікальних слів у документі чи колекції документів	10 000—100 000 термінів
Обчислювальна складність	Час, необхідний для обчислення ваг термінів	Від секунд до хвилин залежно від розміру корпусу
Продуктивність	Швидкість обчислення та масштабованість на сучасних системах	Швидке обчислення; підтримка у *scikit-learn* для *Python*

Таблиця 16.5

Результати обчислення *TF-IDF* для термінів у документах 1—3

Термін	TF-IDF (Документ 1)	TF-IDF (Документ 2)	TF-IDF (Документ 3)
Штучний	0,4	0,2	0
Інтелект	0,4	0,2	0
Машинне	0	0,4	0,2
Навчання	0	0,4	0,2
Математика	0	0	0,4

У третьому документі виділяється термін «математика». Цей приклад демонструє, як модель *TF-IDF* можна використовувати для виділення важливих термінів або пошуку релевантних документів у текстовій колекції.

2. **Мовна модель *Latent Semantic Analysis (LSA)* (1988).** Модель *Latent Semantic Analysis (LSA)*, також відома як *Latent Semantic Indexing (LSI)*, була розроблена наприкінці 1980-х рр. групою дослідників лабораторії *Bell Communications Research*, знаної як *Bellcore*, у складі Сьюзен Дюмо (Susan Dumais), Джорджа Фурнаса (George Furnas), Томаса Ландауера (Thomas Landauer), Річарда Харшмана (Richard Harshman) та інших [16.23]. Модель *LSA* було розроблено для поліпшення процесу інформаційного пошуку та автоматичного індексування тексту.

Як і *TF-IDF*, модель *LSA* також ґрунтується на статистичному підході, але використовує складніший математичний метод для виявлення латентних семантичних структур. Цей підхід охоплює побудову термін-документної матриці, яка представляє частоти появи слів у документах, і застосування сингулярного розкладу матриць для зниження розмірності. Отже, *LSA* використовує статистичні методи для виявлення прихованих зв'язків між словами і документами.

Основна мета *LSA* полягала в подоланні обмежень традиційних методів пошуку, що базувалися на простому зіставленні ключових слів між запитом і документом. Тому модель *LSA* допомагала знаходити документи, які були релевантними до пошукового запиту, навіть якщо в них використовувалися різні слова для опису подібних концепцій. Це досягалося завдяки виявленню прихованих семантичних зв'язків між термінами.

Принципові відмінності LSA від попередніх мовних моделей [16.23]:

1. ***Використання сингулярного розкладу матриці (Singular Value Decomposition, SVD)*** *до матриці термін-документ, щоб знизити її розмірність.* Це сприяє виявленню прихованих семантичних структур у тексті, зменшуючи вплив шуму та синонімів. На відміну від простих частотних моделей, таких як *TF-IDF*, що не враховують семантичні зв'язки між словами, *LSA* дає змогу знаходити подібність між термінами навіть у разі, якщо вони не збігаються повністю.

2. ***Автоматичне виявлення синонімів і полісемії.*** *LSA* автоматично виявляє синонімічні зв'язки між словами і обробляє полісемію (багатозначність слів) через їх позиціонування у спільному семантичному просторі. Більш ранні моделі, як-от *TF-IDF*, мали обмежену здатність до оброблення синонімів та полісемії, оскільки ґрунтувалися на поверхневому частотному аналізі.

3. ***Поліпшення якості інформаційного пошуку.*** *LSA* значно поліпшила результати інформаційного пошуку, допомагаючи знаходити документи, які є релевантними до пошукових запитів навіть за відсутності точного збігу

ключових слів. Попередні моделі могли втратити важливі документи, якщо в них використовувалися різні терміни для опису подібних концепцій.

Наведемо приклад практичного використання моделі *LSA (Latent Semantic Analysis)* для пошуку релевантних документів у інформаційно-пошукових системах [16.23].

Нехай *LSA* використовується для зниження розмірності текстових даних та виявлення прихованих семантичних зв'язків між словами й документами. Наприклад, система пошуку може знаходити документи, що є семантично близькими до запиту, навіть якщо в них не використовуються точні слова із запиту.

Приклад. Припустимо, є три документи:

• Документ 1. «*Штучний інтелект змінює світ. Машинне навчання є частиною штучного інтелекту.*»

• Документ 2. «*Математика важлива для машинного навчання та штучного інтелекту.*»

• Документ 3. «*Штучний інтелект та великі дані створюють нові можливості.*»

Потрібно зробити пошуковий запит: «*Машинне навчання математика*».

Процес роботи *LSA* складається з таких етапів:

1. *TF-IDF перетворення*. Створюється матриця термін-документ (кожний стовпчик відповідає документу, а рядок — слову).

2. *Сингулярний розклад матриці (SVD).* Зменшується розмірність матриці для виявлення прихованих зв'язків між словами та документами.

3. *Пошук релевантності.* Запит «*Машинне навчання математика*» порівнюється з отриманими семантичними векторами документів.

Кількісні параметри і характеристики мовної моделі *LSA*, використаної для вирішення поставленого завдання, наведено в табл. 16.6.

Таблиця 16.6

Кількісні параметри і характеристики моделі *LSA*

Параметр	Опис	Кількісне значення
Розмір даних	Обсяг текстових корпусів, з якими може ефективно працювати *LSA*	До 1 млн документів
Розмірність простору	Кількість прихованих факторів після *SVD*	100—300 прихованих факторів
Час оброблення	Складність обчислення *SVD* для матриці термін-документ	Години на великих корпусах
Швидкість пошуку	Час, необхідний для пошуку релевантних документів після обробки	Дуже швидко після побудови моделі
Поліпшення точності	Відносне підвищення точності семантичного пошуку завдяки *LSA*	На 10—20 % порівняно з *TF-IDF*

Результат:

1) найбільш релевантними будуть *Документ 1* і *Документ 2*, навіть якщо точний збіг слів не є повним;

2) LSA допомагає зрозуміти, що «*машинне навчання*» і «*математика*» є в цих текстах семантично пов'язаними.

3. **Мовна модель *Latent Dirichlet Allocation (LDA)* (2003).** Мовна модель *Latent Dirichlet Allocation (LDA)* була розроблена Девідом Блеєм (David Blei), Ендрю Енжі (Andrew Ng) і Міхаелем Джорданом (Michael Jordan) у рамках їхньої роботи в Університеті Каліфорнії Берклі (University of California, Berkeley) [16.24]. *LDA* є генеративною моделлю, яка описує процес «генерації» документів, що створюються через вибір тем із певного їх розподілу, а потім модель здійснює вибір слів із цих тем. Основна ідея моделі полягає в тому, що кожен документ є комбінацією кількох прихованих тем. Кожна тема становить собою розподіл імовірностей слів, а кожний документ є розподілом тем.

Модель використовує приховані змінні для подання тем у документах. Це допомагає моделі виявляти латентні структури в текстових даних, які не є явними. *LDA* використовує розподіл Діріхле для моделювання розподілів тем у документах і розподілів слів у темах. Це забезпечує гнучкість і варіативність у моделюванні тем та їх розподіленості.

Головні відмінності LDA від попередніх моделей [16.24]*:*

1. ***Модель припускає змішування тем.*** На відміну від попередніх моделей, які могли припускати, що документи мають одну домінантну тему, *LDA* дає можливість кожному документу бути сумішшю кількох тем. Це відображає реальну структуру текстів, які зазвичай містять кілька тем.

2. ***Імовірнісна природа.*** *LDA* є ймовірнісною моделлю, тобто вона працює з розподілами ймовірностей, а не з фіксованими категоріями чи точними підрахунками. Це допомагає моделі краще справлятися з невизначеністю та варіативністю в текстах.

3. ***Автоматичне виявлення тем.*** *LDA* автоматично визначає кількість тем і їх зміст без потреби в ручному налаштуванні. Це контрастує з деякими попередніми методами, які могли потребувати ручного визначення кількості категорій або тем.

4. ***Приховані теми.*** *LDA* моделює теми як латентні (приховані) змінні, що сприяє виявленню концептуальних структур у даних, які не є явними з погляду простих статистичних підрахунків.

Розглянемо приклад практичного використання моделі *LDA*. Нехай *LDA* використовується для тематичного моделювання у великій колекції документів.

Приклад. Задано колекцію з 1000 наукових статей на різні теми (наприклад, машинне навчання, медицина, екологія). Потрібно автоматично визначити, про що йде мова у текстах, не читаючи кожний документ вручну.

***Кроки роботи мовної моделі LDA*:**

1. **Попереднє оброблення текстів** з використанням *LDA*, параметри і характеристики якої наведено в табл. 16.7. Тексти очищуються від стоп-слів, перетворюються на числові вектори на основі термінів.

2. **Тематичне моделювання.** *LDA* автоматично визначає K = 5 тем у текстовій колекції і повертає розподіл слів для кожної з п'яти тем (табл. 16.8).

Цей приклад демонструє, як *LDA* може автоматично визначати теми у великому текстовому корпусі, забезпечуючи ефективну класифікацію та аналіз даних.

Отже, серед наведених статистичних мовних моделей *TF-IDF (Term Frequency-Inverse Document Frequency)* стала найбільш відомою і широко вживаною. Це пов'язано з тим, що *TF-IDF* є простою в реалізації, має зрозумілу математичну основу, не потребує значних обчислювальних ресурсів.

Таблиця 16.7

Кількісні параметри і характеристики моделі *LDA*

Параметр	Опис	Кількісне значення
Розмір даних	Обсяг текстових корпусів, які модель може ефективно обробляти	До 1 млн документів
Кількість тем (K)	Кількість прихованих тем, які визначає модель	5—50 тем
Розмір словника	Кількість унікальних слів у корпусі після попереднього оброблення	10 000— 100 000 термінів
Обчислювальна складність	Час виконання залежно від кількості слів (n), документів (m) і тем (K)	$O(n \times m \times K)$
Час обробки	Тривалість побудови моделі для великого корпусу даних	Від хвилин до годин
Продуктивність	Швидкість обробки та масштабованість на сучасних процесорах	Добре масштабується; підтримується у *Gensim* та *scikit-learn*

Таблиця 16.8

Розподіл слів для визначених п'яти тем

Тема	Найбільш вагомі слова	Інтерпретація
Тема 1	«Алгоритм», «модель», «навчання», «дані», «нейронна мережа»	Машинне навчання
Тема 2	«Пацієнт», «лікар», «медичний», «діагноз», «лікування»	Медицина
Тема 3	«Вуглець», «довкілля», «температура», «зміни», «екологія»	Екологія і клімат
Тема 4	«Економіка», «ринок», «фінанси», «банк», «гроші»	Фінанси та економіка
Тема 5	«Програмування», «код», «система», «технології», «комп'ютер»	Комп'ютерні науки

Третій етап (2010-ті рр). Семантичне подання природної мови

1. **Мовна модель *Word2Vec* (2013).** Мовна модель *Word2Vec (Word to Vector)* була розроблена дослідницькою групою компанії *Google* під керівництвом Томаша Міколова (Tomash Mikolov) у 2013 р. [16.25]. Автори запропонували дві нові архітектури моделей для обчислення безперервних векторних подань слів на основі великих наборів даних. Було продемонстровано, що ці векторні подання забезпечують високу продуктивність у завданнях вимірювання синтаксичної й семантичної подібності слів. Створення моделі *Word2Vec* тісно пов'язане з розвитком глибокого навчання та семантичного подання природної мови в 2010-ті рр. Ця модель стала однією з перших, яка використовувала нейронні мережі з цією метою. Модель *Word2Vec* мала кілька ключових відмінних від попередніх мовних моделей особливостей, які зробили її революційною на момент появи [16.25]:

1. ***Навчання вбудованих векторів (Word Embeddings).*** До появи *Word2Vec* слова в тексті часто зображували через «мішок слів» *(bag-of-words)* або через «одногарячі» вектори *(one-hot encoding)*. У цих підходах кожне слово зображалося як вектор з одним значенням «1» та іншими — нулями, що не враховувало семантичних зв'язків між словами. В моделі *Word2Vec* запропоновано новий підхід, де кожне слово зображено у вигляді щільного багатовимірного вектора (наприклад, у 300-вимірному просторі), який вивчається на основі контексту слова. Це дало змогу враховувати семантичну близькість між словами, що раніше було неможливо.

2. ***Контекстно залежне подання слів.*** У більшості попередніх моделей слова розглядалися ізольовано, без урахування їх контексту в реченні. Наприклад, у моделі типу *N*-грам оцінювалися ймовірності слів, що не давало можливості ефективно враховувати контекст на великій відстані. Водночас модель *Word2Vec* використовує контекстні слова (навколишні слова) для навчання векторів. Модель навчається на завданнях прогнозування слова за його контекстом *(Continuous Bag of Words, CBOW)* або прогнозування контексту за словом *(Skip-gram)*, що дає можливість захоплювати більш складні зв'язки між словами. У цій моделі *CBOW* та *Skip-gram* є спрощеними версіями одношарових нейронних мереж із прямим зв'язком *(feedforward neural networks)*, які використовують для створення векторних подань слів *(word embeddings)*. Їх було запропоновано в алгоритмі роботи моделі *Word2Vec*.

3. ***Ефективність і масштабованість.*** У попередніх моделях їх навчання для великих текстових корпусів було дуже ресурсозатратним і потребувало значної обчислювальної потужності. В моделі *Word2Vec* закладено більш ефективні алгоритми для навчання вбудованих векторів слів, що дало змогу обробляти великі обсяги тексту швидко і з меншими ресурсни-

ми затратами. Це стало можливим завдяки використанню простих нейронних мереж та оптимізації процесу навчання.

4. ***Здатність захоплювати семантичні і синтаксичні зв'язки.*** У попередніх моделях важко було захопити складні зв'язки між словами, як-от семантичні або синтаксичні асоціації. Разом з тим однією з важливих особливостей *Word2Vec* є здатність захоплювати семантичні зв'язки між словами. Вектори, навчені моделлю *Word2Vec*, можуть виявити зв'язки між словами на рівні аналогій, наприклад, «король» — «чоловік» = «королева» — «жінка».

5. ***Простота інтеграції та використання.*** У попередніх моделях використовувалися складні механізми, які було важко інтегрувати в реальні системи обробки природної мови. На відміну від цього модель *Word2Vec* виявилася простою у використанні та зручною для інтеграції в різні системи *NLP*, що сприяло її популяризації та широкому застосуванню.

Ці відмінності зробили модель *Word2Vec* у певному розумінні революційною в галузі обробки природної мови і проклали шлях до розвитку інших сучасних підходів на основі глибокого навчання. Наведемо приклад практичного використання мовної моделі *Word2Vec* у завданні створення рекомендаційної системи для інтернет-магазину.

Приклад. Розглянемо завдання створення рекомендацій клієнтам на основі використання мовної моделі *Word2Vec* і аналізу взаємозв'язків між товарами, що їх купували разом. Нехай маємо файли придбаних речей клієнтами у форматі:

- Клієнт 1: [«телефон», «чохол», «зарядний пристрій»].
- Клієнт 2: [«телефон», «навушники», «захисне скло»].
- Клієнт 3: [«планшет», «клавіатура», «чохол»].

Етапи процесу виконання завдання такі [16.26]:

1. *Підготовка даних.* Кожна купівля перетворюється на «речення», а кожен товар — на «слово».

2. *Навчання моделі.* Використання *архітектури Skip-gram* для навчання векторів слів (товарів).

3. *Рекомендації.* Після навчання модель здатна визначати найбільш подібні товари, наприклад:

- «телефон» → [«чохол», «зарядний пристрій», «навушники»].
- «планшет» → [«клавіатура», «чохол»].

Для вирішення цього завдання використовували мовну модель *Word2Vec*, кількісні параметри і характеристики якої наведено в табл. 16.9.

Цей приклад демонструє, як мовну модель *Word2Vec* можна застосовувати в реальних бізнес-задачах, наприклад для поліпшення рекомендацій у системах електронної комерції.

2. **Мовна модель *GloVe* (2014).** Мовну модель *GloVe* (*Global Vectorsfor Word Representation*) розробили дослідники зі Стенфордського університе-

ту під керівництвом Джеффрі Пенінгтона (Jeffrey Pennington), Річарда Соучера (Richard Socher) та Крістофера Маннінга (Christopher D. Manning). Модель було вперше репрезентовано у 2014 р. [16.27].

Як і *Word2Vec, GloVe* є моделлю, що використовує вектори для подання слів. Вона поєднує статистичну інформацію про частоту спільної появи слів і навчається на глобальній матриці співвідношення слів у великому корпусі текстів. Це дає можливість створити векторний простір, де слова мають змістову підструктуру, що відображає їх семантичну й синтаксичну подібність.

Особливістю *GloVe* є поєднання переваг глобальної матричної факторизації (яка враховує статистичну інформацію з усього корпусу) та локальних методів контекстного вікна (які акцентують увагу на зв'язках між сусідніми словами). Це забезпечує високу ефективність моделі у завданнях обробки природної мови.

Модель GloVe відрізняється від попередніх мовних моделей кількома ключовими концептуальними особливостями [16.27].

1. ***Глобальний підхід до статистики контексту.*** Основна ідея *GloVe* полягає у використанні глобальної статистики спільної появи слів у тексті. Модель створює термін-контекстну матрицю, в якій елементи позначають частоту спільного вживання пар слів у великому текстовому корпусі. Після цього *GloVe* навчає вектори слів так, щоб відношення між векторами відповідали відношенням між частотами спільного вживання слів. На відміну від *GloVe* такі моделі, як *Word2Vec*, покладаються переважно на локальний контекст (безпосередньо сусідні слова) для навчання векторів слів. Вони не враховують глобальні частоти спільного вживання слів у всьому текстовому корпусі.

Таблиця 16.9

Кількісні параметри та характеристики моделі *Word2Vec* для створення рекомендацій клієнтам інтернет-магазину

Параметр	Опис	Значення
Розмір даних	Кількість клієнтів і придбань у текстовому корпусі	10 000 клієнтів, 100 000 придбань
Розмір словника	Кількість унікальних товарів у корпусі	~5 000 унікальних товарів
Розмір векторного подання	Кількість компонентів у векторі для кожного слова (товару)	100—300 компонентів (зазвичай 200)
Алгоритм навчання	Метод, який використовується для побудови векторних подань	*Skip-gram* (оптимальний для великих даних)
Час навчання	Тривалість навчання моделі на корпусі заданого розміру	До 1 години на сучасному процесорі
Точність рекомендацій	Покращення точності порівняно з базовими моделями	Зростання на 15—25 %

2. ***Логарифмічний масштаб частот.*** Модель використовує логарифмічний масштаб частот спільного вживання слів для створення векторних подань. Це дає змогу краще відображати зв'язки між словами, особливо для рідкісних або дуже частих слів. Інші моделі, як, наприклад, *Word2Vec,* не використовують явний логарифмічний підхід до частот, що іноді може призводити до менш точного захоплення семантичних зв'язків.

3. ***Семантичні зв'язки між словами.*** Вектори, навчені за допомогою *GloVe,* ефективно захоплюють семантичні зв'язки між словами. Вони визначають, як часто слова з'являються разом у тексті, що допомагає моделі добре захоплювати аналогії та семантичні зв'язки. *Word2Vec* також може захоплювати семантичні зв'язки, але робить це через навчання на локальному контексті, що може бути менш ефективним порівняно з глобальним підходом *GloVe.*

4. ***Фіксований обсяг моделі.*** *GloVe* використовує статичний підхід до навчання, де вектори слів фіксуються після навчання на конкретному корпусі. Це означає, що модель має постійну структуру та розмір. Попередня модель *Word2Vec* може адаптуватися під нові дані, але це потребує додаткового навчання, що змінює векторні подання.

5. ***Глобальна статистика.*** *GloVe* використовує глобальну статистику (частоти спільного вживання слів у всьому корпусі), тоді як *Word2Vec* базується на локальних контекстах (сусідніх словах).

6. ***Алгоритм навчання.*** *GloVe* використовує матричну факторизацію для навчання векторів, натомість *Word2Vec* з аналогічною метою застосовує спрощені версії одношарових нейронних мереж із прямим зв'язком *(CBOW* та *Skip-gram).*

7. ***Точність у захопленні аналогій.*** Обидві моделі добре вирішують завдання захоплення аналогій, але *GloVe* зазвичай може краще впоратися із захопленням глобальних зв'язків між словами.

Попри деякі відмінності, мовні моделі *Word2Vec* і *GloVe* стали основоположними для розвитку семантичного подання слів у природній мові і проклали шлях для подальших досягнень у сфері глибокого навчання. Вони показали, що можна ефективно навчати вбудовані вектори слів, які можуть бути використані для різних завдань обробки природної мови, як-от класифікація тексту, машинний переклад і пошукові системи. Ці моделі відіграли важливу роль у формуванні сучасних підходів до обробки природної мови, що згодом ознаменувало революцію трансформерів і дало поштовх до розроблення складніших моделей на основі глибоких нейронних мереж — *Transformer, BERT, GPT, T5* та ін. Разом з тим завдяки своїй гнучкості *Word2Vec* стала більш універсальною і переважною в практичному застосуванні.

Приклад. Розглянемо завдання аналізу семантичної подібності та аналогій у текстовому корпусі на основі використання мовної моделі *Word2Vec* [16.12]. Нехай маємо великий текстовий корпус (наприклад, статті з новин або літературні тексти).

Постановка завдання:

1. Виявити слова, які мають близькі значення або використовуються в подібному контексті (наприклад, «король» — «принц», «монарх», «королева»).

2. Автоматично знайти семантичні аналогії між словами (наприклад, «король — чоловік + жінка = королева»).

3. Використати ці знання для завдань, як-от розширення пошуку інформації, автоматичного групування понять або поліпшення рекомендаційних систем.

4. Зберегти семантичні зв'язки між словами, включно із синонімами та родинними поняттями.

5. Розпізнати лінійні зв'язки між словами, наприклад, аналогії («король» — «чоловік», «королева» — «жінка»).

6. Забезпечити точність і швидкість у разі роботи на великих текстових наборах.

7. У підсумку потрібно згенерувати такі вихідні дані: список слів із високою семантичною подібністю; для кожного слова знайти найближчі слова у векторному просторі за допомогою косинусної подібності (наприклад, для «король» — «монарх», «принц», «королева»); здійснити пошук аналогій, наприклад для «король — чоловік + жінка» знайти слово, яке найкраще відповідає результату («королева»).

***Як вхідні дані маємо текстовий корпус*:**

• обсяг — 1,6 млрд слів (наприклад, корпус *Google News*);

• джерело — тексти англійською мовою, очищені від зайвих символів;

• форма — набір документів, розподілений на речення.

Вирішення завдання:

• *здійснено тренування моделі Word2Vec* для навчання векторних подань слів. Виконано налаштування гіперпараметрів: розмір вектора *(embedding size)* — 300 вимірів; контекстне вікно *(window size)* — 5 слів; кількість циклів навчання — 5; обчислювальні ресурси — *GPU* або кластер *CPU*;

• *виконано обчислення подібності:* визначено косинусну подібність між векторами слів для виявлення найближчих слів у семантичному просторі; виконано векторні арифметичні операції для перевірки аналогій (наприклад, «король — чоловік + жінка»).

Результати роботи моделі Word2Vec:

• встановлено семантичну близькість. Найближчі слова до «король» — «монарх» (0,89), «принц» (0,78), «королева» (0,85);

• встановлено аналогію: «король — чоловік + жінка = королева» (косинусна подібність результату з «королева» — 0,89);

• здійснено групування понять. Слова на кшталт «король», «королева», «принц», «монарх» згруповано разом у семантичному просторі.

Обчислювальні витрати. Тренування моделі на 1,6 млрд слів здійснено за 20 хвилин на *GPU*; виконання пошуку подібних слів або аналогій виконано за мілісекунди [16.12].

Четвертий етап (2017 — сьогодення). Мовні моделі, побудовані на трансформерах

Мовні моделі, побудовані на трансформерах і створені на нейронних мережах, становлять собою два різних покоління технологій обробки природної мови. Нейронні мережі (рекурентні *(RNN)* і згорткові *(CNN)*) працюють із текстом як із послідовністю даних, використовуючи внутрішні стани для оброблення контексту. Цей підхід забезпечує здатність моделі враховувати локальні та короткострокові залежності, але має обмеження в моделюванні глобального контексту, що пов'язано з втратою інформації на довгих послідовностях та складністю паралельного оброблення даних.

Поява трансформерів у 2017 р. докорінно змінила підхід до *NLP* завдяки механізму самоуваги *(Self-Attention Mechanism)*, який може одночасно враховувати всі взаємозв'язки між словами в тексті. Це забезпечує трансформерам перевагу в моделюванні глобального контексту, ефективності обробки та масштабованості, даючи змогу працювати з великими обсягами даних і створювати високоякісні векторні зображення слів.

Трансформери вирізняються здатністю обробляти текст паралельно, що значно прискорює навчання, на відміну від послідовної обробки в *RNN*. Крім того, вони ефективно моделюють довгострокові залежності, які важко опрацьовуються нейронними мережами попереднього покоління через проблему загасання градієнтів. Хоча навчання трансформерів є ресурсомістким, вони демонструють універсальність і високу продуктивність у таких завданнях як машинний переклад, генерування тексту та відповіді на запитання.

Порівняно з традиційними нейронними мережами трансформери стали новим стандартом у *NLP*, що забезпечує глибше розуміння тексту завдяки здатності враховувати глобальний контекст і масштабуватися до величезних моделей, як-от *GPT* і *BERT*. Вони відкрили нову еру обробки природної мови, заклавши основу для сучасних інтерактивних систем і мультимодальних застосувань.

1. **Мовна модель *Transformer* (2017).** Мовна модель *Transformer* була розроблена дослідницькою групою *Google Brain* у 2017 р. Основними авторами моделі стали А. Васвані (Ashish Vaswani), Н. Шазер (Noam Shazeer), Н. Пармар (Niki Parmar), Я. Усзкорейт (Jakob Uszkoreit), Л. Джонс (Llion Jones), А. Н. Гомес (Aidan N. Gomez), Л. Кайзер (Lukasz Kaiser) та І. Полосухін (Illia Polosukhin) [16.28]. Модель *Transformer* повністю замінила рекурентні і згорткові мережі механізмом самоуваги, що дало можливість сут-

тєво прискорити обчислення і підвищити ефективність роботи з довгими послідовностями завдяки паралельному обробленню всіх слів у реченні.

Експерименти в галузі машинного перекладу показали, що ця модель перевершує попередні за якістю, є більш паралелізованою і потребує значно менше часу на навчання. *Transformer* став основою для багатьох інших моделей — *BERT, GPT, T5* та інших, які використовують подібні архітектурні підходи для вирішення різноманітних завдань у сфері обробки природної мови.

Модель *Transformer* має кілька ключових концептуальних особливостей, які відрізняють її від попередніх мовних моделей [16.28].

1. ***Архітектура, заснована на механізмі самоуваги (Self-Attention).*** Модель *Transformer* повністю покладається на механізм самоуваги для оброблення залежностей між елементами вхідної послідовності, що дає можливість моделі враховувати всі слова в реченні одночасно, а не обмежуватися локальними вікнами, як у випадку рекурентних нейронних мереж *(RNN).* Це забезпечує паралельність в обчисленнях, що значно прискорює процес навчання і підвищує ефективність використання ресурсів.

2. ***Відмова від рекурентних і згорткових шарів.*** На відміну від попередніх моделей, таких як *RNN* або *CNN, Transformer* не використовує рекурентні або згорткові шари для оброблення послідовностей. Це надає моделі більшої масштабованості та ефективності щодо довгих послідовностей, оскільки вона не страждає від проблеми градієнта, що загасає, і має фіксований час оброблення послідовності.

3. ***Механізм «багатоголової» уваги (Multi-Head Attention).*** Модель використовує декілька незалежних механізмів уваги (голів), які паралельно обробляють різні аспекти вхідних даних. Це дає моделі можливість захоплювати різноманітні види зв'язків між словами, що покращує її здатність розуміти складні структури в тексті.

4. ***Позиційне кодування (Positional Encoding).*** Оскільки *Transformer* не має рекурентної структури, яка б природно захоплювала інформацію про порядок слів у послідовності, було введено позиційне кодування. Завдяки цьому модель розуміє порядок слів у реченні, враховуючи послідовність елементів.

5. ***Декодер для генерування послідовностей.*** Модель складається з двох основних частин — кодера та декодера. Кодер обробляє вхідні дані, а декодер генерує вихідні дані, при цьому декодер також використовує механізм уваги для врахування інформації з кодера. Ця двостадійна структура забезпечує моделі *Transformer* ефективне вирішення завдань послідовного моделювання, наприклад машинного перекладу.

Завдяки цим концептуальним особливостям мовна модель *Transformer* стала основою для багатьох інших моделей — *BERT, GPT, T5* та ін., які використовують подібні архітектурні підходи для вирішення різноманітних завдань у сфері обробки природної мови.

Модель *Transformer* широко застосовують у різних галузях. Одним із перших і найвідоміших практичних застосувань є машинний переклад.

***Приклад*. Машинний переклад.** У статті «Attention Is All You Need» [16.28] автори продемонстрували ефективність моделі *Transformer* на завданні перекладу з англійської на німецьку та з англійської на французьку мови. Кількісні параметри і характеристики моделі для цього завдання наведено в табл. 16.10.

Отримані результати:

• *переклад з англійської на німецьку мову* — 28,4 *BLEU* (поліпшення на 2 *BLEU* порівняно з попередніми найкращими результатами);

• *переклад з англійської на французьку мову* — 41,8 *BLEU* (новий найкращий результат для одномодельних систем).

BLEU (Bilingual Evaluation Understudy) — це метрика, що використовується для оцінювання якості машинного перекладу, порівнюючи перекладений текст із одним або кількома референтними (еталонними) перекладами. *BLEU* вимірює подібність між машинним перекладом і перекладом, зробленим людиною, за допомогою кількісного аналізу збігів. Тобто 28,4 *BLEU* і 41,8 *BLEU* означає, що модель *Transformer* забезпечила середню якість перекладу з англійської на німецьку мову і з англійської на французьку мову на 28,4 % і 41,8 % подібності з еталонними перекладами, відповідно (з урахуванням *n*-грам).

Цей приклад демонструє, як модель *Transformer* у свій час (2017 р.) встановила нові стандарти у завданнях машинного перекладу, забезпечуючи високу якість перекладу та ефективність навчання.

2. **Мовна модель BERT (2018).** *BERT (Bidirectional Encoder Representations from Transformers)* — мовна модель, що здійснила революцію у сфері обробки природної мови *(NLP)*. Розроблена дослідниками компанії *Google* — Джейкобом Девліном, Мінг-Вей Чангом, Кентоном Лі та Крістіною Таутановою, вона була вперше репрезентована у жовтні 2018 р. [16.14]. Ця модель встановила нові стандарти у виконанні численних завдань *NLP*,

Таблиця 16.10

Кількісні параметри і характеристики застосування моделі *Transformer* у машинному перекладі

Параметр	Опис	Значення
Набір даних	Корпус для навчання моделі	*WMT* 2014 (English-to-German, English-to-French)
Розмір моделі	Кількість параметрів у моделі	65 млн параметрів
Ресурси для навчання	Обчислювальні ресурси, використані для навчання	8 GPU
Час навчання	Тривалість навчання моделі на зазначених ресурсах	3,5 дні

як-от запитання-відповіді, мовна інференція та інші завдання, що потребують високого рівня розуміння тексту.

BERT побудована на основі архітектури трансформерів, яка використовує механізм самоуваги *(Self-Attention Mechanism)*. Це дає змогу моделі ефективно аналізувати текст і враховувати взаємозв'язки між словами, навіть якщо вони далеко одне від одного в реченні. Основною особливістю *BERT* є її здатність працювати з глибокими двоспрямованими поданнями тексту, що означає врахування контексту як ліворуч, так і праворуч. Такий підхід забезпечує глибше розуміння значення слів у тексті, краще інтерпретування складних речень.

Однією з ключових інновацій *BERT* є її підхід до навчання. Модель навчається на великих текстових корпусах за допомогою попереднього навчання *(pre-training)*, що сприяє узагальненню знань з різних джерел. Після цього модель може бути адаптована до конкретних задач за допомогою додаткового навчання *(fine-tuning)*, що робить її універсальною і придатною для вирішення різноманітних проблем обробки природної мови.

Завдяки своїм інноваційним підходам *BERT* суттєво поліпшила якість виконання багатьох завдань *NLP*. Глибокий двоспрямований контекст, ефективність самонавчання та універсальність зробили цю модель одним із ключових досягнень у сучасній обробці природної мови. *BERT* відкрила нові можливості для дослідників і практиків, встановивши новий стандарт для мовних моделей.

Модель має кілька ключових відмінностей від попередніх моделей, які значно покращують її здатність розуміти контекст і виконувати різні завдання обробки природної мови [16.14, 16.29].

1. ***Двоспрямоване навчання (Bidirectionality).*** Попередні моделі використовували односпрямоване або контекстуальне подання тексту. Разом з тим *BERT* аналізує текст у двох напрямках одночасно — зліва направо і справа наліво. Це дає моделі можливість глибше розуміти контекст кожного слова, враховуючи як попередні, так і наступні слова в реченні.

2. ***Маскування слів (Masked Language Modeling, MLM).*** Більшість попередніх мовних моделей генерують текст послідовно, передбачаючи наступне слово на основі попереднього контексту. На відміну від них у процесі навчання моделі *BERT* випадкові слова в тексті маскуються (замінюються на спеціальний токен [MASK]) і модель має передбачати ці слова, використовуючи контекст з обох боків. Це допомагає *BERT* вивчати більш глибокі зв'язки між словами.

3. ***Прогнозування наступного речення (Next Sentence Prediction, NSP).*** Попередні моделі не використовували спеціалізованих завдань для навчання на рівні речень. Вони фокусували лише на генеруванні послідовності слів. Разом з тим *BERT* навчається на завданні прогнозування наступного речення. Модель отримує два речення і має визначити, чи є наступне ре-

чення логічним продовженням попереднього. Це сприяє кращому розумінню зв'язків між реченнями.

4. ***Універсальність і налаштування для конкретних завдань (Fine-tuning).*** Попередні моделі хоча й були потужними, не завжди легко налаштовувалися для різних завдань *NLP.* На відміну від них модель *BERT,* після попереднього навчання на великих корпусах тексту, може бути легко налаштована *(fine-tuned)* для конкретних завдань, таких як класифікація тексту, питання-відповіді або розпізнавання сутностей, з використанням відносно невеликих наборів даних.

5. ***Архітектура трансформерів (Transformers Architecture).*** Більшість попередніх моделей *NLP* ґрунтувалися на рекурентних нейронних мережах *(RNN)* або їх варіантах — *LSTM* чи *GRU*, які мали обмеження у зв'язку з послідовним характером оброблення даних. Водночас *BERT* використовує трансформери, які обробляють весь текст паралельно, а не послідовно, що підвищує ефективність і швидкість оброблення великих обсягів тексту.

Ці відмінні особливості зробили *BERT* однією з найпотужніших моделей для вирішення різноманітних завдань *NLP* — класифікації, розпізнавання сутностей, заповнення пропусків у тексті тощо.

Модель *BERT* широко застосовують у різних завданнях обробки природної мови. Одним із практичних прикладів є її використання для семантичної кластеризації текстових даних.

Приклад. Розглянемо завдання семантичної кластеризації текстів на основі інтеграції можливостей *BERT* із модифікованим алгоритмом *Latent Dirichlet Allocation (LDA)* [16.30]. Метою є підвищення точності та адекватності тематичного моделювання текстових даних.

Для вирішення цього завдання використовували модель *BERT*, кількісні параметри і характеристики якої наведено в табл. 16.11.

Таблиця 16.11

Кількісні параметри і характеристики моделі BERT

Параметр	Опис	Кількісне значення
Розмір моделі *BERT*	Кількість шарів, прихованих розмірностей, голів механізму уваги та параметрів	*BERT-Base:* 12 шарів, 768 прихованих розмірностей, 12 голів механізму уваги, 110 млн параметрів *BERT-Large:* 24 шари, 1024 приховані розмірності, 16 голів механізму уваги, 340 млн параметрів
Набір даних	Корпус текстів для навчання і тестування	Текстові набори даних із різними емоційними станами для оцінки ефективності методу
Метрики оцінювання	Метрики для аналізу кластерів	Внутрішньокластерна та міжкластерна відстані; коефіцієнт силуету для оцінки когерентності та інтерпретованості кластерів

Результати. Порівняння результатів використання моделі *BERT* із традиційними моделями *LDA* у завданнях кластеризації текстів демонструє значні переваги першої [16.30]. Основні показники, як-от когерентність кластерів, інтерпретованість і адаптивність до нових даних, підтверджують ефективність інтеграції моделі *BERT* у процес семантичного аналізу. Зокрема, модель *BERT* показала покращення когерентності кластерів на 15 % порівняно з *LDA*. Середня когерентність для *LDA* становила 0,70, тоді як для підходу, що використовує *BERT,* цей показник зріс до 0,85. Це свідчить про те, що кластери, сформовані за допомогою моделі *BERT,* є більш семантично зв'язними й послідовними.

Інтерпретованість кластерів також істотно зросла. Коефіцієнт силуету, який визначає якість кластеризації, для моделей *LDA* становив 0,75, а для *BERT* цей показник зріс до 0,85. Це демонструє чіткіше відокремлення кластерів і поліпшену їхню семантичну структуру, що робить результати кластеризації більш зрозумілими й корисними.

Ще однією перевагою моделі *BERT* є її висока адаптивність до нових тематичних наборів даних. У цьому аспекті точність розпізнавання тематичних змін для *LDA* становила 85 %, тоді як *BERT* досягла рівня 92 %. Це підкреслює здатність моделі *BERT* швидко пристосовуватися до нових умов і забезпечувати високу ефективність навіть у змінних середовищах.

Отже, модель *BERT* не тільки перевершує *LDA* за основними метриками якості кластеризації, а й відкриває нові можливості для більш точного, зрозумілого та адаптивного семантичного аналізу текстів. Це робить її незамінним інструментом у завданнях обробки природної мови.

3. **Мовна модель *T5* (2019).** Мовна модель *T5 (Text-to-Text Transfer Transformer)* була розроблена компанією *Google Research* і вперше репрезентована у 2019 р. в роботі [16.31]. В основу моделі покладено підхід, який уніфікує різноманітні завдання обробки природної мови, перетворюючи їх на формат «текст — текст», що дало змогу використовувати єдину архітектуру для вирішення різних проблем NLP.

До концептуальних засад моделі T5 належать [16.31]:

1. ***Трансформерна архітектура.*** *T5* базується на архітектурі трансформера, яка є основою для багатьох сучасних мовних моделей. Це дає можливість моделі враховувати контекст на основі всього тексту.

2. ***Універсальність тексту.*** *T5* використовує підхід, за якого всі завдання обробки природної мови формулюються як завдання перетворення тексту на текст. Наприклад, переклад, відповіді на запитання та інші завдання обробки тексту виконуються через генерування тексту, що спрощує підхід до навчання і використання моделі.

3. ***Попереднє навчання і тонке налаштування.*** *T5* спочатку навчається на великому корпусі текстів (попереднє навчання) і потім адаптується до конкретних завдань за допомогою тонкого налаштування.

4. ***Масштабування.*** Модель *T5* було навчено на великому наборі даних, завдяки чому вона може виконувати широкий спектр завдань.

5. ***Універсальний текстовий підхід.*** *T5* відрізняється від попередніх моделей тим, що всі завдання обробки природної мови формулюються як завдання генерування тексту, що спрощує і уніфікує підхід до оброблення різних типів завдань.

6. ***Зв'язування завдань.*** У *T5* не потрібно використовувати окремі архітектури для різних завдань. Модель можна налаштувати для будь-якого завдання обробки тексту, просто змінивши його формулювання.

7. ***Акуратне навчання.*** Модель було навчено на наборі даних *C4 (Colossal Clean Crawled Corpus)*, який є великим очищеним набором текстів з Інтернету, що допомагає підвищити якість і результати навчання моделі.

Модель *T5* широко застосовують у різних завданнях обробки природної мови. Одним із практичних прикладів є використання *T5* для перетворення природних мовних запитів на *SQL*-запити, що надає користувачам млжливість взаємодіяти з базами даних без необхідності знання мови *SQL*.

Приклад. Завдання. Вирішити завдання автоматичної трансляції запитів, сформульованих природною мовою, у відповідні запити мови програмування *SQL (Structured Query Language)* за допомогою моделі *T5* з метою взаємодії користувачів з базами даних, враховуючи, що *SQL* — мова програмування, використовувана для роботи з реляційними базами даних [16.32].

Алгоритм вирішення завдання трансляції природних мовних запитів у *SQL* за допомогою моделі *T5:*

1. ***Формулювання завдання.*** Основна мета полягала в тому, щоб автоматизувати процес перетворення запитів природною мовою (наприклад, «Покажи всіх клієнтів, які зробили покупки в березні») на відповідні *SQL*-запити для баз даних. Це дало змогу користувачам без знань *SQL* взаємодіяти з базами даних.

2. ***Підготовка набору даних*:**

• *збирання даних:* Використано спеціалізовані набори даних, що містять пари «природний мовний запит — SQL-запит». Наприклад, запит: «*Які продукти продано більш ніж 100 разів?*». *SQL: SELECT product_name FROM sales WHERE quantity > 100*;

• *попереднє оброблення.* Полягає в очищенні тексту, нормалізації запитів та усуненні дублювання.

3. ***Навчання моделі T5*:**

• *архітектура T5.* Використовувалась як базова модель завдяки її гнучкості в обробленні завдань формату «текст — текст»;

• *формат даних.* Обидві частини (природний мовний запит і *SQL*-запит) було подано у вигляді текстових послідовностей;

• *навчання.* Модель навчалася на великих наборах даних із оптимізацією метрики крос-ентропії.

4. ***Тестування моделі*:**

• *Оцінювання точності.* Для тестування використовували нові запити, не внесені до навчального набору;

• *метрики.* Точність визначалась як відсоток повністю коректно згенерованих *SQL*-запитів *(Exact Match Accuracy).*

5. ***Оптимізація*:**

• *Fine-tuning.* Модель було додатково налаштовано на менших спеціалізованих наборах даних, характерних для конкретних баз даних;

• *адаптація.* Враховано специфіку запитів до інформаційних систем та сховищ даних *OLTP (Online Transaction Processing),* які забезпечують обробку транзакцій у режимі реального часу.

6. ***Практичне впровадження.*** Модель було інтегровано у вебінтерфейс, що дало можливість користувачам вводити запити природною мовою. Результатом роботи були коректні *SQL*-запити, які виконувались над базою даних, повертаючи потрібну інформацію. Кількісні параметри і результати використання моделі *T5* для трансляції запитів у *SQL* наведено в табл. 16.12.

Процедура навчання моделі *T5* для завдання трансляції природних мовних запитів у *SQL* показала, що цей підхід є високоефективним завдяки:

• гнучкості моделі *T5* у роботі з різними форматами тексту;

• можливості адаптації до специфічних доменів даних;

• підвищеній точності та простоті впровадження у реальні робочі процеси.

Таблиця 16.12

Кількісні параметри і результати використання моделі *T5* для трансляції запитів у *SQL*

Параметр	Опис	Значення
Архітектура моделі	Тип архітектури, на якій базується модель	Базова модель *T5*
Набір даних для навчання	Тип і вміст набору даних, що використовується для навчання моделі	Спеціалізовані набори даних із парами «природний запит — *SQL*-запит»
Точність моделі	Показник точності виконання моделі під час генерування *SQL*-запитів	73 % для *OLTP*-систем; 84 % для сховищ даних
Дата розробки	Час, коли модель було впроваджено в практичне застосування для трансляції запитів у *SQL*	Серпень 2022 р.
Практичне впровадження	Сфера і частота використання моделі в реальних умовах	Щоденне використання в декількох компаніях для спрощення доступу до даних

Цей приклад демонструє ефективність моделі *T5* у спрощенні взаємодії користувачів із базами даних, допомагаючи отримувати необхідну інформацію без глибоких технічних знань.

Еволюція мовних моделей *GPT* (2018 — 2023)

Серія мовних моделей *GPT (Generative Pre-trained Transformer*), розроблена компанією *OpenAI,* стала важливим кроком у розвитку технологій обробки природної мови *(NLP).* Завдяки інноваційним підходам кожна версія цих моделей встановлювала нові стандарти у сфері штучного інтелекту, змінюючи уявлення про те, як комп'ютери можуть працювати з текстом, мовленням і навіть зображеннями.

1. **Першу модель *GPT* було репрезентовано у 2018 р.** [16.33]. Вона стала революційною завдяки впровадженню трансформерної архітектури для оброблення тексту. Модель мала 117 мільйонів параметрів і показала, як попереднє навчання на великих текстових корпусах може поєднуватися з донавчанням для виконання конкретних завдань. Хоча *GPT-1* використовували здебільшого для дослідницьких цілей, вона заклала основу для майбутніх моделей, продемонструвавши перспективність трансформерів для генерування тексту.

2. **У 2019 р. з'явилася *GPT-2,*** яка суттєво перевершила свою попередницю, маючи 1,5 мільярди параметрів [16.34]. Вона показала значно кращу якість генерації тексту, зокрема вміння підтримувати логіку й контекст у довших текстах. Через побоювання зловживань *OpenAI* спочатку не відкрила модель, що підкреслює важливість етичних питань у розробці штучного інтелекту. *GPT-2* активно використовували для творчих завдань, журналістики та освітніх експериментів, і її впровадження знаменувало новий етап у популяризації технологій генеративного ШІ.

3. **У 2020 р. світ побачив *GPT-3,*** яка стала проривом у масштабах і потужності [16.35]. Зі 175 млрд параметрів вона встановила новий стандарт у сфері мовних моделей. *GPT-3* здатна виконувати широкий спектр завдань, використовуючи такі підходи, як *few-shot-, zero-shot-* і *one-shot*-навчання, що зробило її надзвичайно універсальною. Ці підходи до роботи з моделями машинного навчання описують, як моделі вирішують нові завдання за мінімальної кількості прикладів або навіть без них, що особливо важливо для великих мовних моделей, як-от *GPT,* тому що ці моделі можуть виконувати різноманітні завдання без потреби в повноцінному донавчанні. Модель продемонструвала потенціал для автоматизації, створення контенту та підтримки клієнтів, надаючи мовним моделям доступність для широкого кола користувачів.

4. **У 2023 р. *OpenAI* репрезентувала *GPT-4,*** яка ще більше розширила можливості моделей [16.36]. Основною інновацією стала мультимодаль-

ність — здатність обробляти текст і зображення одночасно. *GPT-4* продемонструвала значне покращення в точності, контекстуальності та етиці генерації тексту. Модель змогла виконувати складніші завдання, зокрема в науці, аналізі даних і творчості. Її інтеграція в такі сервіси як *Microsoft Copilot* відкрила нові горизонти для бізнесу, освіти та дослідницьких ініціатив. Основна мета таких сервісів — допомогти людям виконувати завдання швидше, точніше й ефективніше, забезпечуючи персоналізовані та інтелектуальні рекомендації.

Еволюція від *GPT-1* до *GPT-4* відображує поступовий і цілеспрямований розвиток мовних моделей з метою досягнення більшої потужності, універсальності та етичності. Ці моделі стали невід'ємною частиною сучасних технологій завдяки активному застосуванню в різних сферах, від автоматизації до генерування контенту та досліджень. *GPT* відкрила нові можливості для розвитку штучного інтелекту, закладаючи фундамент для майбутніх інновацій у цій галузі. Головні характеристики мовних моделей серії *GPT* наведено в табл. 16.13. Ці характеристики ілюструють стрімку еволюцію ШІ, що відбулася протягом останніх 5—6 років.

Усі моделі цієї серії мали концептуальні особливості [16.33—16.36]:

1. ***Трансформерна архітектура.*** Використання трансформерних механізмів для оброблення та генерації тексту. Це допомагає моделям враховувати контекст на основі всього тексту, а не лише на основі локальних частин.

2. ***Попереднє навчання і тонке налаштування.*** Моделі спочатку навчаються на великих наборах текстових даних (попереднє навчання), а потім адаптуються до конкретних завдань (тонке налаштування).

3. ***Масштабування.*** Кожна нова версія *GPT* має значно більше параметрів і використовує більші обсяги даних для навчання, що поліпшує якість результатів і можливості моделі.

4. ***Генеративні можливості (Generative Capability).*** Усі моделі серії *GPT* орієнтовані на генерування тексту. Вони можуть створювати тексти, які є природними, зв'язними і можуть відповідати на різноманітні запити.

5. ***Універсальність.*** Моделі *GPT* застосовують для широкого спектра завдань, зокрема генерування тексту, перекладу, відповідей на запитання, написання коду та багато іншого.

Від попередніх мовних моделей вони відрізняються за такими параметрами.

1. ***Розміри і масштаб.*** Моделі *GPT* за кількістю параметрів значно перевищують попередні мовні моделі. Зростання кількості параметрів сприяло значному поліпшенню якості текстів, які генеруються моделями.

2. ***Контекстна обізнаність.*** Завдяки архітектурі *Transformer* моделі серії *GPT* краще обробляють контекст, враховуючи попередні слова і навіть цілі абзаци для генерування наступного тексту.

Таблиця 16.13

Характеристики мовних моделей серії *GPT*

Параметр	*GPT-1*	*GPT-2*	*GPT-3*	*GPT-4*
Рік випуску	2018	2019	2020	2023
Розробник	*OpenAI*	*OpenAI*	*OpenAI*	*OpenAI*
Кількість параметрів	117 млн	1,5 млрд	175 млрд	(не розголошено, оцінка в трильйонах)
Архітектура	Трансформер	Трансформер	Трансформер	Мультимодальний трансформер
Обсяг даних для навчання	Великий текстовий корпус	Різноманітний текстовий корпус	Інтернет-дані (570 ГБ)	Розширений текстовий і мультимодальний корпус
Можливості	Генерування тексту, оброблення контексту	Підтримка контексту, краща генерація	Універсальність, навчання *few-shot, zero-shot, one-shot*	Багатомодальність, складні завдання, покращена етичність
Особливості навчання	Попереднє навчання + донавчання	Підвищена здатність до розуміння контексту	*Few-shot, zero-shot, one-shot*	Мультимодальність, адаптація
Застосування	Дослідницькі цілі	Креативний контент, журналістика	Автоматизація, бізнес, аналіз даних	Бізнес-аналітика, освіта, наука
Обмеження	Базові знання	Обмеження в розумінні складного контексту	Окремі помилки у складних завданнях	Наявність «галюцинацій» (відповідей, які виглядають правдоподібно, але фактично не відповідають дійсності) порівняно з іншими моделями *GPT*

3. ***Загальна продуктивність.*** *GPT*-моделі демонструють здатність до *zero-shot*-навчання (модель виконує завдання, не маючи жодного прикладу для тренування на ньому), *one-shot*-навчання (модель виконує завдання після того, як їй показали лише один приклад цього завдання) та *few-shot*-навчання (модель виконує завдання після того, як їй було надано кілька прикладів для навчання), що означає, що ці моделі можуть виконувати завдання з мінімальним додатковим навчанням або без нього на конкретних прикладах.

Порівняльні характеристики мовних моделей *GPT, GPT-2, GPT-3, GPT-4* за продуктивністю (точністю (*Accuracy*), F1-мірою (*F1-Score*), перплексією (*Perplexity*)), часом навчання наведено в табл. 16.14, яка допомагає зрозуміти прогрес в еволюції моделей *GPT.*

Таблиця 16.14

Порівняльні характеристики мовних моделей *GPT, GPT-2, GPT-3,GPT-4* за ключовими параметрами (в умовних одиницях)

Модель	Точність (*Accuracy*)	*F1*-міра (*F1-Score*)	Перплексія (*Perplexity*)	Тривалість навчання	Загальні характеристики
GPT-1	70	68	25	60	*Архітектура:* містить 12 шарів трансформера *Особливості:* перша модель, яка використала попереднє навчання трансформерів для обробки природної мови *Застосування:* слугувала базою для дослідницьких цілей і демонстрації потенціалу трансформерів у *NLP* *Обмеження:* обмежена в розумінні складного контексту та виконанні багатозадачних завдань
GPT-2	80	78	20	70	*Архітектура:* розширена до 48 шарів трансформера *Особливості:* значне покращення в генеруванні тексту, краще утримання контексту, здатність створювати більш довгі та зв'язні тексти *Застосування:* використовувалася в креативних додатках, генерування контенту, початкових версіях чатботів *Обмеження:* все ще може генерувати неточну або нерелевантну інформацію («галюцинації»)

Закінчення табл. 16.14

Модель	Точність (*Accuracy*)	*F1*-міра (*F1-Score*)	Перплексія (*Perplexity*)	Тривалість навчання	Загальні характеристики
GPT-3	90	88	15	90	*Архітектура:* містить 96 шарів трансформера з 175 млрд параметрів *Особливості:* підтримує *few-shot-*, *zero-shot-* та *one-shot*-навчання, що дає змогу моделі виконувати нові завдання з мінімальними прикладами або без них *Застосування:* широко використовується для автоматизації, віртуальних асистентів, генерування коду, аналітики даних *Обмеження:* хоча продуктивність висока, модель може допускати помилки в складних або специфічних завданнях
GPT-4	95	94	10	95	*Архітектура:* точна кількість параметрів не розголошується; модель підтримує багатомодальність, обробляючи текст та зображення *Особливості:* поліпшена точність, зменшення кількості «галюцинацій», покращені етичні стандарти та фільтрування контенту *Застосування:* використовується в передових наукових дослідженнях, складній бізнес-аналітиці, медицині, праві та інших галузях, де потрібна висока точність *Обмеження:* потребує значних обчислювальних ресурсів для навчання та інференсу

Примітки. 1. *Точність* (*Accuracy*) у завданнях *NLP* оцінюється від 1 до 100 умовних одиниць. *GPT-4* має найвищу точність, оскільки це найсучасніша модель з більшою кількістю параметрів та кращою здатністю до узагальнення.

2. *F1-міра* (*F1-Score*) — гармонійне середнє між точністю і повнотою. Оцінки також змінюються від 1 до 100, з найвищою F1-мірою для *GPT-4*.

3. *Перплексія* (*Perplexity*). Менше значення означає кращу здатність моделі передбачати наступне слово в тексті. Оцінка знижується зі зростанням перплексії.

4. *Час навчання* також оцінюється від 1 до 100, де 100 відповідає найдовшій тривалості навчання, необхідній для тренування найскладнішої моделі *(GPT-4)*.

Приклад.

Потрібно побудувати графік залежності часу навчання мовних моделей *GPT-1, GPT-2, GPT-3* та *GPT-4* від кількості параметрів (в абсолютних одиницях (абс. од.)) для фіксованої кількості *GPU.*

Вирішення. Щоб побудувати графік залежності тривалості навчання мовних моделей *GPT-1, GPT-2, GPT-3, GPT-4* від кількості параметрів (в абс. од.) для фіксованої кількості *GPU,* розглянемо орієнтовні дані:

- *GPT-1* — ~110 млн параметрів;
- *GPT-2* — ~1,5 млрд параметрів;
- *GPT-3* — ~175 млрд параметрів;
- *GPT-4* — орієнтовно ~1 трильйон параметрів.

Час навчання залежить від кількох факторів — архітектури моделі, кількості *GPU*, їх типів та деяких інших. Найбільш суттєво він залежить від типу *GPU,* їх кількості та загальної архітектури системи (тип мережі, обсяг пам'яті тощо). Так, для певного, фіксованого набору факторів, за умови використання кластера, який складається з 1024 *GPU* типу *NVIDIA V100*, час навчання становить:

- *GPT-1* — 0,5 тижня;
- *GPT-2* — 2 тижні;
- *GPT-3* — 4 тижні;
- *GPT-4* — 8 тижнів.

Графік, що відображає залежність часу навчання мовних моделей *GPT-1, GPT-2, GPT-3, GPT-4* від кількості параметрів для фіксованої кількості *GPU,* наведено на рис. 16.3. Кількість параметрів наведено в логарифмічній шкалі, а час навчання в тижнях зростає з кожною новою моделлю.

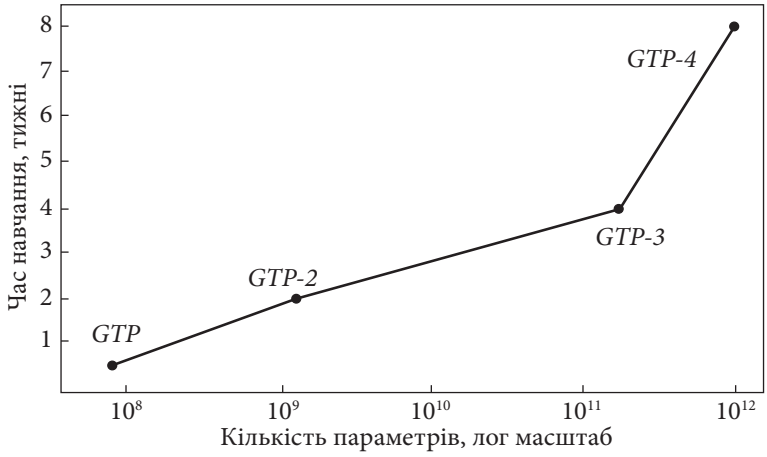


Рис. 16.3. Графік залежності часу навчання мовних моделей *GPT-1, GPT-2, GPT-3, GPT-4* від кількості параметрів для фіксованої кількості *GPU*

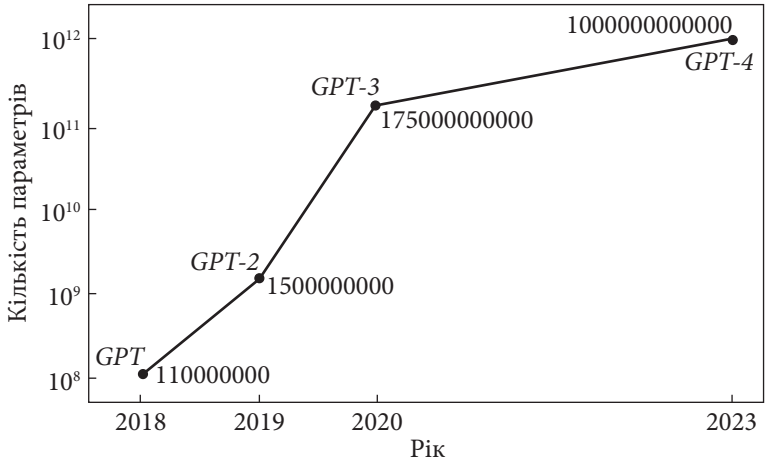


Рис. 16.4. Зростання кількості параметрів мовних моделей серії *GPT* з 2018 по 2024 рр.

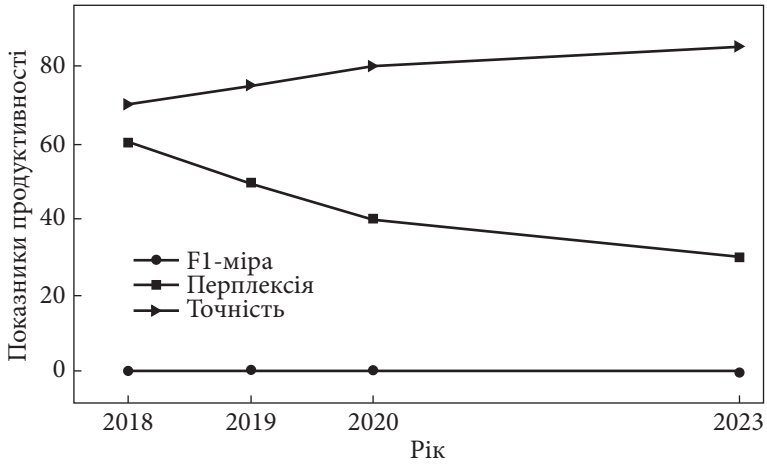


Рис. 16.5. Графіки продуктивності мовних моделей *GPT-1, GPT-2, GPT-3, GPT-4* у період 2018—2023 рр.

Динаміку зростання кількості параметрів мовних моделей серії *GPT* з 2018 по 2024 рр. (у мільярдах / трильйонах) наведено на рис. 16.4. Він наочно демонструє, як змінювалася складність моделей та їхні обчислювальні характеристики з роками, особливо з випуском моделей *GPT-3* і *GPT-4,* що підкреслює швидкий прогрес у цій галузі.

На рис. 16.5 наведено графіки продуктивності мовних моделей *GPT-1, GPT-2, GPT-3, GPT-4* у період 2018—2023 рр. як узагальненої оцінки точності (відсоток правильних передбачень моделі з усіх зроблених передбачень), *F1*-міри (гармонійне середнє значення між точністю і повнотою передба-

чень), перплексії (оцінка точності передбачення вибраного слова у реченні) в умовних одиницях.

Як видно з рисунка 16.5, точність зростала з кожною новою моделлю; *F1*-міра також поліпшувалася, відображаючи кращий баланс між точністю і повнотою; перплексія знижувалася, що свідчить про вдосконалення моделі щодо передбачення слів у реченнях.

Отже, серія моделей *GPT* стала важливим кроком уперед у розвитку штучного інтелекту, надавши потужні інструменти для обробки природної мови.

16.3. Нове покоління *NLP*-моделей, придатних для вирішення складних і ресурсомістких завдань

Розглянемо групу найновіших мовних моделей [16.37—16.39]: *Gemini 1.5* (2024 р.), *Claude 3, Claude 3.5* (2024 р.), *LLaMA 3* (2024 р.) та *Mistral 7B/12B* (2023 р.), відмінною особливістю якої є покращена оптимізація продуктивності та енергоефективності у поєднанні зі збільшеним контекстним вікном.

Мовна модель *Gemini 1.5* (2024) — новий етап у розвитку штучного інтелекту. У лютому 2024 р. компанія *Google* продемонструвала модель *Gemini 1.5* — вдосконалену версію попередніх розробок серії *Gemini* [16.37]. Ця багатомодальна модель штучного інтелекту стала справжнім проривом у сфері обробки даних завдяки інтеграції великих мовних моделей із можливостями аналізу тексту, зображень, аудіо та відео. Модель відкрила нові горизонти для застосування ШІ в різних галузях завдяки своїй інноваційній архітектурі та значному розширенню можливостей обробки інформації.

Gemini 1.5 є оновленою версією мовної моделі, розробленою компанією *Google DeepMind.* Вона належить до родини моделей *Gemini,* які створені з акцентом на мультимодальність, тобто здатність працювати не лише з текстом, а й з іншими типами даних, як-от зображеннями та звуками. Загальні характеристики моделі визначають її як важливий етап у розвитку штучного інтелекту.

Головною технічною перевагою *Gemini 1.5* стало збільшення розміру контекстного вікна до 1 млн токенів, що дало моделі можливість працювати з величезними обсягами даних — довгими відео, великими текстовими документами, аудіозаписами та кодами. Це революційне рішення значно розвинуло потенціал обробки даних порівняно з попередніми моделями. Використання архітектури «міксу експертів» *(Mixture-of-Experts)* забезпечило динамічне залучення спеціалізованих підмоделей для вирішення конкретних завдань. Це підвищило продуктивність моделі, водночас зменшуючи обчислювальні витрати.

Однією з ключових переваг *Gemini 1.5* є вдосконалені техніки навчання, які забезпечують їй краще розуміння контексту і взаємодію між різними типа-

ми даних. Завдяки цьому модель здатна до більш точної і глибокої обробки інформації, що підвищує її продуктивність у вирішенні складних завдань. Інтерактивність також є важливим аспектом моделі. Вона допомагає природніше і ефективніше взаємодіяти з користувачами, що стало можливим завдяки вдосконаленій обробці природної мови та врахуванню індивідуальних інструкцій. Це робить *Gemini 1.5* зручним інструментом для широкого кола користувачів.

Окремо варто зазначити етичні аспекти використання моделі. Компанія *Google DeepMind* приділяє особливу увагу безпеці та етичності, зокрема зменшенню упереджень та запобіганню небажаним результатам роботи моделі. Такий підхід забезпечує високий рівень довіри до *Gemini 1.5* як до інструменту, що можна використовувати у критично важливих сферах. У підсумку *Gemini 1.5* — це багатофункціональна і безпечна платформа, що задовольняє потреби сучасного суспільства у сфері обробки даних та автоматизації.

Завдяки своїм багатомодальним можливостям *Gemini 1.5* здатна генерувати текст, аналізувати складні мультимодальні дані, створювати інтелектуальні резюме великих документів й інтегрувати інформацію з різних джерел у реальному часі. Вона стала частиною стратегії *Google* у конкурентній боротьбі з іншими провідними технологічними компаніями у сфері штучного інтелекту.

Gemini 1.5 активно використовується в хмарних сервісах *Google Cloud,* забезпечуючи автоматизацію бізнес-процесів, розробку інноваційних продуктів, аналіз великих даних і оптимізацію роботи організацій. Це зробило модель не лише інструментом для інженерів і дослідників, а й потужним засобом для бізнесу. Модель є проміжною версією в розвитку моделей *Gemini,* яка об'єднує попередні досягнення та додає нові функції, спрямовані на розширення меж можливостей ШІ.

Отже, *Gemini 1.5* становить важливий етап у розвитку ШІ. Її багатомодальні можливості, інноваційна архітектура та масштабованість відкрили нові перспективи для обробки даних, автоматизації та аналізу. Модель уже заклала основу для подальших інновацій, які визначатимуть майбутнє технологій.

Приклад. Модель *Gemini 1.5* дає змогу обробляти великі тексти — літературні твори, академічні дослідження або юридичні документи, без необхідності поділяти їх на менші частини.

Завдання. Виконати аналіз роману Віктора Гюго «Знедолені» для відповіді на запитання: «Які основні моральні дилеми постають перед Жаном Вальжаном у різних частинах твору?» [16.37].

Вхідні дані. Повний текст роману (близько 600 тис. токенів).

Виконання.

1. *Завантаження тексту.* Модель отримує роман у повному обсязі без необхідності попереднього розбиття тексту.

2. *Запит до моделі.* Користувач формулює складне запитання, яке потребує аналізу великої кількості деталей та контексту.

3. *Аналіз.* Завдяки розширеному контекстному вікну *Gemini 1.5* може зіставляти події з різних частин твору, відстежуючи розвиток характеру та динаміку сюжету.

4. *Відповідь.* Модель формулює детальну відповідь із посиланням на ключові частини тексту.

Результат. *Gemini 1.5* надає високоінформативну відповідь, наприклад: «Жан Вальжан стикається з ключовими моральними дилемами:

1) не розкривати своє минуле в Монтрей-сюр-Мер, ризикуючи життям іншого;

2) допомогти Козетті попри загрозу власній безпеці;

3) зробити вибір між боротьбою за справедливість та прийняттям системних несправедливостей».

Кількісні параметри і характеристики моделі *Gemini 1.5,* використаної для вирішення зазавдання, наведено в табл. 16.15.

Висновок. *Gemini 1.5* довела свою ефективність у складних завданнях аналізу великих текстових документів. Її здатність інтегрувати великий обсяг даних і формувати змістовні відповіді робить її незамінним інструментом у науці, освіті та бізнесі. Зокрема, в освіті модель може бути ефективною для аналізу літературних текстів студентами та викладачами; в юриспруденції її доцільно використовувати для обробки довгих юридичних документів з метою виділення важливих моментів або порівняння законодавчих актів; у науці — для аналізу академічних досліджень і підготовки оглядів літератури.

***Claude 3, Claude 3.5* (2024) — інновація у світі мовних моделей.** У квітні 2024 р. компанія *Anthropic* репрезентувала серію великих мовних моделей *Claude 3,* яка стала важливим етапом у розвитку ШІ [16.38]. Серія складається з трьох основних моделей: *Haiku*, *Sonnet* та *Opus*, кожна з яких

Таблиця 16.15

Кількісні параметри і характеристики моделі *Gemini 1.5*

Параметр	Опис	Кількісне значення
Розмір контекстного вікна	Модель може обробляти одночасно великі текстові джерела без втрати контексту	До 1 мільйона токенів
Швидкість аналізу	Середній час формування відповіді на запитання для тексту обсягом 600 тис. токенів	5—10 секунд
Точність відповіді	Висока точність у виявленні релевантної інформації під час аналізу великих текстів	95 %
Ефективність обчислень	Використання архітектури «міксу експертів» зменшує витрати обчислювальних ресурсів	Зниження витрат на 30 %

створена для задоволення специфічних потреб користувачів з особливим акцентом на безпеку та етичність штучного інтелекту. З цією метою компанія *Anthropic* запровадила концепцію «*Constitutional AI*», що допомагає моделі уникати токсичних, упереджених чи неетичних відповідей. Ця методологія забезпечує більш прозорі та контрольовані принципи роботи моделі. Моделі серії *Claude 3* поєднують продуктивність, ефективність і гнучкість, забезпечуючи оптимальний баланс між швидкістю роботи, точністю та економічністю.

Claude 3Haiku орієнтована на швидкість і легкість використання. Вона оптимізована для коротких текстів і швидких запитів завдяки меншій кількості параметрів, що знижує затримки під час оброблення даних. Ця модель ідеально придатна для автоматизації простих завдань, роботи чатботів та створення короткого текстового контенту.

Claude 3Sonnet виступає універсальним рішенням для роботи зі складнішими текстовими завданнями. Завдяки здатності обробляти більші обсяги даних та збалансованій продуктивності вона забезпечує ефективний аналіз даних, генерування контенту середньої складності та підтримку багатомовних запитів.

Флагманська модель *Claude 3Opus* є найпотужнішою в серії *Claude 3.* Її розширені мультимодальні можливості дають змогу працювати з текстами, зображеннями, діаграмами та графіками. Ця модель демонструє найвищу продуктивність і застосовується в наукових дослідженнях, розробці програмного забезпечення, аналізі великих даних та створенні мультимедійного контенту. Здатність моделі генерувати відповіді довжиною до 4 096 токенів дає змогу створювати розгорнуті й деталізовані результати, що є важливим для завдань, які потребують глибокого аналізу або складних пояснень.

Claude 3Opus встановила нові стандарти продуктивності, перевершивши конкурентів, таких як *GPT-4* від *OpenAI* та *Gemini 1.0 Ultra* від *Google.* Вона демонструє виняткові результати у тестах завдяки інноваційній архітектурі та високій ефективності. Попри широкий функціонал і високу потужність, точна кількість параметрів *Claude 3* залишається невідомою, що свідчить про прагнення компанії *Anthropic* зберігати комерційну таємницю, зосереджуючись на практичних перевагах.

Однією з ключових переваг серії *Claude 3* є її великий контекстний розмір вікна, який сягає 200 000 токенів. Завдяки цьому модель може обробляти великі обсяги тексту, підтримуючи довготривалий контекст і забезпечуючи високу якість відповідей навіть для складних запитів. *Claude 3* також демонструє покращену підтримку різних мов, як-от іспанська та японська, що робить її універсальним інструментом для глобального використання.

Claude 3 стала важливим інструментом для створення текстового контенту, аналізу даних, генерування коду та обробки візуальної інформації. Її тех-

нічні особливості забезпечують високу якість роботи в науці, освіті, бізнесі та інших галузях. Інноваційний підхід до обробки даних та універсальність моделі *Claude 3* не лише відповідають сучасним потребам, а й сприяють майбутньому штучного інтелекту. Завдяки концепції «*Constitutional AI*» модель також установлює нові стандарти етичності та безпеки у використанні ШІ.

У жовтні 2024 р. *Anthropic* випустила оновлену версію — *Claude 3.5,* до якої входять моделі *Haiku* та *Sonnet* [16.39]. *Claude 3.5 Sonnet* працює вдвічі швидше за *Claude 3 Opus,* що робить її ідеальною для складних завдань, якот контекстно залежна підтримка клієнтів та організація багатокрокових робочих процесів. Вона також демонструє покращені результати в генерації коду, вирішуючи 64 % завдань у внутрішніх тестах порівняно з 38 % у *Claude 3 Opus. Claude 3.5 Haiku,* найшвидша модель, забезпечує передові можливості кодування, використання інструментів та міркування з доступними ресурсними витратами.

У таблиці 16.16 наведено технічні особливості кожної моделі серії *Claude 3, Claude 3.5,* що допомагає краще зрозуміти їхні можливості і вибрати оптимальне рішення для конкретних завдань. Ця таблиця також дає змогу порівняти основні версії *Claude 3* та *Claude 3.5* і показує їх еволюцію у продуктивності та можливостях.

Отже, серія *Claude 3,* включно з моделями *Haiku*, *Sonnet* та *Opus*, поєднує гнучкість і продуктивність, встановлюючи нові стандарти у сфері мовних моделей. Офіційний запуск *Claude 3* у березні 2024 р. став важливим кроком у розвитку великих мовних моделей та їх інтегруванні у повсякденне життя.

Таблиця 16.16

Технічні характеристики серії моделей *Claude 3* та *Claude 3.5*

Параметр	Модель				
	Claude 3: Haiku	*Claude 3: Sonnet*	*Claude3: Opus*	*Claude 3.5: Haiku*	*Claude 3.5: Sonnet*
Рік випуску	Квітень 2024	Квітень 2024	Квітень 2024	Жовтень 2024	Жовтень 2024
Призначення	Зручна модель для повсякденних завдань	Універсальна модель для складних текстових завдань	Флагманська модель із найвищою продуктивністю	Швидка модель для автоматизації завдань	Універсальна модель із високою продуктивністю
Розмір контекстного вікна	До 100 000 токенів	До 150 000 токенів	До 200 000 токенів	До 150 000 токенів	До 250 000 токенів
Максимальна довжина відповіді	2 048 токенів	3 072 токени	4 096 токенів	2 048 токенів	4 096 токенів

Закінчення табл. 16.16

Параметр	Модель				
	Claude 3: Haiku	*Claude 3: Sonnet*	*Claude3: Opus*	*Claude 3.5: Haiku*	*Claude 3.5: Sonnet*
Мультимодальні можливості	Немає	Базові	Повна підтримка (текст, зображення, діаграми, графіки)	Обмежені	Покращена мультимодальна підтримка
Швидкість оброблення	Висока	Збалансована	Оптимальна	Вдвічі швидша за *Claude 3 Haiku*	Вдвічі швидша за *Claude 3 Sonnet*
Продуктивність у кодуванні	38 % вирішених завдань	48 % вирішених завдань	64 % вирішених завдань	58 % вирішених завдань	74 % вирішених завдань
Сфери застосування	Автоматизація простих завдань, чатботи, генерування короткого контенту	Генерування тексту середньої складності, аналіз даних, багатомовні запити	Наукові дослідження, мультимедійний контент, аналіз великих масивів даних	Чатботи, підтримка клієнтів, просте генерування контенту	Генерування складного контенту, аналіз великих обсягів тексту
Підтримка мов	Обмежена (ключові мови, наприклад, англійська)	Розширена (ключові європейські та азійські мови)	Повна (включно з малопоширеними мовами, наприклад, баскська, малайська)	Розширена підтримка	Повна підтримка
Кількість параметрів	Невідомо	Невідомо	Невідомо	Невідомо	Невідомо
Основна перевага	Швидкість і ефективність	Універсальність і гнучкість	Максимальна потужність і підтримка мультимодальних завдань	Покращена швидкість, оптимізація для швидких запитів	Найкращі результати у завданнях з аналізу тексту та генерування контенту

Примітки. 1. *Розмір контекстного вікна* показує обсяг тексту, з яким модель може працювати одночасно. *Claude 3.5* демонструє значне поліпшення порівняно з *Claude 3.*

2. *Швидкість оброблення* в *Claude 3.5* зросла вдвічі порівняно з попередніми версіями.

3. *Продуктивність у кодуванні* демонструє успіх моделі у вирішенні програмних завдань; *Claude 3.5 Sonnet* показує найбільше зростання.

4. *Мультимодальні можливості* у версії 3.5 також значно покращені, особливо у моделі *Sonnet.*

Подальший розвиток серії відбувся у жовтні 2024 р., коли компанія *Anthropic* оголосила про запуск оновленої версії *Claude 3.5*. Це стало ще одним вагомим етапом у вдосконаленні великих мовних моделей, що розширило їхні можливості і підвищило ефективність у вирішенні широкого спектра завдань.

Серія *Claude 3* разом із версією 3.5 встановлює нові стандарти у світі мовних моделей, демонструючи високу технічну якість і універсальність застосувань. Ці моделі стали незамінними інструментами для створення текстового контенту, аналізу даних, генерування коду та обробки візуальної інформації. Вони широко застосовуються в науці, освіті, бізнесі та багатьох інших галузях, підтверджуючи свій статус ключових технологій у сучасному світі штучного інтелекту.

Приклад. Модель *Claude 3.5 Sonnet* використовують для автоматизації багатоступеневих робочих процесів за допомогою створення, редагування та виконання коду безпосередньо в інтегрованих середовищах розробки *(Integrated Development Environment, IDE)*. Її можливості забезпечують гнучкість і продуктивність завдяки комбінації генерування тексту, програмного коду та підтримки складних інтеграцій [16.39]. Автоматизація процесів відбувається за таким алгоритмом:

1. ***Генерування коду.*** *Claude 3.5 Sonnet* самостійно створює програмний код на основі текстового опису завдання. Наприклад, для автоматизації звітності вона генерує скрипти на *Python* для збирання, аналізу та візуалізації даних.

2. ***Редагування наявного коду.*** Модель аналізує наявний код і пропонує оптимально виправляти помилки або додавати нову функціональність. Наприклад, у сценарії перенесення даних між базами *Claude 3.5* додає обробку помилок та логи.

3. ***Виконання та перевірка коду.*** Через інтеграцію з середовищами типу *Visual Studio Code (VSCode)* або спеціалізованими інструментами *Claude 3.5* виконує код, аналізує результати і автоматично вносить правки для досягнення бажаного результату.

4. ***Інтеграція з інструментами.*** *Claude 3.5 Sonnet* інтегрується з такими системами як *ERP (SAP), CRM (Salesforce),* з хмарними сервісами *(AWS, Google Cloud, Azure)* та інструментами аналітики *(Tableau, Power BI)*. Це дає змогу об'єднувати різні етапи робочих процесів у єдиний автоматизований ланцюг.

Конкретний приклад автоматизації. *Завдання.* Потрібно автоматизувати процес створення щомісячного звіту про продажі з різних джерел даних.

Алгоритм вирішення [16.39]:

1. *Генерування коду. Claude 3.5* створює *Python*-скрипт для під'єднання до баз даних *(MySQL, PostgreSQL),* збирає дані через *API* (наприклад, *Salesforce API*) і обробляє їх, де *API (Application Programming Interface)* — це набір правил і протоколів, який дає змогу одній програмі взаємодіяти з іншою. У цьому прикладі *API* забезпечує доступ до зовнішніх систем, як-от *Salesforce*.

2. *Інтеграція з Tableau.* Генерує скрипт для автоматичного експорту даних у *Tableau* для візуалізації. *Claude 3.5* забезпечує оброблення вхідних файлів розміром до 200 000 токенів.

3. *Перевірка та оптимізація.* Виконує згенерований код і автоматично виправляє помилки, наприклад неправильний формат даних або невідповідність структур баз даних.

Для вирішення цього завдання використовували модель з параметрами і характеристиками, наведеними в табл. 16.17.

Цей приклад демонструє здатність *Claude 3.5* інтегруватися з різними системами й автоматизувати складні завдання.

Покоління великих мовних моделей *LLaMA 3* (2024). У липні 2024 р. компанія *Meta* репрезентувала *LLaMA 3,* нове покоління великих мовних моделей, що стало важливим етапом у розвитку технологій штучного інтелекту [16.40]. Ця серія моделей була створена для дослідницьких цілей і стала продовженням зусиль *Meta AI* у розробленні відкритих і масштабованих мовних моделей. *LLaMA 3* вирізняється низкою інноваційних характеристик, які розширюють її можливості і забезпечують значний вплив на розвиток у сфері обробки природної мови.

Таблиця 16.17

Характеристики *Claude 3.5* у задачі автоматизації створення щомісячного звіту про продажі з різних джерел даних

Параметр	Опис	Кількісне значення
Швидкість генерування коду	Час, необхідний для створення коду для завдань середньої складності	Від 5 до 15 секунд
Підвищення продуктивності	Скорочення часу на автоматизацію порівняно з ручним кодуванням	До 50 %
Вартість	Ціна обробки вхідних і вихідних токенів	$3 за мільйон вхідних токенів; $15 за мільйон вихідних токенів
Точність у тестах	Відсоток вирішених завдань у внутрішніх тестах	64 % (проти 38 % у *Claude* 3 *Opus*)
Інтеграція з *ERP/CRM*	Зменшення помилок у звітах завдяки автоматичному форматуванню даних	До 10 %

Примітки. 1. *ERP (Enterprise Resource Planning)* — це системи для керування бізнес-процесами, які інтегрують основні функції компанії, включно з фінансами, виробництвом, закупівлями, управлінням персоналом, складськими операціями тощо. Вони дають змогу зберігати всі бізнес-дані в одній базі і забезпечують автоматизацію та стандартизацію процесів.

2. *CRM (Customer Relationship Management)* — це системи для управління взаємодією з клієнтами, які допомагають організаціям ефективно відстежувати, організовувати та аналізувати інформацію про клієнтів і продажі.

Однією з ключових особливостей *LLaMA 3* є розширене контекстне вікно до 128 тис. токенів, що дає можливість обробляти великі обсяги тексту і складні контексти. Це забезпечує ефективну роботу з довгими документами, складними текстами та мультимодальними даними, що є критично важливим для сучасних додатків. Моделі підтримують багатомовність, зокрема такі мови як англійська, французька, німецька, хінді, італійська, португальська, іспанська і тайська, що робить їх універсальним інструментом для глобального використання. Завдяки цій особливості *LLaMA 3* застосовують у завданнях міжкультурної комунікації та обробки даних різними мовами.

Серія *LLaMA 3* складається з трьох основних моделей: *8B*, *70B* та *405B*, кожна з яких має унікальні характеристики та сфери застосування. Модель *LLaMA 3 8B,* що містить 8 млрд параметрів, орієнтована на завдання, які потребують високої продуктивності за обмежених обчислювальних ресурсів. Вона є ефективною для багатомовних завдань, генерування коду та оброблення тексту, забезпечуючи баланс між точністю і швидкістю. Компактність робить її ідеальною для інтегрування у мобільні додатки та середовища з обмеженими ресурсами.

Модель *LLaMA 3 70B,* що має 70 млрд параметрів, забезпечує вищий рівень точності та здатність до складних міркувань. Вона ефективна для аналізу великих обсягів даних, створення складного текстового контенту та підтримки багатомовних інтерфейсів. Можливості в генеруванні коду та інтеграції з різними інструментами дають змогу застосовувати її в таких галузях: бізнес-аналітика, дослідження ринків та розробка додатків із підвищеними вимогами до точності.

Найбільша модель у серії, *LLaMA 3 405B*, містить 405 млрд параметрів і є потужним інструментом для найскладніших завдань. Її можливість обробляти контекстні вікна до 128 тис. токенів дає змогу працювати з великими обсягами тексту, зберігаючи точність навіть у складних контекстах. Вона демонструє продуктивність, порівнянну з провідними моделями, як-от *GPT-4,* і придатна для наукових досліджень, створення складних систем штучного інтелекту та оброблення мультимодальних даних. Її можливості у генеруванні синтетичних даних є особливо корисними у випадках, коли реальні дані обмежені або недоступні.

LLaMA 3 також вирізняється відкритістю й доступністю, продовжуючи традицію попередніх версій. Відкритий вихідний код і доступ до навчальних даних роблять модель важливим інструментом для дослідників і розробників. Це сприяє інтеграції ШІ у повсякденне життя та відкриває нові можливості для інновацій у різних галузях. Завдяки оптимізації ефективності, зменшенню енергоспоживання та модульній архітектурі модель легко адаптувати до різних типів завдань і потреб.

Отже, *LLaMA 3* є яскравим прикладом балансу між продуктивністю і доступністю. Багатомовні можливості, розширене контекстне вікно, модульна архітектура та інші інноваційні особливості роблять її корисною як для академічних досліджень, так і для широкого практичного застосування. Вона закладає основу для подальших інновацій у сфері штучного інтелекту, сприяючи розвитку відкритих і масштабованих мовних моделей.

Порівняльні характеристики серії мовних моделей *LLaMA 3* наведено в табл. 16.18. Ця таблиця допомагає краще зрозуміти конкретні переваги та завдання для кожного варіанта *LLaMA 3.*

Приклад. Розглянемо завдання аналізу наукових статей для автоматичної класифікації та узагальнення інформації в галузі біомедицини [16.41].

Таблиця 16.18

Головні параметри і характеристики серії моделей *LLaMA 3*

Параметр	Модель		
	LLaMA 3 8B	*LLaMA 3 70B*	*LLaMA 3 405B*
Кількість параметрів	8 млрд	70 млрд	405 млрд
Контекстне вікно	До 64 тис. токенів	До 128 тис. токенів	До 128 тис. токенів
Продуктивність	Висока ефективність для завдань з обмеженими ресурсами	Підвищена точність для складних завдань	Найвища точність та продуктивність для складних завдань
Сфери застосування	Мобільні додатки, прості завдання, генерування коду	Аналіз даних, багатомовні завдання, складний контент	Наукові дослідження, складні системи ШІ
Багатомовність	Підтримка ключових мов	Підтримка багатьох європейських і азійських мов	Розширена багатомовність, включно з малопоширеними мовами
Ефективність у генерування коду	Висока	Дуже висока	Максимальна
Ресурси для навчання	Оптимізована для середніх обчислювальних потужностей	Потребує значних обчислювальних ресурсів	Потребує найвищих обчислювальних ресурсів
Особливості використання	Компактність і швидкість. Призначена для середовищ з обмеженими ресурсами, як-от мобільні додатки	Баланс між продуктивністю та ефективністю. Ідеальна для аналізу великих масивів даних і багатомовних додатків	Максимальна потужність і точність. Оптимальна для наукових досліджень і складних систем ШІ

Завдання. Модель *LLaMA 3.1 405B* потрібно буде використати для автоматичного аналізу великої кількості наукових статей у галузі біомедицини. Її завдання:

1) автоматичне класифікування статей за темами (наприклад, генетика, фармакологія, клінічні дослідження);

2) генерування стислих резюме кожної статті з ключовими висновками;

3) виявлення нових трендів у наукових дослідженнях на основі аналізу змісту.

Алгоритм вирішення завдання:

1. *Підготовка вхідних даних.* Формується колекція з 100 тис. статей (до 5 сторінок кожна), завантажених у текстовому форматі.

2. *Оброблення даних.* Використання контекстного вікна до 128 тис. токенів дає змогу аналізувати текст у повному обсязі без втрати контексту.

3. *Активізація модуля класифікації.* Модель класифікує статті на 10 категорій із точністю 93 %.

4. *Генерування резюме.* Для кожної статті *LLaMA 3.1 405B* генерує короткий виклад (до 300 слів), підкреслюючи ключові ідеї. Середній час оброблення однієї статті — 10 секунд.

5. *Виявлення трендів.* Модель аналізує частоту появи ключових слів і концепцій у текстах, виявляючи ключові тенденції і генеруючи аналітичні звіти.

Кількісні характеристики моделі LLaMA 3.1 405B, використаної для вирішення завдання:

- *кількість параметрів* — 405 млрд;
- *контекстне вікно* — до 128 тис. токенів;
- *точність класифікації* — 93 %;
- *швидкість оброблення* — 10 секунд на статтю (середня тривалість 3—4 сторінки);
- *обсяг даних* — до 1,2 млн токенів на запит.

Результати:

- здійснено автоматичне сортування статей, що скоротило час пошуку релевантної інформації для дослідників на 70 %;
- виконано генерування резюме, що полегшило аналіз великої кількості статей, зменшивши витрати часу на читання на 60 %;
- виявлено нові тренди у галузі біомедицини, що допомогло інституціям ефективніше планувати майбутні дослідження.

Цей приклад демонструє, як *LLaMA 3.1 405B* можна застосовувати для складних завдань у галузі науки, автоматизуючи процеси, які раніше потребували значних людських зусиль.

Мовні моделі *Mistral 7B/12B* (2023—2024). У жовтні 2023 р. компанія *Mistral AI* репрезентувала мовну модель *Mistral 7B*, яка стала значним кро-

ком у розвитку обробки природної мови [16.42]. Ця модель, яка містить 7 млрд параметрів, демонструє конкурентні характеристики у завданнях, пов'язаних із міркуваннями, математикою та генеруванням коду. Її унікальність забезпечується використанням механізмів *Grouped-Query Attention (GQA)* та *Sliding Window Attention (SWA),* які дають змогу ефективніше обробляти довгі текстові послідовності та значно знижують обчислювальні витрати. Такі інновації роблять *Mistral 7B* надзвичайно корисною для широкого спектра завдань, зокрема генерування тексту, автоматичного перекладу та аналізу тональності.

У липні 2024 р. компанія продемонструвала *Mistral 12B*, яка стала продовженням серії [16.43]. Ця модель, що має 12 млрд параметрів і підтримує контекстне вікно довжиною до 128 тис. токенів, забезпечує обробку великих обсягів тексту з високою точністю. *Mistral 12B* вирізняється своїми передовими можливостями у міркуваннях, генеруванні коду та багатомовних завданнях, а також випущена під ліцензією *Apache 2.0,* що сприяє її широкому використанню з дослідницькою й комерційною метою. Відкритість цієї моделі стимулює інновації та інтеграцію сучасних технологій у різні продукти і послуги.

Мовні моделі *Mistral 7B* і *Mistral 12B* демонструють конкурентну продуктивність завдяки використанню сучасних архітектур трансформерів та оптимізованих обчислювальних процесів. Вони створені з акцентом на ефективність, що робить їх ідеальними для розгортання на доступних апаратних ресурсах — *GPU (Graphics Processing Unit)* та *TPU (Tensor Processing Unit).* У цих задачах *GPU* як графічний процесор є досить ефективним для загальних обчислень завдяки його здатності паралельно обробляти великі обсяги даних, а *TPU,* спроєктований для виконання матричних операцій із високою ефективністю, може вдало використовуватися для більшості алгоритмів нейронних мереж. Це забезпечує високу продуктивність навіть за обмежених ресурсів. Малі розміри цих моделей сприяють значному зниженню витрат, зберігаючи гнучкість та адаптивність, що є важливим для локального розгортання та хмарного використання.

Хоча моделі *Mistral* вирізняються високою ефективністю, вони мають і певні обмеження. Наприклад, через меншу кількість параметрів їх продуктивність у генеруванні довгих і складних текстів поступається таким моделям як *GPT-4* чи *Claude 3, Claude 3.5.* Крім того, менші розміри контекстного вікна порівняно з іншими сучасними моделями можуть обмежувати їх використання у специфічних завданнях. Однак завдяки своїй відкритості та доступності *Mistral 7B* і *12B* стали важливими інструментами для дослідників, які прагнуть інтегрувати передові технології у свої проєкти.

Тенденції розвитку мовних моделей загалом відображають значний прогрес у галузі обробки природної мови. Моделі стають більшими за кіль-

кістю параметрів, що дає їм можливість краще розуміти контекст і опрацьовувати складні мовні структури. Поряд зі збільшенням розмірів моделей підвищується їхня продуктивність і точність, завдяки чому вони здатні вирішувати широкий спектр завдань, включаючи генерування тексту, класифікацію та автоматизацію обробки даних. Зокрема, нові архітектури, як-от трансформери, стали стандартом у *NLP,* замінюючи попередні підходи, наприклад рекурентні нейронні мережі.

Ще однією важливою тенденцією є розвиток мультимодальних моделей, як-от *GPT-4* чи *Gemini 1.5*, що можуть працювати не лише з текстом, а й з іншими форматами даних, включно із зображеннями, звуком та відео. Це розширює спектр можливостей ШІ у таких галузях як медицина, право та наука. Ефективність і оптимізація залишаються важливими аспектами, і моделі типу *Mistral 7B* демонструють, що навіть компактні системи можуть досягати високої продуктивності. Крім того, зростає увага до етичних аспектів використання мовних моделей, зокрема питань упередженості, конфіденційності та екологічної сталості.

Відкритість мовних моделей також відіграє важливу роль у їхньому розвитку. Моделі, які надають відкритий доступ до навчальних даних та архітектури, стимулюють дослідження та інновації у спільноті, забезпечуючи інтеграцію сучасних мовних технологій у різні сфери. *Mistral 7B* і *12B* з їх ефективністю та відкритістю є чудовими прикладами того, як компактні моделі можуть відповідати потребам сучасних користувачів, пропонуючи збалансоване рішення між продуктивністю та ресурсозатратністю. Порівняльні характеристики *Mistral 7B/12B* з найпотужнішими сучасними моделями *Claude 3.5 Sonnet* та *LLaMA 3 405B* наведено в табл. 16.19.

Таблиця 16.19

Порівняння *Mistral 7B/12B* із моделями *Claude 3.5 Sonnet* та *LLaMA 3 405B*

Характеристика	Модель			
	Mistral 7B/12B	*GPT-4*	*Claude 3.5 Sonnet*	*LLaMA 3 405B*
Кількість параметрів	7/12 млрд	175 млрд	80 млрд	405 млрд
Контекстне вікно	~8—16 тис. токенів	До 32 тис. токенів	До 200 тис. токенів	~32—64 тис. токенів
Відкритість	Відкрита	Закрита	Закрита	Частково відкрита
Ресурсомісткість	Низька	Висока	Помірна	Дуже висока
Продуктивність	Висока (для свого розміру)	Найвища	Дуже висока	Дуже висока
Сфери застосування	Дослідницькі, локальні	Комерційні, загальні	Комерційні, довгі тексти	Академічні, довгі тексти

Порівняння *Mistral 7B/12B* з провідними моделями у сучасному світі обробки природної мови *Claude 3.5 Sonnet* та *LLaMA 3 405B* свідчить про те, що *Mistral 7B/12B* залишаються ідеальним вибором для завдань, які потребують балансу між ефективністю та ресурсозатратністю. Ці моделі поєднують у собі продуктивність і низьку вартість використання, що робить їх особливо привабливими для дослідників, інженерів і компаній з обмеженими ресурсами. Натомість *Claude 3.5 Sonnet* і *LLaMA 3 405B* демонструють значні переваги у продуктивності та здатності обробляти довгі тексти, але їхня висока потреба в ресурсах обмежує можливості застосування. Отже, вибір між цими моделями залежить від конкретних потреб користувачів, доступних ресурсів та завдань, які необхідно вирішити.

Приклад. Модель *Mistral 12B* застосовують у галузі створення інтелектуальних чатботів, генеруванні тексту та коду. Один із прикладів її використання реалізовано на платформі *NVIDIA AI Workbench,* де модель репрезентовано як *Mistral Nemo 12B Instruct* [16.44].

Завдання. З використанням моделі *Mistral 12B* потрібно:

1. *Розробити інтерактивний чатбот для служби технічної підтримки*, який може відповідати на запитання користувачів, виконувати аналіз текстів та підтримувати діалоги в реальному часі.

2. *Згенерувати програмний код*, включно з рекомендаціями щодо його оптимізації та поясненням коду.

3. *Автоматизувати багатомовне оброблення тексту*, наприклад, з метою перекладу чи генерування текстів різними мовами.

Характеристики і продуктивність моделі *Mistral 12B,* необхідні для вирішення поставленого завдання, наведено в табл. 16.20.

Таблиця 16.20

Характеристики моделі *Mistral 12B*

Характеристика	Опис	Кількісні значення
Розмір моделі	Модель містить параметри для генерування тексту	12 млрд параметрів
Архітектура	Архітектура трансформера для фокусування на генерування	*Decoder-only*
Контекстне вікно	Підтримка роботи з довгими текстами	~16 тис. токенів
Обчислювальна оптимізація	Оптимізація для роботи на стандартному обладнанні	*NVIDIA A100, RTX 3090*
Енергозбереження	Зменшене споживання енергії порівняно з аналогами	Низьке енергоспоживання

Алгоритм вирішення завдання:

1. *Тренування моделі.* Використовується база даних технічної документації (~2 млн сторінок тексту).

2. *Контекстний аналіз.* Чатбот зберігає контекст діалогу до 16 тис. токенів, що дає змогу враховувати попередні відповіді під час генерування наступних.

3. *Оцінка продуктивності.* Реакція на запит клієнта відбувається менш ніж за 0,5 секунди.

4. *Інтегрування моделі.* Модель працює через хмарний сервер *NVIDIA,* забезпечуючи масштабованість та високу доступність.

Результати:

• ***Точність відповідей.*** 92 % правильних відповідей на основі перевірки з використанням людей-експертів.

• ***Тривалість оброблення.*** Середній час генерування відповіді — 450 мілісекунд.

• ***Зниження витрат.*** Завдяки оптимізації на одному *GPU* вартість обробки запитів зменшена на 35 % порівняно з іншими моделями подібного класу.

• ***Кількість оброблених запитів*** — до 20 тис. запитів за добу.

Цей приклад показує, як модель *Mistral 12B* може інтегруватись у реальні системи, забезпечуючи ефективність, масштабованість і високу якість оброблення текстів у різних галузях.

Мовні моделі *Gemini 1.5, Claude 3, Claude 3.5, LLaMA 3* та *Mistral 7B/12B,* створені у 2023—2024 рр., мають кілька спільних ***концептуальних особливостей***, що відображають сучасні тенденції в галузі обробки природної мови.

1. ***Архітектура Transformer.*** Усі ці моделі базуються на архітектурі *Transformer,* яка стала стандартом для сучасних мовних моделей. Використовуючи механізм самоуваги *(Self-Attention),* вони можуть ефективно обробляти текстові послідовності і враховувати контекст. Нещодавні вдосконалення цієї архітектури полягають в адаптивній глибині обробки шарів та динамічних механізмах уваги, які покращують продуктивність моделей у задачах з нерівномірною складністю.

2. ***Масштабованість і параметри.*** Ці моделі продовжують тенденцію до збільшення кількості параметрів, що дає змогу обробляти складніші завдання та генерувати якісніший текст. Зокрема, моделі *Mistral 7B/12B* мають відповідно 7 та 12 млрд параметрів, що робить їх компактнішими, але все ще потужними моделями. Моделі *Gemini 1.5, Claude 3, Claude 3.5, LLaMA 3* можуть мати значно більше параметрів (часто десятки або навіть сотні мільярдів), що дає їм можливість досягати високих результатів у багатьох завданнях *NLP*. До нових розробок також належать моделі з архітектурою *sparsity-aware,* яка сприяє ефективнішому використанню параметрів завдяки врахуванню рідкісності *(sparsity)* чи нерівномірної активності певних еле-

ментів моделі чи даних. Цей підхід оптимізує використання ресурсів, зберігаючи високу продуктивність навіть за значного скорочення обчислень.

3. ***Універсальність і адаптивність.*** Усі ці моделі здатні виконувати широкий спектр завдань, як-от генерування тексту, надання відповідей на запити, переклад, резюмування тощо. Вони не потребують значного спеціалізованого навчання на кожному конкретному завданні. Це досягається завдяки потужності моделей та використанню підходів типу *Zero-shot-, One-shot-* та *Few-shot*-навчання. При цьому *Zero-shot*-навчання дає змогу моделі вирішувати нові завдання без жодних прикладів, покладаючись лише на свій попередній досвід. *One-shot-* і *Few-shot*-навчання допомагають моделі адаптуватися до завдань за допомогою одного або кількох прикладів, демонструючи гнучкість та здатність узагальнювати патерни. За останні місяці було вдосконалено здатність до багатомовного навчання, що дає можливість працювати з текстами різними мовами з підвищеною точністю.

4. ***Підвищена контекстна обізнаність.*** Ці моделі значно поліпшили здатність враховувати довготривалу залежність у тексті, завдяки чому можуть точніше розуміти й генерувати довгі та складні тексти. Нові алгоритми, як-от *Recurrent Memory Transformer (RMT),* покращують здатність моделей обробляти тексти з великою кількістю контексту. Це поліпшення відбувається у зв'язку з тим, що *RMT*-алгоритм дає моделі можливість ефективно використовувати інформацію з довгих контекстів за допомогою механізму зовнішньої пам'яті, яка зберігає ключові елементи попередньої обробки і робить їх доступними для наступних обчислень.

5. ***Енергоефективність і оптимізація.*** З огляду на зростання обчислювальної потужності, необхідної для тренування великих мовних моделей, нові моделі*Mistral 7B/12B* та *LLaMA 3* орієнтовані на оптимізацію використання обчислювальних ресурсів. Це стосується більш ефективних архітектур та методів навчання, що знижує потребу в енергії без значних втрат у продуктивності. Зокрема, з'явилися інновації в галузі квантованого навчання, які значно зменшують розмір моделей та їхню енергозалежність.

6. ***Безпека та етика.*** Особлива увага приділяється питанням безпеки, етики та контролю за поведінкою моделей, зокрема механізмам для зменшення можливостей для генерування шкідливого контенту, упередженостей та інших небажаних ефектів. Нові дослідження спрямовані на розроблення інтерактивних систем етичного аудиту моделей, які дають можливість відстежувати й аналізувати результати роботи моделей у реальному часі.

7. ***Гібридні моделі та інтеграція з іншими технологіями.*** *Gemini 1.5* та *Claude 3, Claude 3.5* мають високий рівень інтеграції з іншими технологіями штучного інтелекту та машинного навчання, що дає можливість створювати гібридні системи, які можуть одночасно обробляти текст, зображення та інші типи даних, підвищуючи універсальність і продуктивність. У 2024 р. було про-

демонстровано системи, що інтегрують мовні моделі з технологіями обробки аудіо та відео, розширюючи їх застосування в мультимодальних середовищах.

Кількісні оцінки мовних моделей *Claude 3, Claude 3.5, Gemini 1.5, LLaMA 3, Mistral 7B* та *Mistral 12B,* враховуючи кількість параметрів, продуктивність (точність, F1-міру, перплексію) та тривалість навчання, наведено в табл. 16.21.

Найновіші мовні моделі, наведені в табл. 16.21, демонструють значний прогрес у галузі обробки природної мови, пропонуючи різноманітні рішення для широкого спектра завдань. Вони різняться за кількістю параметрів, точністю, F1-мірою, перплексією, часом навчання та загальними характеристиками, що дає можливість обрати модель, яка найбільше відповідає потребам користувача.

Кількість параметрів є одним із ключових показників, що впливають на продуктивність моделей. Найбільші моделі — *LLaMA 3* (200—300 млрд параметрів) і *Claude 3.5* (200—250 млрд параметрів), забезпечують високу точність і здатність вирішувати складні завдання завдяки своїй масштабності. Натомість компактні моделі, наприклад *Mistral 7B* і *12B* з відповідно 7 і 12 млрд параметрів, орієнтовані на ефективність і використання в умовах обмежених ресурсів. *Claude 3* і *Gemini 1.5* пропонують збалансоване співвідношення кількості параметрів і продуктивності, що робить їх універсальними.

У контексті точності моделі *Claude 3.5* і *Claude 3* є беззаперечними лідерами з показниками 96 і 95 умовних одиниць відповідно. Це свідчить про їхню здатність правильно передбачати результати навіть у складних завданнях. *Gemini 1.5, LLaMA 3* та *Mistral 12B* також демонструють високі показники точності (від 91 до 93), тоді як *Mistral 7B* з точністю 90 є менш продуктивною, проте залишається ефективною завдяки своїй компактності.

За показником F1-міри, який є гармонійним середнім між точністю та повнотою, також виділяються *Claude 3.5* (95) та *Claude 3* (94) як найбільш збалансовані моделі. *Gemini 1.5* (92) і *LLaMA 3* (91) також на високому рівні за цим показником, але поступаються лідерам. *Mistral 7B* (88) і *Mistral 12B* (89) залишаються надійними моделями для менш вимогливих сценаріїв.

Перплексія, що оцінює здатність моделі передбачати наступне слово в тексті, є найнижчою для *Claude 3.5* (9) і *Claude 3* (10). Це свідчить про їхню здатність ефективно працювати з контекстом. Вищі значення перплексії у *Mistral 7B* (16) і *LLaMA 3* (14) свідчать про можливі обмеження у складних завданнях, де потрібна глибока контекстна обізнаність.

Тривалість навчання моделей варіюється від 70 *(Mistral 7B)* до 100 *(LLaMA 3),* де найдовший час притаманний моделям із найбільшою кількістю параметрів. Енергоефективні моделі *Mistral 7B* і *12B* досягають оптимального співвідношення між часом навчання та продуктивністю, що робить їх привабливими для мобільних пристроїв і завдань із обмеженим

Таблиця 16.21

Порівняння групи найновіших мовних моделей за кількістю параметрів, продуктивністю та часом навчання (в умовних одиницях)

Модель	Кількість параметрів	Точність (*Accuracy*)	F1-міра (*F1-Score*)	Перплексія (*Perplexity*)	Тривалість навчання	Загальні характеристики
Claude 3	~100—200 млрд	95	94	10	90	Високий рівень генерування тексту, добре розуміння контексту, підтримка складних діалогів
Claude 3.5	~200—250 млрд	96	95	9	95	Покращене розуміння контексту, вища продуктивність у генеруванні тексту, оптимізація для мультимодальних завдань
Gemini 1.5	~200 млрд	93	92	12	85	Висока продуктивність генерування тексту, високий рівень інтеграції мультимодальних функцій
LLaMA 3	~200—300 млрд	92	91	14	100	Відмінне розуміння природної мови, підвищена продуктивність у складних завданнях, як-от у генеруванні креативних текстів
Mistral 7B	7 млрд	90	88	16	70	Компактна модель, висока продуктивність на малих наборах даних, ефективність на мобільних пристроях
Mistral 12B	12 млрд	91	89	15	75	Збалансована продуктивність, добре працює для вирішення складних завдань, оптимальне поєднання компактного розміру з високою ефективністю

Примітки. 1. *Точність* (*Accuracy*) відображає, наскільки правильно модель передбачає результати. Тут значення подано в умовних одиницях від 1 до 100, де 100 — ідеальна точність. Наприклад, *Claude 3* має найвищу оцінку — 95.

2. *F1-міра* (*F1-Score*) — це гармонійне середнє між точністю та повнотою. Значення також наведено в умовних одиницях від 1 до 100.

3. *Перплексія* (*Perplexity*) вимірює, наскільки добре модель передбачає ймовірність наступного слова в тексті. Менші значення кращі, тому в таблиці оцінка знижується з підвищенням перплексії.

4. *Час навчання* також оцінюється від 1 до 100, де 100 означає найдовшу тривалість навчання. Зокрема, *LLaMA 3* має найдовший час навчання за 300 млрд параметрів.

бюджетом ресурсів. *Claude 3.5* (95) і *Claude 3* (90) пропонують збалансовану тривалість навчання, враховуючи їхню високу продуктивність.

Загальні характеристики моделей підкреслюють їх унікальність. *Claude 3.5* вирізняється покращеним розумінням контексту, високою точністю та здатністю до мультимодальних завдань. *Gemini 1.5* є лідером у мультимодальній інтеграції, забезпечуючи високу продуктивність у текстових і візуальних завданнях. *LLaMA 3* демонструє сильні сторони у складних сценаріях (генерування креативних текстів), але потребує значних ресурсів. *Mistral 7B* і *12B* є найкращим вибором для компактних і мобільних застосунків, забезпечуючи баланс між розміром і ефективністю. *Claude 3* залишається універсальною моделлю, придатною для широкого спектра завдань.

Отже, кожна з моделей має свої сильні сторони та галузі застосування. *Claude 3.5* є найкращим вибором для завдань, які потребують максимальної точності, тоді як *Gemini 1.5* та *LLaMA 3* добре придатні для мультимодальних і креативних завдань. *Mistral 7B* і *12B* оптимальні для сценаріїв з обмеженими ресурсами, а *Claude 3* відзначається універсальністю для різноманітних застосувань. Такий вибір дає можливість адаптувати модель залежно від конкретних потреб та обмежень завдання.

Приклад. Завдання. Потрібно побудувати графіки залежності часу навчання мовних моделей *Claude 3, Claude 3.5, Gemini 1.5, LLaMA 3, Mistral 7B* та *Mistral 12B* від кількості параметрів (в абсолютних одиницях) для фіксованої кількості і певного типу *GPU*.

Вирішення. Для побудови графіків залежності часу навчання мовних моделей *Claude 3, Claude 3.5, Gemini 1.5, LLaMA 3, Mistral 7B* та *Mistral 12B* від кількості параметрів спершу оцінимо кількість параметрів цих моделей і орієнтовну тривалість їх навчання. Припустимо, маємо таку кількість параметрів для цих мовних моделей:

- *Claude 3* — ~70 млрд параметрів;
- *Claude 3.5* — ~200 млрд;
- *Gemini 1.5* — ~300 млрд;
- *LLaMA 3* — ~200 млрд;
- *Mistral 7B* — ~7 млрд;
- *Mistral 12B* — ~12 млрд.

За умови, що використовується 1024 *GPU* типу *NVIDIA A100 GPU*, орієнтовний час навчання для зазначених моделей становить:

- *Claude 3* — 9 тижнів;
- *Claude 3.5* — 9,5 тижнів;
- *Gemini 1.5* — 8,5 тижнів;
- *LLaMA 3* — 10 тижнів;
- *Mistral 7B* — 7 тижнів;
- *Mistral 12B* — 7,5 тижнів.

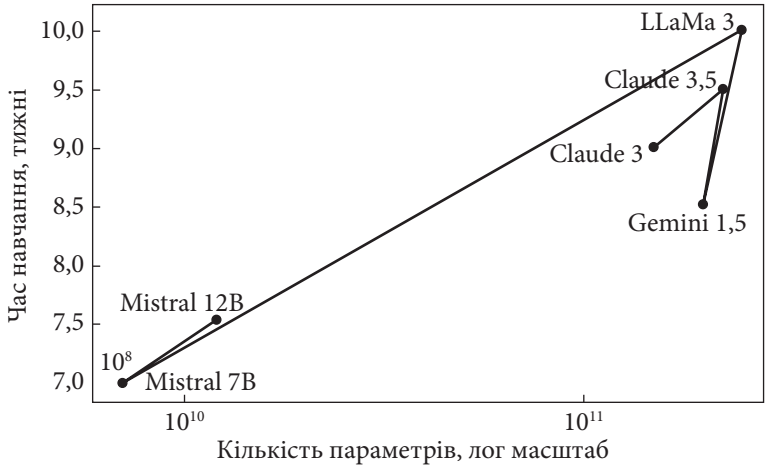


Рис. 16.6. Залежність часу навчання мовних моделей *Claude 3, Claude 3.5, Gemini 1.5, LLaMA 3, Mistral 7B* та *Mistral 12B* від кількості параметрів

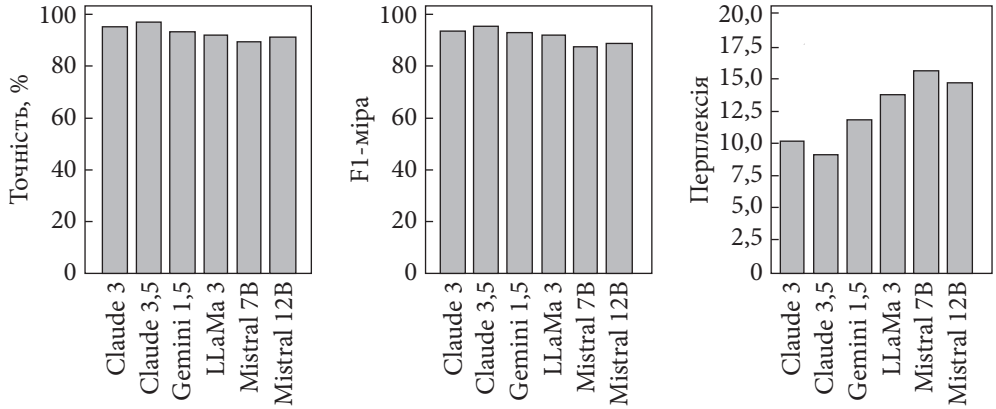


Рис. 16.7. Графіки, що ілюструють відносну продуктивність мовних моделей *Claude 3, Claude 3.5, Gemini 1.5, LLaMA 3, Mistral 7B* та *Mistral 12B*

На рис. 16.6 наведено графік, що відображає залежність часу навчання зазначених мовних моделей від кількості параметрів для фіксованої кількості *GPU* (1024 одиниці) типу *NVIDIA A100 GPU*. Бачимо, що тривалість навчання практично лінійно зростає зі збільшенням кількості параметрів моделей.

На рис. 16.7 наведено графіки, що ілюструють відносну продуктивність мовних моделей *Claude 3, Claude 3.5, Gemini 1.5, LLaMA 3, Mistral 7B* та *Mistral 12B* за трьома метриками:

1) *точність* (*Accuracy*) — відображає відсоток правильних передбачень моделі. Найвищі показники мають *Claude 3.5* і *Claude 3*;

2) *F1-міра* (*F1-Score*) — показує баланс між точністю і повнотою. Найкращий результат демонструє *Claude 3.5*;

3) *перплексія* (*Perplexity*) — оцінює здатність моделі передбачати наступне слово у тексті. Найнижчі значення (найкращі) у *Claude 3.5* і *Claude 3*.

Таким чином, отримані результати порівняння мовних моделей дають змогу зробити такі висновки:

1. Аналіз залежності тривалості навчання мовних моделей (рис. 16.6) від кількості параметрів для фіксованої кількості *GPU (1024 NVIDIA A100 GPU)* показав, що час навчання моделей практично лінійно зростає зі збільшенням кількості параметрів. Це свідчить про те, що масштабованість моделей є ключовим фактором, що впливає на час навчання. Компактні моделі *Mistral 7B* і *Mistral 12B* демонструють суттєво нижчі витрати часу на навчання порівняно з більшими моделями *Claude 3.5* чи *Gemini 1.5*.

2. Аналіз продуктивності моделей (рис. 16.7) за трьома основними метриками — точність *(Accuracy)*, F1-міра *(F1-Score)* та перплексія *(Perplexity)* — виявив, що моделі *Claude 3.5* і *Claude 3* є найкращими за всіма показниками. Вони демонструють найвищу точність, збалансованість між точністю та повнотою передбачень (за F1-мірою) і найнижчу перплексію, що є показником точності передбачення тексту.

3. Разом з тим менш масштабні моделі *Mistral 7B* і *Mistral 12B,* хоч і поступаються великим моделям за продуктивністю, все ж забезпечують оптимальний баланс між часом навчання, вимогами до ресурсів та достатньою продуктивністю для багатьох практичних завдань.

4. Отже, вибір мовної моделі залежить від специфіки завдання: для високоточних завдань, які потребують максимальної продуктивності, доцільно використовувати моделі на кшталт *Claude 3.5.* Водночас компактні моделі *Mistral 7B* і *Mistral 12B* є відмінним вибором для ресурсозалежних сценаріїв, де важлива ефективність і швидкість навчання.

16.4. Апаратне і програмне забезпечення машинного навчання мовних моделей

Розвиток машинного навчання і глибокого навчання потребує ефективного використання апаратного арсеналу для забезпечення обчислювальної потужності. Центральні процесори *(Central Processing Unit, CPU),* графічні *(Graphics Processing Unit, GPU),* тензорні *(Tensor Processing Unit, TPU)* та нейронні *(Neural Processing Unit, NPU)* процесори є ключовими технологіями, які застосовують для вирішення різних завдань машинного навчання. Кожна з цих технологій має свої особливості, переваги та недоліки, що визначають їх використання для різних завдань і моделей [16.45, 16.46].

Головні обчислювальні платформи для машинного навчання мовних моделей

Ефективне використання обчислювальних технологій і платформ для машинного навчання мовних моделей потребує ретельного врахування не лише специфіки завдання, але й ключових факторів, як-от тривалість навчання, економічність, а також вимоги до енергоспоживання. Вибір правильної платформи є критичним для оптимізації процесів машинного навчання та досягнення максимальних результатів у конкретних сценаріях. Розглянемо головні з них.

Центральний процесор *(Central Processing Unit, CPU)* є універсальним обчислювальним пристроєм, здатним виконувати широкий спектр завдань. У контексті машинного навчання *CPU* зазвичай використовують для підготовки даних — очищення, нормалізації та перетворення даних на зручний формат. *CPU* також ефективні для виконання невеликих моделей і швидких інференцій.

Основними перевагами *CPU* є їхня універсальність, доступність і низьке енергоспоживання. Вони придатні для виконання простих моделей, як-от логістична регресія чи дерева рішень, а також для інтеграції невеликих моделей у мобільні чи вебдодатки. Проте через обмежену паралельність обчислень *CPU* поступаються *GPU* та *TPU* у навчанні великих нейронних мереж.

Графічний процесор *(Graphics Processing Unit, GPU)* вперше був розроблений компанією *NVIDIA* у 1999 р. з метою виконання паралельних обчислень. Це зробило його надзвичайно ефективним для обробки великих наборів даних і навчання нейронних мереж. *GPU* широко застосовують для навчання моделей середнього і великого масштабу — *BERT, GPT-2* та ін. Завдяки високій швидкості обчислень *GPU* є стандартним вибором для дослідників і розробників. Головною перевагою *GPU* є їхня здатність виконувати тисячі паралельних обчислень, що значно пришвидшує навчання моделей. Проте *GPU* мають високе енергоспоживання, і їхня вартість є вищою порівняно з *CPU*. Попри це *GPU* залишаються оптимальним вибором для більшості завдань, які потребують інтенсивних обчислень.

Тензорний процесор *(Tensor Processing Unit, TPU)* є спеціалізованим пристроєм, створеним компанією *Google* в 2016 р. для навчання нейронних мереж. Процесори *TPU* оптимізовані для роботи з *TensorFlow* — популярною платформою для машинного навчання, яка забезпечує інструменти й бібліотеки для побудови, навчання і розгортання моделей ШІ. *TPU* особливо корисні для навчання великих мовних моделей, наприклад *GPT-3*. *TPU* забезпечують високу продуктивність, оптимальне енергоспоживання і є потужним інструментом для масштабного машинного навчання. *TPU* вирізняються своєю архітектурою, яка забезпечує швидке оброблення матричних обчислень, характерних для глибоких нейронних мереж. Це дає

змогу навчати великі моделі швидше і з меншими витратами енергії порівняно з *GPU*. Проте *TPU* мають обмежену доступність і спеціалізовані переваги для певних фреймворків.

Нейронний процесор *(Neural Processing Unit, NPU)* — спеціалізований пристрій, створений компанією *Huawei* в 2017 р. *NPU* призначений для виконання завдань, пов'язаних із штучними нейронними мережами. Цей тип процесора розроблено для інтенсивних обчислень, характерних для алгоритмів машинного навчання, з акцентом на максимальну швидкість і ефективність. Завдяки своїй архітектурі *NPU* стає важливим компонентом сучасних пристроїв, як-от смартфонів, пристроїв Інтернету речей *(IoT)* та автономних систем, що використовують моделі штучного інтелекту для вирішення широкого спектра завдань.

Основною метою створення *NPU* є прискорення роботи алгоритмів ШІ та глибокого навчання *(Deep Learning)* задля зниження енергоспоживання і витрат на обчислення. Це досягається завдяки спеціалізації *NPU* на операціях, які є основою глибоких нейронних мереж. Можливості *NPU* сприяють виконанню пристроями інтелектуальних завдань без необхідності звернення до хмарних обчислень, що особливо важливо для швидких і автономних рішень.

Однією з ключових характеристик *NPU* є їх оптимізація для нейронних мереж. Вони здатні обробляти матричні множення та згортки, що є базовими операціями для більшості моделей глибокого навчання. Завдяки підтримці паралельної обробки *NPU* забезпечують значне пришвидшення як навчання моделей, так і їх інференції, що робить їх незамінними для реального часу. Порівняно з іншими процесорами — *CPU* та *GPU, NPU* споживають значно менше енергії, оскільки їхня архітектура спеціалізована для вузького класу завдань. Завдяки цьому вони надзвичайно ефективні для використання в мобільних пристроях та інших енергообмежених середовищах.

Отже, *NPU* є ключовою технологією для сучасного розвитку ШІ. Оптимізація для нейронних мереж, висока енергоефективність і швидкість обчислень роблять їх важливим компонентом інтелектуальних пристроїв, що працюють у реальному часі. Завдяки своїм характеристикам *NPU* дають змогу інтегрувати потужні функції ШІ у пристрої з обмеженими ресурсами, відкриваючи нові можливості для їх використання у смартфонах, автономних транспортних засобах, *IoT* та багатьох інших сферах.

Порівняльні характеристики центральних *(CPU),* графічних *(GPU),* тензорних *(TPU)* і нейронних *(NPU)* процесорів наведено в табл. 16.22 [16.46].

Порівняння характеристик *CPU, GPU, TPU* та *NPU* [16.46] ілюструє їхні сильні сторони й сфери застосування у завданнях машинного навчання. Ці процесори різняться за ключовими параметрами: паралельність, енергоспоживання, швидкість обчислень та вартість, що визначає їхню ефективність у різних сценаріях.

Паралельність є одним із головних факторів продуктивності. *CPU,* маючи від 4 до 64 ядер, добре справляється з багатозадачністю і послідовними обчисленнями, але обмежений у завданнях, які потребують високої паралельності. *GPU,* оснащені тисячами ядер (до 5000 у *NVIDIA A100*), ефективніші для масивних паралельних обчислень, необхідних для навчання нейронних мереж. *TPU* виходять на перше місце за цим показником, маючи до 128 000 обчислювальних блоків *ALU (Arithmetic Logic Unit),* що дає їм можливість працювати з дуже масштабними моделями. *ALU* — це ключові компоненти центрального *(CPU),* графічного *(GPU)* і тензорного *(TPU)* процесорів та інших спеціалізованих чипів, які виконують основні математичні та логічні операції. Саме ці блоки безпосередньо обробляють числові дані та виконують інструкції, необхідні для роботи програм. *NPU,* хоча й менш масштабні за кількістю обчислювальних блоків, оптимізовані для локального оброблення та виконання завдань у реальному часі, наприклад, для інференції на мобільних пристроях.

Енергоспоживання відіграє важливу роль у виборі апаратного забезпечення. *CPU* споживають невеликі обсяги енергії (~50 Вт), що робить їх еко-

Таблиця 16.22

Порівняльні характеристики процесорів *CPU, GPU, TPU* та *NPU*

Характеристика	*CPU*	*GPU*	*TPU*	*NPU*
Призначення	Універсальні завдання	Навчання нейронних мереж	Масштабні нейронні мережі	Інференція та завдання ШІ у реальному часі
Паралельність	4—64 ядра	До 5 000 ядер	До 128 000 *ALU (Arithmetic Logic Units)*	До 10 000 спеціалізованих обчислювальних блоків
Енергоспоживання	~50 Вт	~200—300 Вт	~150 Вт (залежно від моделі)	~5—20 Вт (для мобільних пристроїв)
Швидкість обчислень	До 500 *GFLOPS*	До 15 *TFLOPS (NVIDIA A100)*	До 120 *TFLOPS (TPU v4)*	Висока (залежно від пристрою, наприклад, до 1 *TOPS* для мобільних *NPU)*
Вартість	$ 300—500 (звичайний процесор)	$ 5 000—10 000 *(NVIDIA A100)*	~$ 8/год у хмарних сервісах *(TPU v4)*	Інтегровано у пристрої, наприклад, у смартфони та *IoT*
Сфера застосування	Підготовка даних, невеликі моделі	Середні й великі моделі	Великі мовні моделі	Обробка зображень, мови, *AR (Augmented Reality)* доповненої реальності у реальному часі, мобільні додатки

номічно вигідними для завдань підготовки даних і роботи з невеликими моделями. *GPU* (споживають 200—300 Вт) забезпечують високу продуктивність для середніх і великих моделей. *TPU* є оптимізованими для масштабних задач, як-от *GPT-3,* із середнім енергоспоживанням ~150 Вт. *NPU* демонструють найкращі показники енергоефективності (~5—20 Вт), що робить їх придатними для мобільних пристроїв і *IoT*, де обмеження енергоспоживання є критичним.

Швидкість обчислень, вимірювана у *FLOPS (Floating Point Operations Per Second),* є вирішальним фактором у продуктивності процесорів. *CPU* досягають продуктивності до 500 *GFLOPS,* чого достатньо для базових задач. *GPU* типу *NVIDIA A100*, досягають 15 *TFLOPS*, що робить їх стандартним вибором для середніх моделей. *TPU* мають найвищу продуктивність — до 120 *TFLOPS (TPU v4),* оптимальну для навчання масштабних нейронних мереж. *NPU,* хоч і менш потужні в абсолютних одиницях (до 1 *TOPS* для мобільних версій), забезпечують високу швидкість виконання завдань у реальному часі, наприклад розпізнавання зображень чи голосу.

Вартість також є важливим фактором. *CPU* мають найнижчу вартість ($ 300—500), що робить їх доступними для широкого спектра завдань. *GPU* є дорожчими ($ 5 000—10 000 для *NVIDIA A100*), але виправдовують свою ціну для середніх і великих моделей. *TPU* доступні через хмарні сервіси, із вартістю ~$ 8 за годину роботи. *NPU* зазвичай інтегровані у мобільні процесори (наприклад, *Huawei Kirin* або *Apple A*-серії), тому вартість зазвичай не виділяється окремо, що робить їх доступними для споживачів без додаткових витрат.

Сфера застосування для кожного процесора залежить від його особливостей. *CPU* найкраще придатні для підготовки даних, аналізу та роботи з невеликими моделями. *GPU* оптимальні для навчання середніх і великих моделей завдяки високому ступеню паралельності. *TPU* спеціалізуються на масштабних нейронних мережах, забезпечуючи максимальну продуктивність із оптимізованим енергоспоживанням. *NPU* найкраще виконують завдання інференції в реальному часі на пристроях з обмеженими ресурсами — смартфонах та *IoT.*

Час і вартість навчання мовних моделей є визначальними факторами для вибору обчислювальних платформ з огляду на велику ресурсомісткість цих процесів. Високий рівень складності моделей, як-от *GPT-3,* потребує масштабних обчислювальних ресурсів, значних обсягів даних і тривалого часу навчання. У цьому контексті вибір обчислювальної платформи (*CPU, GPU, TPU* або *NPU*) стає критично важливим для оптимізації витрат і часу. Дані стосовно часу і вартості навчання моделі *GPT-3* на різних типах процесорів наведено в табл. 16.23.

Дані табл. 16.23 демонструють суттєву різницю в ефективності, продуктивності та економічній доцільності використання кожної обчислю-

вальної платформи. На прикладі навчання мовної моделі *GPT-3* можна дійти таких висновків:

• *CPU,* попри свою універсальність, є найменш ефективним рішенням для навчання масштабних моделей, як-от *GPT-3*. Значні тривалість і вартість навчання роблять його придатним лише для завдань малого масштабу або попереднього оброблення даних.

• *GPU* є значно продуктивнішими процесорами завдяки своїй паралельній архітектурі. Вони забезпечують баланс між часом навчання (3—6 міс.) та витратами (~$10—20 млн), що робить їх популярним вибором для середніх і великих моделей машинного навчання. Однак їхня вартість усе ще залишається високою для завдань, що потребують масштабування.

• *TPU* є найбільш економічно вигідним варіантом для навчання *GPT-3*. Завдяки високій продуктивності час навчання скорочується до 1—2 міс., а вартість становить ~$5—10 млн. *TPU* оптимізовані для завдань глибокого навчання, таких як навчання мовних моделей, що робить їх найкращим вибором для масштабних і ресурсомістких проєктів.

• *NPU,* хоча й мають велику енергоефективність і швидкість для інференції, не придатні для навчання моделей такого масштабу як *GPT-3* через архітектурні обмеження. Їхня роль полягає в обробленні даних у реальному часі.

На прикладі навчання масштабної мовної моделі *GPT-3* бачимо, що оптимальним вибором є *TPU,* які забезпечують найменші час і вартість навчання завдяки спеціалізованій архітектурі для матричних обчислень. *GPU* залишаються добрим варіантом для завдань середнього масштабу, де потрібні гнучкість і висока продуктивність. *CPU* найкраще придатні для базових завдань підготовки даних і невеликих моделей, тоді як *NPU* виконують специфічні завдання інференції, але не придатні для масштабного навчання. Ці результати підкреслюють важливість вибору відповідної

Таблиця 16.23

Час і вартість навчання моделі *GPT-3* на різних типах процесорів

Процесор	Кількість процесорів	Час навчання	Вартість навчання	Особливості
CPU	1 000	~2 роки	~$ 50—70 млн	Висока універсальність, але низька паралельність обчислень
GPU	256	~3—6 місяців	~$ 10—20 млн	Висока паралельність, оптимізовані для навчання моделей *ML*
TPU	128	~1—2 місяці	~$ 5—10 млн	Спеціалізовані для матричних обчислень, ефективні для *ML*
NPU	Недоступний для *GPT-3*	Непридатний	Непридатний	*NPU* орієнтовані на інференцію, не на навчання масштабних моделей

платформи для досягнення балансу між часом, вартістю та продуктивністю у завданнях машинного навчання.

Отже, вибір між *CPU, GPU, TPU* та *NPU* визначається специфікою завдання, доступним бюджетом, вимогами до продуктивності, а ще такими важливими факторами як час і вартість навчання мовних моделей. *CPU* залишаються універсальним рішенням для базових завдань, підготовки даних або аналізу невеликих моделей завдяки їхній доступності та низькому енергоспоживанню. *GPU,* забезпечуючи високу паралельність і продуктивність, стають оптимальним вибором для середніх і великих моделей, пропонуючи баланс між швидкістю навчання і вартістю. *TPU* є найкращим варіантом для масштабних завдань навчання мовних моделей *GPT-3* завдяки спеціалізованій архітектурі, яка дає змогу значно скоротити час і витрати. *NPU,* хоча й не призначені для навчання великих моделей, є лідерами у виконанні інференції в реальному часі завдяки своїй енергоефективності та швидкості, особливо в мобільних пристроях та *IoT.*

Приклад. *Постановка завдання.* Потрібно оцінити доцільність вибору графічних процесорів *GPU* типу *NVIDIA A100* для навчання мовної моделі *LLaMA 3* з 300 млрд параметрів. Виконати оцінку тривалості навчання мовної моделі *LLaMA 3* для 300 млрд параметрів можна з огляду на кілька факторів: кількість і тип графічних процесорів, обсяг даних для навчання, ефективність алгоритмів оптимізації і розмір мініпакетів *(batch size).*

Вирішення. Нехай маємо мовну модель з такими параметрами:

1) кількість параметрів — 300 млрд;

2) кількість даних для навчання — 1,8 трлн токенів;

3) апаратне забезпечення — 1024 графічних процесорів типу *NVIDIA A100;*

4) швидкість навчання — близько 200 *TFLOPs* (терафлопс) на один *GPU.*

Кроки розрахунку:

1. Кількість *FLOPs (floating point operations)* для одного параметра на один токен обчислюють так. Для однієї ітерації навчання (одного *forward-* і *backward*-проходження) потрібно приблизно 6 *FLOP* на один параметр для одного токена. Тобто для 300 млрд параметрів це становитиме

$6 \cdot 300 \cdot 10^9 = 1{,}8 \cdot 10^{12}$ *FLOPs* на один токен.

2. Загальну кількість *FLOPs* на весь обсяг даних за обсягу даних у 1,8 трлн токенів обчислюють так:

$1{,}8 \cdot 10^{12} \cdot 10^{12} = 1{,}8 \cdot 10^{24}$ *FLOPs*.

3. Загальна обчислювальна потужність для 1024 *GPU,* кожен з яких має продуктивність 200 *TFLOPS*, становить $2{,}048 \cdot 10^{17}$ *FLOPs* на секунду.

4. Час навчання моделі *LLaMA 3* з 300 млрд параметрів на 1024 процесорах *GPU* типу *NVIDIA A100* загальною обчислювальною продуктивністю $2{,}048 \cdot 10^{17}$ *FLOPs* на секунду, за обсягу даних в 1,8 трлн токенів може зайняти приблизно $9 \cdot 10^6$ секунд, або 104 дні.

Висновок. У зв'язку з тим, що час навчання моделі *LLaMA 3* з 300 млрд параметрів на 1024 процесорах *GPU* типу *NVIDIA A100* виявився досить великим (104 дні), очевидно, що вибір цього типу *GPU* з обчислювальною продуктивністю 19,5 *TFLOPs* для *FP32 (Single Precision)* може виявитися неприйнятним з практичної точки зору. У такому разі можна рекомендувати обрати *GPU* типу *NVIDIA H100*, який має приблизно втричі вищу обчислювальну продуктивність (60 *TFLOPs*) для цього типу задач. Це робить сьогодні *H100* найпотужнішим *GPU* на ринку для *AI* завдань.

Мови програмування, бібліотеки та фреймворки для обробки природної мови

Для вирішення завдань *NLP* використовують різні мови програмування, бібліотеки та фреймворки, які допомагають розробникам працювати з текстовими даними і розробляти моделі. Розглянемо кожну з цих складових [16.47—16.49].

Мови програмування для обробки природної мови — це програмні інструменти, які дають можливість автоматизувати аналіз та маніпуляцію з текстами під час оброблення природної мови (наприклад, англійської, української, китайської тощо) [16.47, 16.48]. Вони забезпечують інструменти і структури для написання коду, який виконує різні операції з текстом, і надають можливості для розробки алгоритмів та моделей, які можуть «розуміти», інтерпретувати й генерувати мову людини.

Тобто мови програмування, використовувані для *NLP,* надають розробникам інструменти для роботи з текстом, уможливлюючи автоматизацію процесів, які раніше потребували участі людини.

Головні завдання, які вирішують мови програмування в NLP:

1. *Токенізація (Tokenization)* — розбиття тексту на менші одиниці — слова або речення.

2. *Морфологічний аналіз* — визначення граматичних характеристик слів — частин мови, часів дієслів тощо.

3. *Лематизація і стемінг* — перетворення слів до їх основної форми (леми) або основи (стему).

4. *Розпізнавання іменованих сутностей* (*Named Entity Recognition, NER)* — виявлення і класифікація імен, організацій, локацій та інших специфічних термінів у тексті.

5. *Парсинг* — синтаксичний аналіз речень для визначення їх граматичної структури.

6. *Моделювання мовних моделей* — створення моделей, що можуть передбачати наступне слово в реченні або генерувати нові тексти.

7. *Класифікація тексту* — визначення категорії тексту (наприклад, позитивний чи негативний відгук).

8. *Переклад тексту* — автоматичний переклад тексту з однієї мови на іншу.

9. *Аналіз сентименту* — визначення емоційного забарвлення тексту (наприклад, чи є текст позитивним, нейтральним чи негативним).

10. *Вилучення інформації* (*Information Extraction)* — автоматичне вилучення важливої та структурованої інформації (певних сутностей, відносин, фактів або подій) з неструктурованих текстових даних.

Огляд основних мов програмування в NLP. Найпоширеніші мови програмування для *NLP* станом на 2024 р. наведено в табл. 16.24.

Мови програмування, наведені у табл. 16.24, охоплюють широкий спектр завдань — від аналізу даних і наукових обчислень до створення масштабованих систем та інтерактивних вебдодатків [16.47, 16.48]. Зокрема:

Python є універсальною мовою, яка вирізняється простотою синтаксису і широким набором бібліотек для різноманітних завдань. Вона ідеально придатна для аналізу даних, машинного навчання та автоматизації. *Python* є чудовим вибором для швидкого створення прототипів завдяки своїй легкості у використанні. Проте у складних обчислювальних сценаріях продуктивність *Python* може поступатися іншим мовам — *C++* чи *Julia.*

R — це спеціалізована мова, орієнтована на статистичний аналіз та візуалізацію даних. Її головна перевага — потужні інструменти для роботи з числовими даними та створення графіків. Хоча *R* менш універсальна ніж *Python*, вона незамінна у статистичних дослідженнях і є основним інструментом для науковців, які займаються аналізом даних.

Java має довгу історію використання в корпоративному програмуванні. Її портативність через *JVM*, висока продуктивність і безпека роблять мову оптимальним вибором для масштабованих додатків, мобільних платформ і роботи з великими даними. Однак складність синтаксису та необхідність налаштувань можуть бути перешкодою для новачків.

Julia — це сучасна мова, створена для наукових обчислень і високопродуктивних завдань. Вона поєднує простоту *Python* із швидкістю *C++*, що робить її популярною в моделюванні та чисельних симуляціях. Проте обмежена кількість бібліотек і порівняно новий статус мови можуть створити бар'єри для широкого впровадження.

JavaScript — основна мова для розроблення вебдодатків. Можливість працювати як на клієнтській (фронтенд), так і на серверній (бекенд) сторонах робить її універсальним інструментом для створення інтерактивних вебдодатків. Попри це, *JavaScript* менш ефективна для складних обчислювальних задач.

C++ є мовою високої продуктивності, яка забезпечує контроль над пам'яттю і оптимізацію на низькому рівні. Це робить її ідеальною для розроблення ігор, операційних систем і додатків, де важлива швидкість. Однак складний синтаксис і необхідність детального управління пам'яттю можуть ускладнити її використання.

Таблиця 16.24

Найпоширеніші мови програмування для *NLP* (2024 рік)

Мова програмування	Сфера застосування	Популярність *GhatGPT* (1—10)	Головні характеристики, обмеження	Кількість бібліотек	Продуктивність *(GFLOPS)*
Python	*Дослідницька діяльність, освіта, бізнес:* реалізація базових процесів *NLP,* машинне навчання, розробка чатботів, обробка текстів, автоматизація бізнес-процесів	10	*Характеристики:* об'єктно орієнтована, процедурна, функціональна мова, має велику кількість бібліотек, простий синтаксис, зручний для новачків, широку підтримку спільноти та досконалу документацію *Обмеження:* низька продуктивність порівняно з компільованими мовами, менш ефективна для програм, які потребують низькорівневого доступу до ресурсів	~200+	~5—10
R	*Академічні і наукові дослідження:* статистичний аналіз текстів, текстова аналітика, соціальні дослідження	9	*Характеристики:* функціональна, процедурна мова, спеціалізована для статистики, аналізу даних та побудови графіків, багато бібліотек для статистичного аналізу (*ggplot2, dplyr, tm*), активна підтримка в наукових колах *Обмеження:* повільніше виконання порівняно з *Python,* менше універсальних можливостей для розробки	~50+	~2—5
Java	*Корпоративне середовище і бізнес:* багатопотокові застосунки, програми підтримки масштабних обчислень	8	*Характеристики:* об'єктно орієнтована мова, має високу надійність і масштабованість, підтримку багатопотоковості, велику кількість бібліотек для *NLP,* кросплатформність через *JVM (Java Virtual Machine)* *Обмеження:* складніший синтаксис, довший час розроблення, потребує більше пам'яті для виконання	~40+	~10—15

Закінчення табл. 16.24

Мова програмування	Сфера застосування	Популярність *GhatGPT* (1—10)	Головні характеристики, обмеження	Кількість бібліотек	Продуктивність *(GFLOPS)*
Julia	*Академічні дослідження, технологічні стартапи: Data Mining,* обчислювальна лінгвістика	7	*Характеристики:* функціональна, процедурна мова, має високу продуктивність, близьку до *C++*, спеціалізована для масштабних обчислень, має можливість роботи з великими наборами даних, легкість інтеграції з *Python, R, C++* *Обмеження:* менша кількість бібліотек порівняно з *Python*, відносно нова мова з меншим розміром спільноти	~20+	~20—25
JavaScript	*Бізнес, технологічні стартапи:* створення інтерактивних вебзастосунків і чатботів, клієнтоорієнтованих рішень	7	*Характеристики:* об'єктно орієнтована, функціональна мова, має високу інтерактивність у вебдодатках, використовується як на боці клієнта, так і на сервері *(Node.js)*, зручний синтаксис для створення швидких прототипів *Обмеження:* має обмежену продуктивність для масштабних обчислень, не оптимізована для великих і складних завдань *NLP*	~10+	~5
C++	*Академічні дослідження, промисловість, бізнес:* розробка продуктивних систем *NLP,* створення алгоритмів і фреймворків, систем обробки даних у реальному часі	6	*Характеристики:* об'єктно орієнтована, процедурна мова, має найвищу продуктивність серед мов, придатна для низькорівневих і реальночасових застосунків, використовується для розробки фреймворків, як-от *TensorFlow* *Обмеження:* складність синтаксису, менш активна спільнота в контексті *NLP*	~15+	~50—60

Ruby	*Бізнес, технологічні стартапи:* розробка вебзастосунків, орієнтованих на інтеграцію в *NLP*, текстова аналітика, прості *NLP*-програми	6	*Характеристики:* об'єктно орієнтована мова, має простий і зрозумілий синтаксис, придатна для швидкого розроблення прототипів, має достатню кількість бібліотек для *NLP* (проте менше ніж у *Python*) *Обмеження:* низька продуктивність, менший набір інструментів для складних завдань	~10+	~5
PHP	*Бізнес, технологічні стартапи:* розробка вебзастосунків, орієнтованих на інтеграцію в *NLP*, текстова аналітика, інтеграція *NLP* у бізнес-процеси	5	*Характеристики:* об'єктно орієнтована, процедурна, орієнтована на веброзробку, має легку інтеграцію з базами даних, використовується для розробки вебдодатків із вбудованими *NLP*-функціями *Обмеження:* низька продуктивність у масштабних завданнях, має обмежену кількість бібліотек для *NLP*	~5+	~3
MATLAB	*Академічні і наукові дослідження:* статистичний аналіз текстів, текстова аналітика, соціальні дослідження	5	*Характеристики:* об'єктно орієнтована мова, має інтуїтивний синтаксис для роботи з числовими даними, велику кількість вбудованих функцій для інженерних задач, підтримує інтеграцію з іншими мовами — *Python*, *C++*, *Java* *Обмеження:* має високу вартість ліцензії, закриту екосистему, менш ефективна для великих системних проєктів	~10+	~8—10
Scala	*Фінансовий сектор, фондові ринки, бізнес:* обробка великих даних, розподілені обчислення, аналітика фондових ринків, текстова аналітика	4	*Характеристики:* об'єктно орієнтована та функціональна мова, має високу масштабованість, багатофункціональність і можливість одночасного оброблення великих обсягів даних, інтегрується з екосистемою *Java* *Обмеження:* складний синтаксис для новачків, тривале навчання через велику кількість можливостей, менш поширена порівняно з *Java* чи *Python*	~15+	~7—10

Ruby — це мова, відома своєю простотою і зручністю для створення вебдодатків завдяки фреймворку *Ruby on Rails.* Вона популярна серед стартапів і малих проєктів, але обмежена продуктивність може бути недоліком для великих систем.

PHP залишається однією з основних мов для веброзроблення, особливо в роботі з популярними *CMS,* як-от *WordPress.* Завдяки простоті у використанні вона доступна для початківців, але з точки зору сучасного вебпрограмування *PHP* вважається менш елегантним порівняно з новими технологіями.

MATLAB спеціалізується на інженерних і наукових обчисленнях, надаючи потужні інструменти для моделювання та оброблення сигналів. Вона популярна в академічному середовищі, але її висока вартість і закрита екосистема обмежують використання у великих корпоративних проєктах.

Scala як мова для роботи з великими даними та розподіленими системами поєднує в собі переваги функціонального і об'єктно орієнтованого програмування. Її інтеграція з *Java* робить *Scala* чудовим вибором для оброблення великих даних із використанням *Apache Spark.* Проте складність синтаксису і тривале навчання можуть бути перешкодою для новачків.

Концептуальні характеристики найпоширеніших мов програмування у завданнях *NLP.* Основні критерії вибору мови програмування для *NLP*

Вибір мови програмування для завдань обробки природної мови є важливим кроком, який значною мірою впливає на ефективність та якість остаточного рішення. Розглянемо основні критерії, що допомагають зробити правильний вибір, та їх значення у контексті *NLP* [16.49].

Одним із найважливіших критеріїв є ***продуктивність і швидкодія***. У завданнях *NLP* часто працюють із великими обсягами тексту, а сучасні методи-трансформери потребують значних обчислювальних ресурсів. Тому мови, які забезпечують ефективне використання ресурсів і швидке виконання алгоритмів, як-от *C++* чи *Julia,* є оптимальним вибором для високонавантажених систем. Водночас *Python,* хоч і менш продуктивний, компенсує цей недолік широким спектром оптимізованих бібліотек.

Зручність у використанні також відіграє ключову роль, особливо для команд із різним рівнем підготовки. Мови з простим синтаксисом і зрозумілими бібліотеками (*Python*), дають можливість швидко розпочати розроблення і зосередитися на завданні, а не на технічних деталях. Це особливо важливо для проєктів, де потрібно швидко створити прототип або протестувати нові ідеї.

Ще одним важливим критерієм є ***підтримка сучасних методів NLP.*** Мова програмування повинна мати доступ до бібліотек, які реалізують передові алгоритми — *BERT, GPT* та інші трансформери. Наприклад, *Python*

пропонує широкий вибір інструментів для *NLP,* включно з *Hugging Face Transformers, spaCy, NLTK,* що робить його провідним вибором для цієї галузі.

Гнучкість* і *можливості розширення визначають, наскільки легко мову можна адаптувати до специфічних завдань. Наприклад, мови, які підтримують модульність і кастомізацію (*Scala* чи *Python*), допомагають створювати унікальні рішення, які повністю відповідають вимогам проєкту. Це особливо важливо для складних завдань, що виходять за межі стандартних сценаріїв.

Розвинена ***екосистема бібліотек* і *фреймворків*** є ще одним значущим фактором. Мова повинна мати доступ до інструментів для всіх етапів *NLP*-процесу — від токенізації до розгортання моделей. *Python* є лідером у цьому аспекті завдяки наявності фреймворків *TensorFlow, PyTorch* та ін.

Не менш важливим є ***ком'юніті та підтримка***. Розвинена спільнота забезпечує доступ до документації, прикладів використання та допомоги у вирішенні проблем. Наприклад, *Python* і *Java* мають одну з найбільших спільнот розробників, що спрощує розробку навіть для новачків.

Масштабованість мови програмування визначає її здатність обробляти великі обсяги даних або працювати в розподілених системах. Мови *Scala* (з інтеграцією *Apache Spark*) та *Java* часто використовують для створення масштабованих *NLP*-рішень, що працюють у хмарних середовищах.

Портативність є важливою для інтеграції з іншими системами чи мовами програмування. Наприклад, *Java* і *Python* легко інтегруються з різними платформами й технологіями, що робить їх універсальними для розроблення багатоплатформових рішень.

Ще одним важливим критерієм є ***сумісність із наявними технологіями.*** Це означає і підтримку роботи з хмарними сервісами, базами даних та *API. Python* і *JavaScript* мають численні рішення для інтеграції з сервісами *Google Cloud NLP* або *AWS Comprehend.*

Фінансова доступність мови також впливає на вибір. Мови з безкоштовними інструментами та бібліотеками, як-от *Python,* є більш привабливими для стартапів і невеликих команд. *MATLAB,* навпаки, має високу вартість ліцензії, що може обмежувати її використання в деяких проєктах.

Можливість інтерактивної роботи сприяє легкому виконанню експериментів та візуалізації результатів. Наприклад, *Python* відмінно працює у середовищі *Jupyter Notebook*, що є великим плюсом для дослідників і студентів.

Надійність та стабільність мови є критично важливими для розробки комерційних рішень, де помилки можуть призвести до серйозних наслідків. *Java* і *C++* добре відомі своєю стабільністю в масштабних системах.

Нарешті, ***можливості паралельного виконання*** важливі для оброблення великих обсягів тексту або одночасного виконання численних завдань. Мови *C++* і *Scala* забезпечують високий рівень багатопотоковості, що робить їх оптимальними для таких завдань.

Отже, оцінка цих критеріїв у контексті конкретного завдання дає можливість обрати найефективніший інструмент.

Сфери застосування мов програмування в *NLP.* Розроблення систем обробки природної мови потребує ретельного вибору мови програмування, який залежить від специфічних завдань, доступних ресурсів та необхідного рівня продуктивності. Кожна мова програмування має певні переваги й обмеження, що визначають її оптимальні сфери застосування в *NLP*: дослідницьку діяльність, освіту, бізнес, технологічні стартапи або промислові рішення.

Python залишається основним вибором для широкого спектра *NLP*-завдань завдяки універсальності, простоті використання та багатій екосистемі бібліотек. Він є стандартом для дослідницької діяльності, оскільки дає змогу швидко реалізовувати і тестувати нові алгоритми за допомогою таких бібліотек як *NLTK, SpaCy, Hugging Face, TensorFlow* і *PyTorch.* У бізнесі *Python* застосовують для розроблення чатботів, автоматизованого аналізу відгуків клієнтів та автоматизації бізнес-процесів. У освітньому середовищі *Python* популярний для навчання студентів базових концепцій *NLP* завдяки його простому синтаксису і широкому доступу до документації.

Java посідає важливе місце в промислових застосунках *NLP.* Завдяки стабільності та масштабованості *Java* активно використовується в корпоративному середовищі для створення систем обробки текстів у реальному часі, наприклад, автоматичному перекладі або класифікації електронної пошти. У бізнесі *Java* незамінна для розроблення систем із великим навантаженням, зокрема багатопотокових застосунків і програм із підтримкою масштабних обчислень.

Julia демонструє високу продуктивність у завданнях, що потребують значних обчислювальних ресурсів. Її основні сфери — академічні дослідження, обчислювальна лінгвістика та обробка великих текстових наборів. *Julia* також використовують у технологічних стартапах, які працюють з новітніми алгоритмами *NLP* і потребують високої швидкості обчислень для аналізу великих даних.

C++ вирізняється найвищою продуктивністю серед усіх розглянутих мов і є стандартом для створення високоефективних алгоритмів і фреймворків, як-от *TensorFlow.* Її основні сфери застосування — розробка продуктивних систем *NLP* для технологічних компаній, стартапів і науково-дослідних установ. У промисловості *C++* використовують для створення програмного забезпечення з реальночасовою обробкою даних.

R і ***MATLAB*** є основними інструментами для академічних і наукових досліджень. *R* застосовують у статистичному аналізі текстів і соціальних дослідженнях. *MATLAB* використовують в освітніх програмах і дослідженнях, що потребують математичної точності та обробки великих масивів текстових даних.

JavaScript активно застосовується в бізнесі для створення інтерактивних вебзастосунків і чатботів. Можливості мови дають змогу інтегрувати *NLP*-рішення безпосередньо у вебдодатки, що робить її популярною серед технологічних стартапів і компаній, які розробляють клієнтоорієнтовані рішення.

Ruby і ***PHP*** є зручними для розроблення невеликих вебзастосунків, орієнтованих на інтеграцію *NLP*. *Ruby* популярна у стартапах завдяки простоті використання, тоді як *PHP* знаходить застосування у веброзробленні та інтеграції *NLP* у бізнес-процеси.

Scala завдяки інтеграції з *Apache Spark* широко використовується для оброблення великих даних і розподілених обчислень. Основними сферами застосування цієї мови є фінансовий сектор, аналітика ринків і бізнес-рішення, що потребують масштабованих обчислювальних систем.

Отже, *Python* є універсальною мовою програмування для більшості *NLP*-завдань, орієнтованих на дослідження, бізнес та освіту. *Julia* та *C++* є лідерами для високопродуктивних завдань у технологічних стартапах та науці. *Java* придатна для промислових і корпоративних систем. Мови *R* і

Таблиця 16.25

Рейтинги практичного використання мов програмування за версією *ChatGPT*

Мова програмування	Продуктивність і швидкодія	Зручність у використанні	Гнучкість і можливості розширення	Ком'юніті та підтримка	Вартість впровадження	Підтримка сучасних методів *NLP*
Python	7	9	9	10	10	10
R	6	7	6	8	9	8
Java	9	6	8	9	8	7
Julia	8	8	7	6	9	7
JavaScript	7	9	9	10	10	6
C++	10	5	10	8	7	6
Ruby	6	8	7	7	8	5
PHP	5	8	6	9	8	5
MATLAB	7	6	5	6	4	4
Scala	9	7	8	8	8	7

Примітки. 1. *Продуктивність і швидкодія* оцінює ефективність виконання коду та ресурсомісткість.

2. *Зручність у використанні* ґрунтується на простоті синтаксису та навчання.

3. *Гнучкість і можливості розширення* враховують здатність адаптувати мову для різних завдань.

4. *Ком'юніті та підтримка* визначають доступність документації, бібліотек і допомоги від спільноти.

5. *Вартість впровадження* оцінює фінансову доступність інструментів і технологій.

6. *Підтримка сучасних методів NLP* відображає здатність працювати з сучасними алгоритмами та бібліотеками.

MATLAB зосереджені на дослідженнях, натомість *JavaScript, Ruby* та *PHP* використовуються для інтеграційних і спеціалізованих проєктів. *Scala* залишається лідером для розподіленого оброблення великих даних. Вибір мови програмування визначається конкретною сферою застосування, що забезпечує найкращі результати в *NLP.*

Рейтинги практичного використання мов програмування за низкою важливих критеріїв (шкала оцінювання (1—10), 1 – найменше використання; 10 – найбільше використання) наведено в табл. 16.25.

Висновки і рекомендації. Вибір мови програмування для вирішення завдань обробки природної мови *(NLP)* є важливим кроком, який визначає успіх і ефективність проєкту. Розробники мають враховувати характер завдань, доступні ресурси та вимоги до продуктивності, адже сучасні мови програмування пропонують різноманітні можливості для *NLP,* кожна з яких має свої сильні сторони і обмеження. Тобто вибір мови програмування залежить від характеру завдань. *Python* залишається універсальним вибором для досліджень, бізнесу та освіти, тоді як *Julia* та *C++* ідеальні для високопродуктивних рішень. *Java* забезпечує стабільність для корпоративних систем, а *R* і *MATLAB* корисні для вузькоспеціалізованих наукових завдань. *JavaScript, Ruby* і *PHP* спрямовані на інтеграційні та спеціалізовані завдання, натомість *Scala* забезпечує масштабованість для обробки великих даних. Оптимальний вибір мови дає можливість розробникам досягти ефективності, продуктивності та високої якості в реалізації *NLP*-рішень.

Бібліотеки для обробки природної мови. Бібліотеки для обробки природної мови є важливими інструментами для дослідників і практиків, які працюють над завданнями, пов'язаними з аналізом тексту, перекладом, аналізом тональності тощо [16.50]. Вони є набором готових до використання функцій і класів, розроблених для конкретних завдань *NLP.* Бібліотеки дають змогу фахівцям й користувачам швидко й ефективно виконувати операції токенізації, стемінгу, лематизації, векторизації тексту та аналізу тональності. У табл. 16.26 наведено порівняльну інформацію про десять найпоширеніших бібліотек, яка допомагає зробити висновки щодо сильних сторін, особливостей і відповідності різним сценаріям використання кожної з проаналізованих бібліотек.

Бібліотека ***Transformers*** від ***HuggingFace*** є найкращим вибором для завдань, що потребують точності, масштабованості та інтеграції з передовими мовними моделями — *GPT, BERT* і *RoBERTa*. Її універсальність і підтримка понад 100 попередньо навчених моделей робить цю бібліотеку ідеальною для глибокого навчання. Однак високі вимоги до апаратного забезпечення можуть обмежити її застосування в проєктах з обмеженими ресурсами. Завдяки своїй функціональності *Transformers* дає змогу значно скоротити час на розроблення складних *NLP*-систем.

Таблиця 16.26

Найпоширеніші бібліотеки для завдань *NLP*

Бібліотека	Основні завдання	Характеристики	Рейтинг популярності за версією *ChatGPT* (1—10)	Особливості і відмінності
Transformers (Hugging Face)	Робота з попередньо навченими мовними моделями (*GPT, BERT, RoBERTa* тощо)	Підтримка 100+ моделей Розмір моделей: 200 МБ — 15 ГБ Витрати *GPU:* 8+ ГБ пам'яті для великих моделей	10	Найкращий вибір для глибокого навчання Високі вимоги до апаратного забезпечення
spaCy	Токенізація, лематизація, частиномовний аналіз, розпізнавання сутностей, векторизація тексту	Підтримка 60+ мов Розмір попередньо навчених моделей: 50—500 МБ Швидкість обробки: до 50 000 токенів/с	9	Дуже швидка обробка тексту Придатна для роботи з великими обсягами даних Інтеграція з сучасними мовними моделями
Flair	Розпізнавання сутностей, частиномовний аналіз, векторизація тексту	Підтримка 25+ мов Швидкість обробки: до 20 000 токенів/с Інтеграція з векторним поданням текстових даних (*embeddings, ELMo, BERT*)	8	Оптимізована для завдань з використанням ембедингу Менший вибір моделей порівняно з *Transformers*
FastText (від Facebook)	Векторизація тексту, побудова ембедингу, текстова класифікація	Розмір моделей: 1—2 ГБ Підтримка 170+ мов Швидкість: до 5 млн слів/год	7	Швидке створення ембедингу Ефективна для текстів з рідкісними словами та субсловами
Gensim	Темомоделювання, робота з векторними просторами, побудова та аналіз *Word2Vec, Doc2Vec, FastText*	Підтримка 10+ методів векторизації Швидкість: обробка до 1 млн документів/год	7	Спеціалізується на роботі з векторними моделями Не підтримує складні завдання, як-от аналіз тональності

Закінчення табл. 16.26

Бібліотека	Основні завдання	Характеристики	Рейтинг популярності за версією *ChatGPT* (1—10)	Особливості і відмінності
Stanza (Stanford NLP)	Токенізація, лематизація, синтаксичний аналіз, розпізнавання сутностей	Підтримка 70+ мов Швидкість: обробка до 5 000 токенів/с Розмір моделей: 100—500 МБ	6	Висока якість аналізу тексту Порівняно повільна обробка великих наборів даних
TextBlob	Аналіз тональності, токенізація, переклад, генерування *N*-грам	Підтримка 10+ мов Простота у використанні Швидкість: до 10 000 токенів/с	6	Легка інтеграція для невеликих проєктів Обмежені можливості порівняно зі *spaCy* або *Transformers*
NLTK (Natural Language Toolkit)	Токенізація, стемінг, лематизація, синтаксичний аналіз, аналіз тональності	50+ модулів для базових *NLP* завдань Підтримка 25+ мов Швидкість: низька за оброблення великих даних	5	Великий набір інструментів, але обмежена швидкість Оптимальна для навчання або невеликих проєктів
OpenNLP	Токенізація, стемінг, синтаксичний аналіз, розпізнавання сутностей	Підтримка 15+ мов Розмір моделей: до 50 МБ Швидкість: середня, залежить від особливостей завдання	4	Легка інтеграція з *Java*-додатками Менше інструментів для сучасного глибокого навчання ніж у *Transformers*
Polyglot	Аналіз текстів багатьма мовами (токенізація, аналіз тональності, розпізнавання сутностей)	Підтримка 165+ мов Швидкість: обробка до 10 000 токенів/с	4	Орієнтована на багатомовні задачі Менш відома та менш документована порівняно зі *spaCy* або *Transformers*

Інша потужна бібліотека — ***spaCy*** вирізняється своєю швидкістю оброблення тексту (до 50 000 токенів за секунду) і оптимізацією для роботи з великими обсягами даних. Вона є ефективною для практичних завдань розпізнавання сутностей та частиномовного аналізу, а також забезпечує легку інтеграцію з сучасними мовними моделями. Цю бібліотеку часто обирають завдяки її продуктивності й універсальності у комерційних проєктах.

Бібліотека ***Flair***, відома своєю інтеграцією з ембедингами *ELMo* та *BERT*, спеціалізується на завданнях, що потребують високої точності, наприклад розпізнавання сутностей і частиномовний аналіз. Її перевага полягає у здатності ефективно використовувати сучасні ембединги, що забезпечує високу якість результатів. *Flair* також добре інтегрується з іншими бібліотеками, що додає їй гнучкості.

Бібліотеки ***FastText*** і ***Gensim*** мають унікальні переваги для спеціалізованих завдань. *FastText* оптимізована для створення ембедингів, враховуючи субслова, що робить її ефективною для роботи з рідкісними словами та текстами. *Gensim*, зі свого боку, забезпечує широкий вибір методів для темомоделювання та роботи з великими текстовими корпусами. Обидві бібліотеки придатні для завдань, пов'язаних із векторизацією тексту, але менш ефективні для складних завдань аналізу тексту.

Бібліотека ***Stanza*** є надійним інструментом для багатомовного аналізу, підтримуючи понад 70 мов. Висока якість обробки тексту робить її придатною для дослідницьких проєктів, хоча порівняно низька швидкість може бути обмеженням для обробки великих обсягів даних. ***Polyglot***, попри менш широку популярність, забезпечує підтримку 165 мов, що робить її унікальним вибором для багатомовних завдань. Її сильна сторона — орієнтація на аналіз тексту рідкісними мовами.

Бібліотеки ***TextBlob, NLTK*** і ***OpenNLP*** придатні для освітніх і базових завдань. *NLTK* є ключовим інструментом для навчання та створення прототипів, забезпечуючи широкий набір функцій для базового аналізу тексту. *TextBlob* вирізняється простотою використання і є корисною для аналізу тональності та генерування *N*-грам. *OpenNLP* інтегрується з *Java,* завдяки чому вона є зручною для застосування в середовищах, де ця мова програмування домінує.

Отже, вибір бібліотеки залежить від специфіки завдання. Для глибоких і складних завдань варто використовувати *Transformers,* для швидких і продуктивних рішень — *spaCy* або *Flair*. Для спеціалізованих завдань, як-от векторизація тексту, ефективні *FastText* і *Gensim*. Для багатомовного аналізу найкращими варіантами будуть *Polyglot* і *Stanza*. *NLTK, TextBlob* і *OpenNLP* залишаються важливими інструментами для базових завдань та освітніх цілей. Кожна з бібліотек має свої сильні сторони, які роблять її цінним інструментом для *NLP*-фахівців.

Фреймворки. Ці інструменти є більш комплексними порівняно з бібліотеками. Вони забезпечують структуру для розробки, тестування та розгортання *NLP*-моделей [16.51]. Фреймворки часто охоплюють кілька бібліотек і надають інструменти для оброблення даних, тренування моделей, їх оцінювання та інтеграції в готові продукти. Фреймворки є основою для ефективної роботи з *NLP*-завданнями. Залежно від складності проєкту, доступних ресурсів і мети вибір фреймворку може істотно вплинути на продуктивність та результати. *Transformers, TensorFlow* і *PyTorch* є лідерами для створення високопродуктивних моделей. Фреймворки *spaCy* та *AllenNLP* добре придатні для швидких і точних рішень, тоді як *Polyglot* і *OpenNLP* можуть бути оптимальними для освітніх або багатомовних завдань. Кожен з них має свої унікальні переваги, що робить їх цінними інструментами в арсеналі *NLP*-фахівців. У табл. 16.27 наведено десять найпоширеніших фреймворків, використовуваних для *NLP.* Рейтинг популярності фреймворків визначено за версією *ChatGPT.*

Порівнюючи таблиці 16.26 і 16.27, зазначимо, що у сучасних завданнях обробки природної мови *(NLP)* часто використовують *бібліотеки* і *фреймворки*, які забезпечують інструменти для аналізу, оброблення тексту та створення моделей [16.50, 16.51]. Однак відмінність між цими категоріями інструментів не завжди очевидна, оскільки деякі з них: *Transformers (Hugging Face), spaCy, Polyglot, Stanza, Flair, OpenNLP* та інші, мають подвійний характер і можуть виконувати функції як бібліотеки, так і фреймворка.

Бібліотеки зазвичай забезпечують набір готових функцій і класів для виконання певних завдань. Вони орієнтовані на конкретні операції — токенізацію, лематизацію, аналіз тональності чи векторизацію тексту. Наприклад, *NLTK* є класичною бібліотекою, яка пропонує широкий набір базових інструментів для оброблення тексту. Її перевага полягає у гнучкості використання: користувач самостійно визначає, як і де інтегрувати функціонал бібліотеки в проєкт. Інші приклади — *Gensim* або *FastText* — також демонструють вузькоспеціалізований підхід, зосереджуючись на створенні векторних репрезентацій тексту або аналізі текстових корпусів.

Фреймворки є комплекснішими інструментами, які надають структуру або навіть повноцінний «скелет» для розроблення моделей. Вони включають усі етапи роботи — від підготовки даних до створення і розгортання готових *NLP*-рішень. Фреймворки часто охоплюють кілька бібліотек і забезпечують автоматизацію багатьох процесів, що робить їх зручними для створення великих і складних проєктів. Наприклад, *Hugging Face Transformers* дає можливість не лише завантажувати й використовувати попередньо навчені моделі, а й виконувати їх налаштування *(fine-tuning)* та інтегрувати в готові програмні продукти. Так само *TensorFlow* і *PyTorch*, які часто розглядають як фреймворки, забезпечують не лише інструменти

Таблиця 16.27

Найпоширеніші фреймворки, які використовують для завдань *NLP*

Фреймворк	Основні завдання	Кількісні характеристики	Рейтинг популярності за версією *ChatGPT* (1—10)	Особливості і відмінності
Hugging Face Transformers	Робота з попередньо навченими моделями (*GPT, BERT* тощо), *Fine-tuning* для специфічних завдань *NLP*	100+ попередньо навчених моделей Підтримка десятків мов *GPU*-вимоги: 8—16 для великих моделей	10	Простота у використанні Широкий набір сучасних моделей Активна спільнота розробників
TensorFlow	Створення і тренування моделей глибокого навчання, зокрема для *NLP*	Підтримка великих обсягів даних *Application Programming Interface (API): Keras* для спрощення роботи *GPU*-вимоги: висока масштабованість	9	Придатний для промислових рішень Багато інструментів для відлагодження та оптимізації
PyTorch	Гнучка розробка і тренування *NLP*-моделей, придатний для експериментів	Висока швидкість навчання *GPU*-вимоги: ефективність використання пам'яті Інтеграція з різними бібліотеками (наприклад, *AllenNLP)*	9	Популярний серед дослідників Підтримка динамічного графа обчислень
AllenNLP	Моделювання та оцінювання *NLP*-завдань (розпізнавання сутностей, класифікація текстів)	Підтримка сучасних мовних моделей Плагіни для інтеграції з іншими фреймворками	8	Спеціалізований на завданнях аналізу тексту Активно використовується в дослідженнях
spaCy	Швидке оброблення текстів, токенізація, розпізнавання сутностей, частиномовний аналіз	Швидкість: до 50 000 токенів/с Моделі для 60+ мов *GPU*-вимоги: низькі	8	Придатний для комерційних проєктів Легка інтеграція з іншими *NLP*-інструментами Висока продуктивність

Закінчення табл. 16.27

Фреймворк	Основні завдання	Кількісні характеристики	Рейтинг популярності за версією *ChatGPT* (1—10)	Особливості і відмінності
FastAPI	Створення *API* для інтеграції *NLP*-моделей у вебдодатки	Швидке розгортання Низькі вимоги до ресурсів Підтримка асинхронних обчислень	7	Оптимальний для інтеграції *NLP*-рішень у виробничі середовища Легка підтримка *RESTful API*
Stanza (Stanford NLP)	Токенізація, лематизація, розпізнавання сутностей, аналіз тональності	Підтримка 70+ мов *GPU*-вимоги: середні Розмір моделей: 100—500 МБ	7	Висока якість аналізу тексту Орієнтація на багатомовний аналіз Менш оптимізований для швидкодії
Flair	Використання сучасних ембедингів *(ELMo, BERT)* для завдань *NLP*	Швидкість: до 20 000 токенів/с *GPU*-вимоги: середні Інтеграція з іншими бібліотеками	7	Сильна сторона — розпізнавання сутностей Легка інтеграція з наявними рішеннями Менш популярний ніж *Transformers*
OpenNLP	Базові *NLP*-завдання (токенізація, стемінг, синтаксичний аналіз)	Підтримка 15+ мов Розмір моделей: до 50 МБ *GPU*-вимоги: низькі	6	Інтеграція з *Java*-додатками Менш потужний для сучасних завдань *NLP* Придатний для освітніх цілей
Polyglot	Багатомовний аналіз тексту, токенізація, аналіз тональності	Підтримка 165+ мов Швидкість: до 10 000 токенів/с *GPU*-вимоги: низькі	6	Сильна сторона — багатомовний аналіз Менш відомий і документований порівняно з іншими фреймворками

для тренування моделей, а й модульну структуру, що значно спрощує завдання розробників.

Відмінність між бібліотеками і фреймворками полягає в тому, що бібліотека виконує певне завдання за запитом користувача, а фреймворк визначає загальну структуру роботи, спрямовуючи користувача в рамках певної архітектури. Однак деякі інструменти, як-от *spaCy* чи *Transformers,* можуть одночасно виконувати обидві ролі. Як бібліотеки вони надають готові функції, наприклад, для токенізації або роботи з попередньо навченими моделями. Як фреймворки вони забезпечують комплексну інфраструктуру для розроблення, тренування та розгортання *NLP*-рішень.

Це подвійне використання інструментів пояснюється їхньою гнучкістю та широким функціональним охопленням. Наприклад, *spaCy* може використовуватись як бібліотека для розпізнавання сутностей або частиномовного аналізу, але також дає змогу будувати повноцінні конвеєри обробления тексту, включно з кастомними компонентами. *Transformers* від *Hugging Face* є яскравим прикладом інструменту, який пропонує як бібліотечний підхід (завантаження готових моделей), так і фреймворковий (інтеграція, кастомізація, *fine-tuning*).

Вибір між використанням інструменту як бібліотеки чи фреймворка залежить від масштабу завдання та рівня автоматизації, необхідної для проєкту. Для вузьких і специфічних завдань, наприклад створення векторних зображень тексту або базового аналізу, бібліотеки є оптимальним вибором. Натомість для масштабних проєктів, де потрібна комплексна інтеграція кількох етапів оброблення, фреймворки пропонують структуру, що значно полегшує розроблення. Отже, кожний інструмент має певні унікальні переваги, які визначають його роль у сучасних завданнях *NLP.*

Порівняльний аналіз фреймворків (табл. 16.27) для завдань обробки природної мови допомагає краще зрозуміти, які з них придатні для різних сценаріїв використання залежно від їх сильних сторін та особливостей [16.51].

Hugging Face Transformers є лідером серед фреймворків для *NLP* завдяки простоті використання та широкому вибору попередньо навчених моделей. Цей інструмент є незамінним для задач *fine-tuning*, що робить його придатним як для дослідницьких, так і для промислових проєктів. Його підтримка десятків мов та активна спільнота розробників сприяють популярності серед фахівців. Проте високі вимоги до апаратного забезпечення обмежують використання *Hugging Face Transformers* у проєктах з низькими ресурсами.

TensorFlow має потужну інфраструктуру для створення і тренування моделей глибокого навчання. Високорівнева програмна інтерфейсна бібліотека для створення моделей машинного і глибокого навчання *(API Keras)* спрощує вирішення багатьох завдань, що робить його ідеальним для про-

єктів із великими обсягами даних. Фреймворк також має багаті можливості для оптимізації моделей. Однак *TensorFlow* може бути складним для початківців через порівняну трудомісткість навчання.

PyTorch популярний серед дослідників завдяки своїй гнучкості та підтримці динамічного графа обчислень. Він забезпечує високу швидкість навчання моделей та інтеграцію з іншими інструментами, як-от фреймворк *AllenNLP,* призначений для побудови, тренування та оцінки сучасних моделей. Фреймворк *PyTorch* придатний для експериментальних проєктів, де потрібна адаптивність. Як і *TensorFlow, PyTorch* орієнтований на більш досвідчених користувачів.

AllenNLP спеціалізується на завданнях розпізнавання сутностей і класифікації тексту. Побудований на базі іншого, потужного і гнучкого фреймворку для розроблення моделей машинного та глибокого навчання *PyTorch,* цей фреймворк активно використовуваний в академічних дослідженнях. Його особливістю є інтеграція з сучасними мовними моделями та підтримка плагінів, що додають *AllenNLP* гнучкості.

spaCy є потужним інструментом для швидкого оброблення тексту, включно з токенізацією, лематизацією та розпізнаванням сутностей. Завдяки високій продуктивності (до 50 000 токенів/с) та легкій інтеграції з іншими інструментами *spaCy* широко використовується в наукових і комерційних проєктах. Його головною перевагою є низькі вимоги до ресурсів, хоча він менш універсальний порівняно з *Transformers.*

FastAPI є ідеальним для виробничих середовищ завдяки підтримці асинхронних обчислень і низьким вимогам до ресурсів. Простота робить його зручним для створення вебінтерфейсів, що дає змогу додаткам взаємодіяти один з одним через мережу (зазвичай через Інтернет) за допомогою стандартних протоколів *RESTful API (Representational State Transfer API) для NLP.*

Stanza від ***Stanford NLP*** відзначається високою якістю багатомовного аналізу тексту. Фреймворк підтримує понад 70 мов, завдяки чому є корисним для проєктів, орієнтованих на глобальний ринок. Однак його продуктивність поступається *spaCy,* що обмежує використання у великих проєктах.

Flair спеціалізується на використанні сучасних ембедингів *ELMo* та *BERT* для високоточного розпізнавання сутностей. Інтеграція з іншими бібліотеками додає йому гнучкості, але порівняно з *Transformers* він менш популярний і не забезпечує такої широкої підтримки моделей.

OpenNLP пропонує базові інструменти для токенізації, стемінгу та синтаксичного аналізу. Простота й інтеграція з *Java*-додатками роблять його корисним для освітніх цілей або невеликих проєктів. Проте *OpenNLP* поступається більш сучасним інструментам у гнучкості та потужності.

Polyglot вирізняється підтримкою понад 165 мов, що робить його унікальним для багатомовних завдань. Однак обмежена документація і нижча

продуктивність порівняно з іншими фреймворками обмежують його використання.

Кожний з описаних фреймворків має свої сильні сторони, які роблять його придатним для певних сценаріїв використання. *Hugging Face Transformers* та *TensorFlow* придатні для складних і масштабованих проєктів. *PyTorch* і *AllenNLP* ідеальні для дослідницьких завдань. *spaCy* забезпечує високу продуктивність у комерційних проєктах, тоді як *FastAPI* спрощує інтеграцію *NLP*-моделей у вебдодатки. Для багатомовного аналізу найкращими є *Stanza* та *Polyglot*. Вибір фреймворку залежить від специфіки проєкту, доступних ресурсів і вимог до продуктивності.

Приклад. Завдання. Використовуючи один з інструментів для створення моделей машинного та глибокого навчання *API Keras (Application Programming Interface Keras)*, створити базову нейронну мережу для класифікації мовою *Python*.

Вирішення. *API Keras* — це високорівнева бібліотека, яка інтегрується з *TensorFlow,* що забезпечує її масштабованість і гнучкість для широкого спектра завдань. Вона орієнтована на те, щоб зробити процес створення нейронних мереж доступним навіть для початківців, забезпечуючи при цьому достатню функціональність для досвідчених розробників. Ця бібліотека пропонує зручний і простий у використанні синтаксис для опису моделей. Так, створення базової нейронної мережі для класифікації можна реалізувати за допомогою лише кількох рядків коду мовою *Python*:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential([
    Dense(64, activation='relu', input_shape=(784,)),
    Dense(10, activation='softmax')
])
```

Цей приклад демонструє використання *Sequential API* для створення моделі, яка складається з двох шарів: щільного шару з 64 нейронами та функцією активації *ReLU* і вихідного шару з 10 нейронами для багатокласової класифікації. Такі високорівневі абстракції дають можливість зосередитися на побудові моделі, не відволікаючись на низькорівневі технічні деталі.

Важливо зазначити, що інтеграція *Keras* з *TensorFlow* дає змогу використовувати потужність цієї платформи для роботи на *CPU, GPU* або *TPU.* Крім того, доступ до таких сучасних технологій, як згорткові нейронні мережі *(CNN),* рекурентні нейронні мережі *(RNN)* та трансформери, забезпечує актуальність *Keras* для найсучасніших досліджень у сфері машинного навчання.

Отже, *API Keras* є важливим інструментом у сучасному арсеналі розробників і дослідників. Простота у використанні, багатий функціонал і інтеграція з *TensorFlow* роблять цю бібліотеку хорошим вибором для швидкого прототипування й створення масштабованих, складних моделей глибокого навчання.

16.5. Деякі суперечки і дискусії навколо проблеми обробки природної мови

У сфері обробки природної мови точаться суперечки і дискусії, що стосуються етичних, технічних і наукових аспектів. Наведемо основні теми дискусій навколо цієї проблеми [16.52].

1. **Етичні питання та упередження стосовно моделей *NLP:***

1) ***упередженість (Bias).*** Моделі *NLP* можуть виявляти упередженість щодо раси, статі, релігії та інших соціальних аспектів, оскільки вони тренуються на великих наборах даних, які часто містять історичні та соціальні упередження. Це може призводити до дискримінаційних результатів, наприклад, під час класифікації текстів або генерування відповідей;

2) ***дискримінація та упередженість у питаннях рекрутингу.*** Використання *NLP* для автоматичного сортування резюме в процесі відбору кандидатів для прийому на роботу може призвести до відтворення соціальних упереджень, наявних у навчальних даних.

2. **Проблеми конфіденційності та безпеки:**

1) ***конфіденційність даних.*** Великі мовні моделі типу *GPT* або *BERT* можуть навчатися на величезних наборах даних, включно з конфіденційними або персональними даними. Постає питання щодо коректності використання таких даних і можливого порушення конфіденційності;

2) ***маніпуляції та фейки.*** *NLP*-моделі можуть бути використані для створення дезінформації, фейкових новин та підроблення текстів, що може мати негативні наслідки для суспільства.

3. **Відповідність результатів і пояснюваність моделей:**

1) ***моделі чорних ящиків (Black-box models).*** Багато сучасних моделей глибокого навчання, використовуваних у *NLP,* є складними для розуміння і пояснення непрофесіоналам у сфері ШІ. Це призводить до ускладнення інтерпретації результатів таких моделей, що створює проблеми для їх використання у критично важливих галузях — медицині, військовій справі або праві;

2) ***пояснюваність (Explainability).*** Зараз ведуться активні дослідження щодо того, як можна зробити моделі *NLP* більш зрозумілими і прозорими для користувачів та розробників.

4. **Якість даних та збирання даних:**

1) ***якість і репрезентативність даних.*** Зібрані дані можуть не відображати всього різноманіття мовного використання в різних соціальних і культурних групах, що може призвести до нерепрезентативних результатів;

2) ***збалансованість даних.*** Завдання побудови збалансованих наборів даних, які коректно відображають різні точки зору та позиції соціальних груп, залишається викликом для дослідників.

5. **Мовний бар'єр і мультимовність:**

1) ***проблеми мультимовності.*** Хоча англомовні моделі є найрозвиненішими, *NLP* для інших мов, особливо для мов з обмеженими ресурсами, стикається з багатьма проблемами. Це створює нерівність у доступі до технологій *NLP;*

2) ***рідкісні мови.*** Мови з малими кількостями текстових даних часто ігноруються в дослідженнях, що ускладнює розроблення *NLP*-технологій для таких мов.

6. **Проблема узагальнення та доменної адаптації:**

1) ***узагальнення моделей.*** Багато *NLP*-моделей демонструють високу продуктивність на стандартизованих тестових наборах даних, але їх здатність поширювати висновки на нові домени чи мови залишається проблематичною;

2) ***доменно-специфічні моделі.*** Є проблема з адаптацією моделей до нових доменів, де можуть бути потрібні додаткові дані для тренування або перенавчання моделі.

Ці проблеми спричинюють активні обговорення серед дослідників і практиків *NLP,* оскільки від їх вирішення залежить якість, етичність та ефективність майбутніх технологій у цій сфері.

Міжнародна професійна громадськість напрацьовує ***підходи і стратегії***, які можна запропонувати ***для подолання суперечок*** і неузгодженостей навколо проблеми обробки природної мови. Наведемо деякі ключові напрями [16.52].

1. **Зменшення упередженості в моделях *NLP:***

1) ***розроблення дебайзингових методів*** (від англ. *debiasing* — «усунення упередженості»). Активно розробляються методи зменшення упередженості в моделях, як-от перетренування на збалансованих наборах даних, застосування технік перевірки упередженості *(bias check)* та впровадження регуляризаційних технік;

2) ***відбір і збалансування навчальних даних.*** Один з підходів полягає у створенні збалансованих і репрезентативних наборів даних, які б охоплювали різні соціальні групи та точки зору, з метою мінімізації впливу упередженості;

3) ***етичний дизайн.*** Під час розроблення моделей впроваджуються принципи етичного дизайну, відповідно до яких розробники враховують можливі негативні наслідки своїх моделей та активно працюють над зменшенням упередженості.

2. **Підвищення прозорості та пояснюваності моделей:**

1) ***розроблення інструментів пояснення.*** Створюються інструменти й методики для пояснення рішень, які приймають *NLP*-моделі, що допомагає користувачам зрозуміти, як і чому було зроблено конкретне передбачення;

2) ***моделі з більшою інтерпретованістю.*** Розробляються моделі, які є менш складними і більш інтерпретованими, або здійснюється інтеграція пояснювальних шарів у складні моделі;

3) ***стандарти прозорості.*** Вводяться стандарти, які зобов'язують розробників документувати архітектуру та навчальні дані, що використовуються для створення моделей.

3. **Посилення конфіденційності та безпеки даних:**

1) ***анонімізація даних.*** Розробляються методи анонімізації даних, які дають можливість використовувати реальні дані без ризику порушення конфіденційності;

2) ***федеративне навчання.*** Моделі навчаються на даних користувачів, не передаючи ці дані на центральні сервери, що зменшує ризики порушення конфіденційності;

3) ***безпека даних.*** Здійснюється посилення заходів щодо захисту даних, які використовуються для навчання *NLP*-моделей, включно з шифруванням та безпечними методами обміну даними.

4. **Інтеграція етичних принципів у розробку *NLP:***

1) ***етичні кодекси та керування розробками.*** Багато компаній-розробників ШІ впроваджують етичні кодекси, які керують розробленням і впровадженням *NLP*-технологій, враховуючи їхній вплив на суспільство;

2) ***етичний аудит.*** Здійснюється проведення незалежного аудиту моделей та їх використання для виявлення можливих етичних ризиків і неузгодженостей;

3) ***етична освіта для розробників.*** Вводяться програми навчання, спрямовані на підвищення обізнаності серед розробників щодо етичних питань і відповідальності під час створення *NLP*-систем.

5. **Розширення підтримки різних мов і культур:**

1) ***розвиток мультимовних моделей.*** Створюються і підтримуються мультимовні моделі, які можуть працювати з мовами з обмеженими ресурсами;

2) ***залучення локальних експертів.*** Практикується співпраця з локальними експертами для збирання якісних даних та адаптації моделей до культурних особливостей різних регіонів;

3) ***підтримка меншинних мов.*** Нарощуються ініціативи зі створення ресурсів і технологій для меншинних мов, що забезпечує більш справедливий доступ до *NLP*-технологій.

6. **Узгодження моделей з правовими та регуляторними нормами:**

1) ***відповідність GDPR та іншим законам*** *(General Data Protection Regulation).* У країнах Європейського Союзу діє Загальний регламент про захист даних, який був прийнятий парламентом ЄС і набув чинності 25 травня 2018 р. Відповідно до закону *GDPR,* моделі *NLP* мають відповідати вимогам захисту даних у Європі, що містить систему правил доступу до персональних даних.

2) ***співпраця з регуляторами.*** Ведеться співпраця з урядовими органами для встановлення регуляторних норм, що керують використанням *NLP*-технологій, включно з вимогами до прозорості та етичного використання;

3) ***гармонізація законодавства України з європейськими стандартами.*** Україна не є офіційним учасником Загального регламенту про захист даних *(GDPR),* оскільки цей регламент застосовується безпосередньо в країнах ЄС. Проте у межах євроінтеграційних процесів Україна прагне гармонізувати своє законодавство із європейськими стандартами [16.53]. Зокрема, розроблено законопроєкт № 5628 «Про захист персональних даних», який має на меті привести національні норми у відповідність до вимог *GDPR* та Модернізованої конвенції. Цей законопроєкт передбачає створення незалежного наглядового органу для контролю за дотриманням норм у сфері персональних даних;

4) ***регулювання штучного інтелекту в Україні.*** У 2020 р. Кабінет Міністрів України схвалив Концепцію розвитку штучного інтелекту, яка визначає основні напрями та завдання у цій сфері [16.54]. Україна також вивчає досвід ЄС щодо регулювання ШІ, зокрема положення проєкту Регламенту ЄС про штучний інтелект *(AI Act),* який класифікує системи ШІ за рівнями ризику і встановлює відповідні вимоги [16.55]. Імплементація цих положень в українське законодавство планується поступово, з урахуванням національних особливостей і потреб [16.56].

Отже, хоча Україна формально не приєдналася до *GDPR* та інших законів ЄС про захист даних і регулювання ШІ, вона активно працює над адаптацією свого законодавства до європейських стандартів у цих сферах, що є важливим кроком на шляху до євроінтеграції.

Зазначені підходи мають на меті створення більш етичних, безпечних та ефективних *NLP*-систем, які враховують потреби користувачів і суспільства, а також сприяють зниженню ризиків, пов'язаних із неналежним використанням технологій обробки природної мови.

Висновки

Еволюція методів і моделей обробки природної мови *(NLP)* демонструє значний прогрес, починаючи від ранніх символічних підходів до сучасних систем глибокого навчання. У 1960-х рр. основою *NLP* були методи, що ґрунтувалися на правилах та граматиках. У 1990-х рр. домінували статис-

тичні методи, зокрема моделі на основі марковських ланцюгів. З початку 2010-х рр. глибоке навчання, зокрема трансформери, стало панівним підходом, що значно підвищило точність і можливості оброблення тексту.

Новітній етап розвитку *NLP,* який почався у 2017 р., ознаменувався революцією завдяки трансформерам і моделям на основі нейронних мереж. Впровадження моделей *BERT, GPT-2, GPT-3, GPT-4, RoBERTa, T5, OpenAI Whisper, Claude 3, Claude 3.5, Gemini 1.5, LLaMA-3, Mistral-7B/12B* та їх похідних радикально трансформувало можливості *NLP.* Ці моделі забезпечують високу точність у завданнях машинного перекладу, генерування текстів, створення зображень і музики, класифікації тощо. Завдяки цьому *NLP* стала ключовим інструментом у багатьох сферах людської діяльності — бізнесі, медицині, освіті та креативній індустрії.

У розробленні *NLP*-систем важливу роль відіграють мови програмування, бібліотеки та фреймворки. *Python* є основною мовою програмування для *NLP* завдяки широкому спектру бібліотек *NLTK, spaCy, Hugging Face Transformers.* Крім того, у певних сценаріях активно використовуються й інші мови: *Java, JavaScript, R, Julia, C++.* Ці інструменти надають можливості для ефективної роботи з текстовими даними, побудови моделей та інтеграції їх у масштабовані системи.

Однак проблема обробки природної мови супроводжується дискусіями щодо етичних аспектів, упередженості моделей і впливу цих явищ на суспільство. Упередження в моделях можуть призводити до дискримінаційних результатів, що викликає занепокоєння серед дослідників і користувачів. Особливу увагу привертають питання прозорості та пояснюваності моделей, що є критично важливими в автоматизованому прийнятті рішень, а також приватності даних, які використовують для навчання моделей.

Для вирішення зазначених проблем необхідно впроваджувати регуляторні норми, етичні стандарти і механізми, що сприяють прозорості та справедливості моделей *NLP.* Це охоплює розроблення методів для виявлення і зменшення упередженості, підвищення пояснюваності моделей та забезпечення захисту приватності даних. Ефективне вирішення цих проблем потребує співпраці між дослідниками, розробниками та законодавцями. Регуляторні рамки мають бути спрямовані на мінімізацію ризиків та забезпечення справедливого використання *NLP* у суспільстві.

СПИСОК ЛІТЕРАТУРИ

16.1. Weizenbaum J. ELIZA-A Computer Program for the Study of Natural Language Communication Between Man and Machine. *Communications of the ACM.* 1966. Vol. 9. No. 1. P. 36—45. https://dl.acm.org/doi/10.1145/365153.365168

16.2. Winograd T. Procedures as a Representation for Data in a Computer Program for Understanding Natural Language. *Cognitive Psychology.* 1972. Vol. 3. No. 1. P. 1—191. URL: https://www.sciencedirect.com/science/article/abs/pii/0010028572900023

16.3. Salton G., Buckley C. Term-weighting Approaches in Automatic Text Retrieval. *Information Processing & Management*. 1988. Vol. 24. No. 5. P. 513—523. URL: https://www.sciencedirect.com/science/article/abs/pii/0306457388900210

16.4. Deerwester S., Dumais S. T., Furnas G. W., Landauer T. K., Harshman R. Indexing by Latent Semantic Analysis. *Journal of the American Society for Information Science*. 1990. Vol. 41. No. 6. P. 391—407. URL: https://www.cs.bham.ac.uk/~pxt/IDA/lsa_ind.pdf

16.5. Landauer T. K., Foltz P. W., Laham D. An Introduction to Latent Semantic Analysis. *Discourse Processes*. 1998. Vol. 25. Iss. 2—3. P. 259—284. URL: https://www.tandfonline.com/doi/abs/10.1080/01638539809545028

16.6. Blei D. M., Ng A.Y., Jordan M. I. Latent Dirichlet Allocation. *Journal of Machine Learning Research*. 2003. Vol. 3. P. 993—1022. https://dl.acm.org/doi/10.5555/944919.944937

16.7. Mikolov T., Chen K., Corrado G., Dean J. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: https://arxiv.org/pdf/1301.3781

16.8. Pennington J., Socher R., Manning C. D. *GloVe: Global Vectors for Word Representation*. Stanford University, 2014. URL: https://nlp.stanford.edu/projects/glove/

16.9. Devlin J., Chang M.-W., Lee K., Toutanova K. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2018. arXiv: https://arxiv.org/abs/1810.04805

16.10. Radford A., Narasimhan K., Salimans T., Sutskever I. Improving Language Understanding by Generative Pre-Training. OpenAI, 2019. 12 p. URL: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

16.11. Raffel C., Shazeer N., Roberts A., Lee K., Narang S., Matena M. et al. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research*. 2020. Vol. 21. P. 1—67. arXiv: https://arxiv.org/pdf/1910.10683

16.12. Anthropic. *Meet Claude*. 2024. URL: https://www.anthropic.com/claude

16.13. Touvron H., Lavril T., Izacard G. et al. LLaMA: Open and Efficient Foundation Language Models. 2023. arXiv:https://arxiv.org/pdf/2302.13971

16.14. Liu Y., Ott M., Goyal N., Du J., Joshi M., Chen D. et al. RoBERTa: A Robustly Optimized BERT Pretraining Approach. 2019. arXiv: https://arxiv.org/abs/1907.11692

16.15. Zhang S., Roller S., Goyal N. et al. OPT: Open Pre-trained Transformer Language Models. Meta AI, 2022. arXiv: https://arxiv.org/pdf/2205.01068

16.16. Theverge. *Google's Gemini AI is getting faster with its Flash upgrade*. 2024. URL: https://www.theverge.com/2024/7/25/24206071/google-gemini-ai-flash-upgrade?utm_source=chatgpt.com

16.17. Anthropic. *AI research and products that put safety at the frontier*. 2023. URL: https://www.anthropic.com/

16.18. Meta AI (Facebook). *Introducing Meta Llama 3: The Most Powerful Open Available LLM to Date*. 2024. URL: https://my.ai.se/resources/2980

16.19. Jiang A. Q., Sablayrolles A., Mensch A. et al. *Mistral 7B*. 2024. URL: https://mistral.ai/assets/Mistral_7B_paper_v_0_1.pdf

16.20. OpenAI. *GPT-4*. March 14, 2023. URL: https://openai.com/index/gpt-4-research/

16.21. Woods William A. What's in a Link: Foundations for Semantic Networks. In: Bobrow D. G., Collins A. (Eds). Representation and Understanding: Studies in Cognitive Science. Academic Press, 1975. P. 35—82. URL: https://www.semanticscholar.org/paper/What's-in-a-Link%3A-Foundations-for-Semantic-Woods/1e5443f50e81f42e0cf2ad1132d60c50c0dc7f2e

16.22. ResearchGate: Robertson S. E., Walker S., Jones S., Hancock-Beaulieu M. M., Gatford M. Okapi at TREC-3. *Proceedings of the Third Text REtrieval Conference (TREC-3)*. 1995. P. 109—126. URL: https://www.researchgate.net/publication/221037764_Okapi_at_TREC-3

16.23. Hofmann T. Probabilistic Latent Semantic Indexing. *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'99)*. 1999. P. 50—57. https://dl.acm.org/doi/10.1145/312624.312649

16.24. Blei D. M., Lafferty J. D. Dynamic Topic Models. *Proceedings of the 23rd International Conference on Machine Learning (ICML'06)*. 2006. P. 113—120. https://dl.acm.org/doi/10.1145/1143844.1143859

16.25. Pennington J., Socher R., Manning C. D. GloVe: Global Vectors for Word Representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2014. Doha, Qatar. Association for Computational Linguistics. P. 1532—1543. URL: https://aclanthology.org/D14-1162.pdf

16.26. Mikolov T., Sutskever I., Chen K., Corrado G. S., Dean J. Distributed Representations of Words and Phrases and Their Compositionality. 2013. arXiv: https://arxiv.org/abs/1310.4546

16.27. Levy O., Goldberg Y. Linguistic Regularities in Sparse and Explicit Word Representations. *Proceedings of the Eighteenth Conference on Computational Natural Language Learning (CoNLL)*. 2014. P. 171—180. URL: https://aclanthology.org/W14-1620.pdf

16.28. Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A. N. et al. *Attention Is All You Need*. 2017. arXiv: https://arxiv.org/abs/1706.03762

16.29. Clark K., Luong M.-T., Le Quoc V., Manning C. D. ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators. 2020. arXiv: https://arxiv.org/abs/2003.10555

16.30. Нарожний В., Харченко В. Метод семантичної кластеризації вдосконаленого алгоритму й алгоритму BERT. 2024. No. 1 (27). С. 141—145. URL: https://www.researchgate.net/publication/381969774_Semantic_clustering_method_using_integration_of_advanced_LDA_algorithm_and_BERT_algorithm

16.31. Brown T. B., Mann B., Ryder N., Subbiah M., Kaplan J., Dhariwal P. et al. Language Models are Few-Shot Learners. 2020. arXiv: https://arxiv.org/abs/2005.14165

16.32. Wong A., Pham L., Lee Y., Chan S., Sadaya R., Khmelevsky Y. et al. *Translating Natural Language Queries to SQL Using the T5 Model*. 2023. arXiv: https://arxiv.org/html/2312.12414

16.33. Radford A., Wu J., Child R., Luan D., Amodei D., Sutskever I. Language Models are Unsupervised Multitask Learners. OpenAI, 2019. 24 p. URL: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

16.34. Zhang S., Roller S., Goyal N. et al. OPT: Open Pre-trained Transformer Language Models. Meta AI, 2022. arXiv: https://arxiv.org/pdf/2205.01068

16.35. Zhou H., Prabhumoye S., Gu J. Learning from Demonstrations for Few-Shot Prompting. 2022. arXiv: https://arxiv.org/abs/2204.05862

16.36. Bubeck S., Chandrasekaran V., Eldan R. et al. Sparks of Artificial General Intelligence: Early Experiments with GPT-4. 2023. arXiv: https://arxiv.org/abs/2303.12712?utm_source=chatgpt.com

16.37. Georgiev P., Lei V. I., Burnell R. et al. Gemini 1.5: Unlocking Multimodal Understanding Across Millions of Tokens of Context. 2024. arXiv: https://arxiv.org/abs/2403.05530?utm_source=chatgpt.com

16.38. Enis M., Hopkins M. From LLM to NMT: Advancing Low-Resource Machine Translation with Claude. 2024. arXiv: https://arxiv.org/abs/2404.13813?utm_source=chatgpt.com

16.39. Anthropic. *Claude 3.5 Sonnet*. 21.06.2024. URL: https://www.anthropic.com/news/claude-3-5-sonnet?utm_source=chatgpt.com

16.40. Grattafiori A., Dubey A., Jauhri A. et al. The Llama 3 Herd of Models. 2024. arXiv: https://arxiv.org/abs/2407.21783?utm_source=chatgpt.com

16.41. Huggingface. *Meta-llama/Llama-3.1-405B*. 2024. URL: https://huggingface.co/meta-llama/Llama-3.1-405B?utm_source=chatgpt.com

16.42. Jiang A. Q., Sablayrolles A., Mensch A. et al. Mistral 7B. October 2023. arXiv: https://arxiv.org/abs/2310.06825?utm_source=chatgpt.com

16.43. Mistral.ai. *Mistral NeMo: 12B Model Built in Collaboration with NVIDIA*. July 18, 2024. URL: https://mistral.ai/news/mistral-nemo/?utm_source=chatgpt.com

16.44. Build.nvidia. *Most Advanced Language Model for Reasoning, Code, Multilingual Tasks; Runs on a Single GPU*. 2024. URL: https://com/nv-mistralai/mistral-nemo-12b-instruct/projects

16.45. Aitechtrend.com. *TPU Vs GPU Vs CPU: Which Hardware Should You Choose for Deep Learning?* March 1, 2023. URL: https://aitechtrend.com/tpu-vs-gpu-vs-cpu-which-hardware-should-you-choose-for-deep-learning/

16.46. Singh R. CPU vs GPU vs TPU vs NPU: Key Differences and Use Cases. *Appscribed.com*. December 11, 2024. URL: https://appscribed.com/cpu-vs-gpu-tpu-npu-differences-comparison-use-cases/?utm_source=chatgpt.com

16.47. Jurafsky D., Martin J. H. Speech and Language Processing (3rd ed. draft). 2023. URL: https://web.stanford.edu/~jurafsky/slp3/

16.48. Bender E. M., Gebru T. et al. On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? 2021. URL: https://dl.acm.org/doi/10.1145/3442188.3445922

16.49. Research.google/blog/. *The Latest Research from Google*. 2024. URL: https://research.google/blog/

16.50. Tech with Herschel. *Understanding Programming Language Quality: Criteria, Standards, and Benchmarks*. 2024. URL: https://techwithherschel.com/understanding-programming-language-quality-criteria-standards-and-benchmarks/

16.51. Nlpcloud.com. *Топ-10 фреймворків, сервісів та гравців у сфері обробки природної мови у 2022 році*. 21 березня 2022 року. URL: https://nlpcloud.com/uk/top-10-nlp-frameworks-services-2022.html?utm_source=chatgpt.com

16.52. Mehrabi N., Morstatter F. et al. A Survey on Bias and Fairness in Machine Learning. 2021. https://dl.acm.org/doi/10.1145/3442188.3445922

16.53. Humanitarian Media Hub. *Реформування законодавства України у сфері захисту персональних даних — ключ до євроінтеграції*. 28 листоп. 2024. URL: https://hmh.news/8026/reformuvannya-zakonodavstva-ukrayiny-u-sferi-zahystu-personalnyh-danyh-klyuch-do-yevrointegracziyi/?utm_source=chatgpt.com

16.54. Golaw.ua. *Правове регулювання штучного інтелекту в Україні та світі*. 3 лют. 2022. URL: https://golaw.ua/ua/insights/publication/pravove-regulyuvannya-shtuchnogo-intelektu-v-ukrayini-ta-sviti/?utm_source=chatgpt.com

16.55. Pravo.ua. *Штучний інтелект: проблеми та перспективи правового регулювання в Україні та ЄС*. 15 серп. 2023. URL: https://pravo.ua/shtuchnyi-intelekt-problemy-ta-perspektyvy-pravovoho-rehuliuvannia-v-ukraini-ta-ies/?utm_source=chatgpt.com

16.56. Січкар А. Євроінтеграція: регулювання штучного інтелекту в Україні та ЄС. *Jurliga.ligazakon.net*. 11 черв. 2024. URL: https://jurliga.ligazakon.net/analitycs/228365_vrontegratsya-regulyuvannya-shtuchnogo-ntelektu-v-ukran-ta-s?utm_source=chatgpt.com

Післямова

Сучасна епоха цифрових технологій дедалі більше орієнтується на штучний інтелект як один із ключових рушіїв наукового і технічного прогресу. У цьому контексті підручник «Системи і методи штучного інтелекту» слугує методологічним ресурсом для студентів, викладачів і дослідників, які прагнуть глибше опанувати концептуальні основи, алгоритми та методи застосування ШІ в різних сферах науки й техніки. Видання насамперед орієнтоване на студентів технічних університетів, які навчаються за спеціальностями «Комп'ютерні науки» та «Системний аналіз і науки про дані». Водночас його зміст виходить за межі академічного курсу, формуючи базу для фундаментальних і прикладних досліджень у галузі обчислювального інтелекту.

Однією з ключових цілей підручника є глибокий аналіз нейронних мереж, що охоплює як класичні архітектури багатошарових перцептронів, так і сучасні моделі глибокого навчання. У першому розділі розглядаються методи навчання, починаючи з базового алгоритму зворотного поширення помилки та його варіацій, включаючи прискорені градієнтні методи, алгоритми спряжених градієнтів і генетичні підходи. Значну увагу приділено нейронним мережам із радіально-базисними функціями, їхній адаптації та застосуванню в різних класах завдань. У другому розділі проаналізовано мережі самоорганізації, зокрема алгоритм навчання Кохонена, а також мережі зі зворотними зв'язками, такі як мережі Хопфілда й Хеммінга, що застосовуються у задачах розпізнавання образів і оптимізації.

Третій розділ присвячений нечіткій логіці та її інтеграції з методами нейронних мереж. Наведено класичні підходи до нечіткого логічного висновку, зокрема алгоритми Мамдані, Цукамото, Сугено і Ларсена, які широко використовуються в системах керування, прогнозування та аналізу великих обсягів даних. Значне місце відведено нечітким нейронним мережам, як-от архітектури *ANFIS (Adaptive Neuro-Fuzzy Inference System)* і *TSK (Takagi — Sugeno — Kang)*, що поєднують переваги нечітких систем із потужністю нейромережевого навчання. У четвертому розділі розглянуто гібридні моделі, які використовують метод групового урахування аргументів (МГУА), що дають змогу створювати адаптивні системи аналізу інформації.

П'ятий і шостий розділи присвячені розгляду традиційних та сучасних алгоритмів інтелектуального аналізу даних і кластеризації. Розглядаються методи опорних векторів *(Support Vector Machine, SVM)*, дерев рішень, алгоритмів *Random Forest* та інші моделі, що застосовуються для класифікації та прогнозування. Значну увагу приділено кластерному аналізу, зокрема як ієрархічним, так і нечітким методам кластеризації, що використовуються для виявлення прихованих закономірностей у великих масивах даних.

Особливий інтерес становлять сьомий і восьмий розділи, присвячені еволюційним алгоритмам та методам ройового інтелекту. Генетичні алгоритми, диференційна еволюція та алгоритми мурашиних колоній розглядаються у контексті їх використання для оптимізації та моделювання складних процесів. Такі методи застосовуються у задачах пошуку оптимальних рішень у багатовимірних просторах, зокрема в задачах оптимізації нейромережевих структур та розв'язанні комбінаційних задач, як-от задача комівояжера.

У дев'ятому розділі підручника розглянуто глибокі нейронні мережі, включаючи згорткові, рекурентні та трансформерні архітектури, які сьогодні визначають розвиток штучного інтелекту. Значну увагу приділено питанням регуляризації глибокого навчання, методам *Dropout*, ансамблевому навчанню та використанню сучасних оптимізаційних алгоритмів — *Adam (Adaptive Moment Estimation)* і *RMSprop (Root Mean Square Propagation)*.

Десятий розділ присвячений розвитку методів глибокого навчання та гібридних мереж на основі самоорганізації, зокрема методу МГУА. Детально описані алгоритми глибокого навчання із використанням різних моделей нейронів, зокрема нейронів Ванга — Менделя та нео-фаззі нейронів. Наведено результати численних експериментів із синтезу структури таких мереж та їх застосування у прогнозуванні на фондових ринках, а також порівняння алгоритмів глибокого навчання з класичними нейронними мережами.

В одинадцятому розділі підручника зроблено акцент на спеціальному класі мереж глибокого навчання — згорткових нейромережах (ЗНН) та їх застосуванні у задачах обробки зображень. Розглянуто основні шари ЗНН, їх моделі та архітектури сучасних згорткових мереж, охарактеризовано їхні властивості. Докладно описано методи навчання згорткових нейронних мереж, механізми уваги, а також сучасні архітектури — *ResNet, Inception* та *Xception*. Дванадцятий розділ присвячений дослідженню згорткових та гібридних нейронних мереж у задачах медичної діагностики на основі аналізу медичних зображень. Розглянуто завдання аналізу пухлин молочної залози та алгоритми розробки нових архітектур гібридних ЗНН. Оцінено їхню ефективність у діагностиці раку молочної залози та порівняно їх із класичними ЗНН. Тринадцятий розділ акцентує увагу на актуальній проблемі діагностики пухлин легенів та розробленні нових методів і архітектур ЗНН для класифікації раку легенів.

Чотирнадцятий — шістнадцятий розділи присвячені застосуванню штучного інтелекту в обробці природної мови *(Natural Language Processing, NLP)*, що є однією з найдинамічніших галузей сучасного ШІ. Висвітлено основні тенденції NLP, починаючи з класичних статистичних методів і закінчуючи трансформерними моделями, що є основою сучасних мовних моделей, таких як *GPT (Generative Pre-trained Transformer), BERT (Bidirectional Encoder Representations from Transformers)* та інших. Окрему увагу приділено еволюції мовних технологій від перших алгоритмів синтаксичного аналізу до сучасних нейромережевих систем, а також викликам, пов'язаним з упередженнями, обмеженнями ресурсів і тлумачністю моделей.

Матеріал, викладений у підручнику, відповідає актуальному стану розвитку штучного інтелекту у період 2024—2025 рр. Водночас стрімкий розвиток у сфері ШІ та досліджень складних систем зумовлює трансформацію сучасних технологічних і соціальних парадигм. Еволюція ШІ характеризується переходом від традиційних алгоритмічних підходів до побудови прозорих, адаптивних та відповідальних систем, здатних до інтерпретації власних рішень і функціонування в умовах невизначеності. Простежується тенденція до заміщення вузькоспеціалізованих моделей гнучкими когнітивними архітектурами, що забезпечують узагальнення знань і контекстну адаптацію.

Значного поширення набувають квантові алгоритми, які відкривають нові можливості у вирішенні задач підвищеної складності, зокрема в оптимізації, моделюванні та обробці великих обсягів даних. Генеративні моделі стають інструментом проривних змін у соціальних науках, освітньому процесі, охороні здоров'я, промисловості й наукових дослідженнях. Посилюється роль обчислювальної біології та прогнозних аналітичних систем, що формують основу для нової якості наукового моделювання.

Імітаційні середовища та цифрові двійники відіграють ключову роль у глибокому аналізі соціальних і техногенних процесів, забезпечуючи інструментарій для ухвалення стратегічно виважених рішень. У галузях оборони та кібербезпеки пріоритетного значення набуває розвиток автономних інтелектуальних агентів, здатних до ситуаційного усвідомлення, адаптивної поведінки та мінімізації ризиків, пов'язаних із людським фактором.

Дослідження складних систем дедалі частіше орієнтуються на побудову моделей глобальних ризиків і формування інтегрованих середовищ підтримки прийняття рішень. Аналіз соціотехнічних взаємодій слугує основою для розроблення стійких економічних і соціальних стратегій. Прогнозні технології як інструмент попередження кризових сценаріїв набувають особливої ваги у забезпеченні сталого розвитку.

Актуальним викликом є створення самонавчальних і саморегульованих систем, що поєднують автономність з відповідальністю. Їх впровадження у

стратегічне планування, управління ресурсами та антикризове реагування відкриває нові можливості для ефективного адаптивного управління змінами у суспільстві.

Майбутнє штучного інтелекту пов'язане з перспективою розроблення штучного загального інтелекту *(Artificial General Intelligence, AGI)* — системи, здатної виконувати будь-яке когнітивне завдання на рівні або вище людського. На відміну від вузькоспеціалізованого ШІ *(Artificial Narrow Intelligence, ANI) AGI* передбачає універсальність, здатність до самостійного навчання та генерації нових знань. Очікується, що *AGI* стане основним рушієм у розв'язанні фундаментальних наукових проблем у галузях фізики, біології, енергетики, забезпечуючи технологічні прориви світового масштабу.

У середньо- та довгостроковій перспективі людська і механічна праця поступово втрачатиме свою домінантну роль, а функції управління ресурсами й критичними інфраструктурами переходитимуть до *AGI*-систем. У цьому контексті контроль над *AGI* розглядається як ключовий виклик для міжнародної наукової спільноти: за відсутності адекватних етичних і регуляторних механізмів існує небезпека технологічної дестабілізації або втрати керованості над складними автоматизованими структурами.

У пропонованому підручнику зроблена спроба врахувати зазначені тенденції розвитку ШІ. Книга поєднує класичні методології та новітні досягнення у сфері ШІ, що є важливим чинником якісної підготовки студентів, аспірантів і дослідників, орієнтованих на наукову та інженерну діяльність. Окрім фундаментального викладу теоретичних основ, видання пропонує практичні підходи до розроблення інноваційних рішень, що формують нове покоління інтелектуальних систем.

Зміст

ПЕРЕЛІК СКОРОЧЕНЬ 3
ПЕРЕДМОВА 7
ВСТУП 18

Розділ 1
НЕЙРОННІ МЕРЕЖІ

1.1. Нейронна мережа зі зворотним поширенням. Архітектура, функції 47
1.2. Градієнтний метод навчання нейронної мережі зі зворотним поширенням ... 51
1.3. Градієнтний алгоритм навчання для мережі з довільним числом шарів 52
1.4. Удосконалення градієнтного алгоритму навчання нейронної мережі BP 55
1.5. Прискорення збіжності алгоритмів навчання нейронних мереж. Алгоритм спряжених градієнтів 58
1.6. Генетичний алгоритм навчання нейронної мережі 60
1.7. Нейронні мережі з радіально-базисними функціями (РБФ-мережі) 62
Висновки 84
Список літератури 85

Розділ 2
НЕЙРОННІ МЕРЕЖІ ЗІ ЗВОРОТНИМИ ЗВ'ЯЗКАМИ І САМООРГАНІЗАЦІЄЮ

2.1. Нейронна мережа Хопфілда та її застосування 89
2.2. Нейронна мережа Хеммінга. Архітектура і алгоритм роботи 99
2.3. Самоорганізаційні (самоорганізовані) нейронні мережі. Алгоритми навчання Кохонена 104
2.4. Застосування нейронних мереж Кохонена 114

Висновки 123
Список літератури 124

Розділ 3
СИСТЕМИ НЕЧІТКОГО ЛОГІЧНОГО ВИСНОВКУ, НЕЧІТКІ НЕЙРОННІ МЕРЕЖІ ТА ЇХ ЗАСТОСУВАННЯ В ЗАДАЧАХ ПРОГНОЗУВАННЯ

3.1. Алгоритми нечіткого логічного висновку 127
3.2. Методи приведення до чіткості 135
3.3. Теореми про універсальну апроксимацію 136
3.4. Нечіткі нейронні мережі з висновком Мамдані і Цукамото 138
3.5. Градієнтний алгоритм навчання ННК Мамдані і Цукамото 140
3.6. Нечітка нейронна мережа *ANFIS* 142
3.7. Нечіткі нейронні мережі *TSK* та Ванга — Менделя 147
3.8. Каскадна нечітка (нео-фаззі) нейронна мережа 160
3.9. Експериментальні дослідження каскадної нечіткої (нео-фаззі) мережі 172
Висновки 174
Список літератури 228

Розділ 4
МЕТОД ІНДУКТИВНОГО МОДЕЛЮВАННЯ В ЗАДАЧАХ ІНТЕЛЕКТУАЛЬНОГО АНАЛІЗУ ДАНИХ

4.1. Загальна характеристика та основні принципи МГУА 179
4.2. Нечіткий МГУА. Основні ідеї. Математична модель НМГУА 183
4.3. Опис алгоритму НМГУА 184
4.4. НМГУА з гауссовими та дзвоноподібними функціями належності 185
4.5. Нечіткий МГУА з різними видами часткових описів 188
4.6. Адаптація коефіцієнтів лінійної інтервальної моделі 192
4.7. Порівняльний аналіз МГУА та нечіткого МГУА 196
Висновки 201
Список літератури 202

Розділ 5
НЕЧІТКІ НЕЙРОННІ МЕРЕЖІ В ЗАДАЧАХ КЛАСИФІКАЦІЇ

5.1. ННМ *NEFClass*: архітектура, властивості, алгоритм навчання бази правил і параметрів ФН 207

5.2. Аналіз властивостей *NEFClass*. Модифікована система нечіткої класифікації *NEFClass-M* 211

5.3. Експериментальні дослідження ННМ *NEFClass* 214

5.4. Розпізнавання об'єктів електрооптичних зображень з використанням нечітких НМ 215

5.5. Застосування системи *NEFClass* у завданні розпізнавання об'єктів електрооптичних зображень 220

5.6. Експерименти з розпізнавання об'єктів на електрооптичних зображеннях 222

5.7. Метод опорних векторів 230

5.8. Алгоритм налаштування *SVM* 242

5.9. Застосування методу опорних векторів 245

5.10. Метод класифікації *Random Forest* 249

Висновки 251

Список літератури 252

Розділ 6
КЛАСТЕРНИЙ АНАЛІЗ В ІНТЕЛЕКТУАЛЬНИХ СИСТЕМАХ

6.1. Кластер-аналіз. Постановка задачі. Критерії якості та метрики кластер-аналізу 258

6.2. Класифікація алгоритмів кластерного аналізу 261

6.3. Нечіткий метод *k*-середніх 267

6.4. Визначення початкового розміщення центрів кластерів 269

6.5. Алгоритм нечіткого кластер-аналізу Густавсона — Кесселя 271

6.6. Застосування нечітких методів *k*-середніх і Густавсона — Кесселя в задачах автоматичної класифікації 272

6.7. Адаптивні робастні алгоритми кластеризації 289

Висновки 297

Список літератури 298

Розділ 7
ГЕНЕТИЧНІ АЛГОРИТМИ ТА ЕВОЛЮЦІЙНЕ МОДЕЛЮВАННЯ

7.1. Генетичні алгоритми 303
7.2. Застосування генетичного алгоритму для задачі структурного синтезу мереж 322
7.3. Еволюційне програмування 326
7.4. Диференціальна еволюція 339
7.5. Самонавчання ДЕ 344
7.6. Знаходження найкоротшого шляху на графі на основі ГА 345
Висновки 353
Список літератури 354

Розділ 8
РОЙОВІ АЛГОРИТМИ ОБЧИСЛЮВАЛЬНОГО ІНТЕЛЕКТУ

8.1. Загальна характеристика кооперативної поведінки в ройових алгоритмах ... 359
8.2. Ройова оптимізація 360
8.3. Вибір параметрів алгоритмів РО 366
8.4. Гібридні алгоритми РО 368
8.5. Мурашині алгоритми 370
8.6. Алгоритми колоній мурах 382
8.7. Застосування мурашиних алгоритмів у задачі комівояжера 391
8.8. Аналіз результатів експериментів 395
Висновки 401
Список літератури 402

Розділ 9
ГЛИБОКІ НЕЙРОННІ МЕРЕЖІ. АЛГОРИТМИ ГЛИБОКОГО НАВЧАННЯ

9.1. Автоасоціатори. Автокодувальники 407
9.2. Машини Больцмана (БМ) 411
9.3. Алгоритм навчання «контрастна дивергенція» (КД-*k*) 416

9.4. Стекові мережі автоасоціаторів 417

9.5. Глибоке мережеве навчання 418

9.6. Алгоритми оптимізації функції вартості в машинному навчанні для НМ 419

9.7. Регуляризація глибокого навчання 422

Висновки 428

Список літератури 429

Розділ 10

ГІБРИДНІ НЕЧІТКІ МЕРЕЖІ ГЛИБОКОГО НАВЧАННЯ, АЛГОРИТМИ НАВЧАННЯ ТА ОПТИМІЗАЦІЇ

10.1. Архітектура нейронечіткої системи МГУА, що еволюціонує 436

10.2. Нейронечітка мережа з малою кількістю параметрів налаштування як вузол МГУА-системи 437

10.3. Експериментальні дослідження гібридних нейронечітких мереж 439

10.4. Дослідження гібридних неонечітких НМ на основі МГУА та оптимізація структури в задачах прогнозування 442

10.5. Оптимізація структури та дослідження гібридних неонечітких нейронних мереж МГУА в задачах прогнозування 446

10.6. Неонечіткі системи МГУА з розширеними неонечіткими нейронами 453

10.7. Гібридна система обчислювального інтелекту на основі пакетного та групового оброблення даних 458

Висновки 468

Список літератури 469

Розділ 11

ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ

11.1. Поняття згортки 475

11.2. Загальні особливості згорткових нейронних мереж, шари ЗНМ та їх функції 476

11.3. Архітектури згорткових нейронних мереж 483

Висновки 490

Список літератури 490

Розділ 12
ЗГОРТКОВІ І ГІБРИДНІ НЕЙРОННІ МЕРЕЖІ У ЗАВДАННЯХ АНАЛІЗУ МЕДИЧНИХ ЗОБРАЖЕНЬ ТА КЛАСИФІКАЦІЇ ПУХЛИН МОЛОЧНОЇ ЗАЛОЗИ

12.1. Опис набору даних *BreakHis* 496
12.2. Вибір архітектури згорткової нейронної мережі 498
12.3. Вибір гіперпараметрів тренувальних моделей 500
12.4. Результати бінарної класифікації пухлин молочної залози 501
12.5. Результати багатокласової класифікації зображень 502
12.6. Гібридна ЗНМ ННМ для класифікації пухлин молочної залози 505
12.7. Гібридна згорткова нейронна мережа на основі автокодувальника для виявлення раку молочної залози 510
Висновки 518
Список літератури 519

Розділ 13
СИСТЕМА ВИЯВЛЕННЯ РАКУ ЛЕГЕНІВ НА ОСНОВІ ЗАСТОСУВАННЯ СТРАТЕГІЙ ГЛИБОКОГО НАВЧАННЯ

13.1. Двовимірні згорткові нейронні мережі із багаторазовим навчанням 523
13.2. Рекурентні нейронні мережі з механізмом уваги та 2D-ЗНМ 525
13.3. Тривимірна згорткова нейронна мережа для виявлення раку легенів 537
13.4. Набори даних та їх попереднє оброблення 544
13.5. Експериментальні дослідження і аналіз результатів 545
Висновки 548
Список літератури 549

Розділ 14
СИСТЕМИ ОБРОБКИ ПРИРОДНОЇ МОВИ

14.1. Базові засади обробки природної мови 555
14.2. Головні завдання і сфери застосування *NLP* 560

14.3. Як працюють моделі обробки природної мови? 578
Висновки 592
Список літератури 593

Розділ 15
МЕТОДИ І МОДЕЛІ ОБРОБКИ ПРИРОДНОЇ МОВИ

15.1. Традиційні методи і моделі машинного навчання 598
15.2. Методи і моделі глибокого навчання в обробці природної мови 617
15.3. Методи глибокого навчання з підкріпленням 627
Висновки 633
Список літератури 634

Розділ 16
ДЕЯКІ ПРАКТИЧНІ АСПЕКТИ ЗАСТОСУВАННЯ СИСТЕМ ОБРОБКИ ПРИРОДНОЇ МОВИ

16.1. Еволюція систем обробки природної мови у період 1964—2024 рр. 639
16.2. Найпоширеніші класи мовних моделей 647
16.3. Нове покоління *NLP*-моделей, придатних для вирішення складних і ресурсомістких завдань 678
16.4. Апаратне і програмне забезпечення машинного навчання мовних моделей 698
16.5. Деякі суперечки і дискусії навколо проблеми обробки природної мови 724
Висновки 727
Список літератури 728

ПІСЛЯМОВА 732

Zgurovsky M.Z., Zaychenko Yu.P. Artificial Intelligence Systems and Methods / National Academy of Sciences of Ukraine, Educational and Scientific Complex «Institute of Applied SystemsAnalysis», Igor Sikorsky Kyiv Polytechnic Institute. — Kyiv: Akademperiodyka, 2025. — 744 p.

ISBN 978-966-360-551-7

The textbook provides a systematic overview of modern methods and technologies in artificial and computational intelligence, with a focus on their practical applications in various areas of human activity. It outlines the specific features of each field and highlights their differences in problem-solving approaches, making them complementary. Special attention is given to a detailed analysis of neural networks, including their architectures, training algorithms, and key theoretical aspects, supported by practical examples. The textbook also covers fuzzy logic methods and neural networks based on them, which are used effectively for forecasting and analyzing the behavior of complex systems. Additionally, it examines advancements in natural language processing and the evolution of this field from the 1960s to the present day. The textbook is intended for a wide range of students, postgraduates, and professionals who study and apply artificial intelligence in their professional activities.

Наукове видання

НАЦІОНАЛЬНА АКАДЕМІЯ НАУК УКРАЇНИ
НАВЧАЛЬНО-НАУКОВИЙ КОМПЛЕКС
«ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ»
КПІ ім. ІГОРЯ СІКОРСЬКОГО

ЗГУРОВСЬКИЙ Михайло Захарович
ЗАЙЧЕНКО Юрій Петрович

СИСТЕМИ І МЕТОДИ ШТУЧНОГО ІНТЕЛЕКТУ

Редактори *Л.Є. Канівець, О.М. Чадюк*

Художнє оформлення *Є.О. Ільницького*
Технічне редагування та виготовлення ілюстрацій *О.А. Бурдік*
Комп'ютерна верстка *О.В. Вакаренко*

Підп. до друку 29.09.2025. Формат 70 × 100/16.
Гарн. Minion Pro. Ум. друк. арк. 60,45. Обл.-вид. арк. 55,46.
Тираж 150 прим. Зам. № 7804.

Видавець і виготовлювач
Видавничий дім «Академперіодика» НАН України
01024, Київ, вул. Терещенківська, 4

Свідоцтво про внесення до Державного реєстру суб'єктів
видавничої справи серії ДК № 544 від 27.07.2001 р.

ВИДАВНИЧИЙ ДІМ АКАДЕМ ПЕРІОДИКА