



Міністерство освіти і науки України
Сумський державний університет

Харченко В. О.

МОДЕЛЮВАННЯ НЕЙРОННИХ МЕРЕЖ

Навчальний посібник

Рекомендовано вченою радою Сумського державного університету

Суми
Сумський державний університет
2024

УДК 004.032.26:001.103(075.8)(0.034)

X 22

Рецензенти:

О. В. Лисенко – доктор фізико-математичних наук, професор, професор кафедри прикладної математики та моделювання складних систем Сумського державного університету;

Н. О. Красношлик – кандидат технічних наук, доцент, доцент кафедри прикладної математики та інформатики Навчально-наукового інституту інформаційних та освітніх технологій Черкаського національного університету ім. Богдана Хмельницького

Рекомендовано до видання
вченою радою Сумського державного університету
як навчальний посібник
(протокол № 8 від 8 лютого 2024 року)

Харченко В. О.

X 22 Моделювання нейронних мереж : навч. посіб. /
В. О. Харченко. – Суми : Сумський державний університет, 2024. – 263 с.

У навчальному посібнику наведено теоретичні основи, методи та алгоритми моделювання нейронних мереж, приклади їх застосування й завдання для виконання практичних робіт із типовими прикладами подання результатів. Видання укладене в співробітництві із Сумською ІТ-компанією CPSCS.

Рекомендований для здобувачів закладів вищої освіти спеціальностей «Прикладна математика» та «Комп'ютерні науки».

УДК 004.032.26:001.103(075.8)(0.034)

© Сумський державний університет, 2024

Зміст

Розділ 1. Основні поняття теорії нейронних мереж. Нейромережеві моделі	9
Тема 1. Біологічний нейрон та його математична модель.	
Штучний нейрон. Активаційна функція	9
1.1. Біологічний нейрон	9
1.1.1. Структурна класифікація нейронів	11
1.1.2. Типи нейронів	12
1.1.3. Передавання нервового імпульсу	13
1.1.4. Класифікація нейромедіаторів	15
1.2. Штучний нейрон	16
1.2.1. Розширена модель штучного нейрона	17
1.3. Компоненти штучного нейрона	20
Тема 2. Архітектура, структура та класифікація штучних нейронних мереж	24
2.1. Архітектура з'єднань штучних нейронів	24
2.2. Структура нейронної мережі	25
2.3. Класифікація нейронних мереж	28
Тема 3. Навчання штучної нейронної мережі	30
3.1. Навчання з учителем	31
3.2. Навчання без учителя	33
3.3. Навчання з частковим залученням учителя	34
3.4. Навчання з підкріпленням	35
3.5. Перенавчання нейронної мережі	37
Тема 4. Персептрон. Навчання персептрону.	
Лінійна роздільність і персептронне подання	38
4.1. Персептрон Розенблата	38
4.2. Навчання персептрона	40
4.3. Геометричне подання алгоритму навчання	42
4.4. Лінійна роздільність	45
4.5. Сигмоїдальний нейрон	47
4.6. Приклад застосування одношарового персептрона та сигмоїдального нейрона	49

Розділ 2. Методи навчання багат шарових нейронних мереж	51
Тема 5. Багат шарові нейронні мережі прямого поширення. Багат шаровий перцептрон. Задача «XOR»	51
5.1. Багат шарова нейронна мережа	51
5.1.1. Лінійна функція	51
5.1.2. Сигмоїдна функція	53
5.1.3. Результуюча функція	53
5.1.4. Функція витрат	54
5.1.5. Пряме поширення	54
5.2. Дво шаровий перцептрон	56
5.2.1. Задача лінійної нероздільності класів	56
5.2.2. Дво шаровий перцептрон для задачі «XOR»	57
Тема 6. Метод градієнтного спуску та алгоритм зворотного поширення помилки.	61
6.1. Метод градієнтного спуску	61
6.1.1. Поверхня помилок	61
6.1.2. Ітеративна процедура пошуку мінімуму	62
6.1.3. Збурення й відхилення від локальної рівноваги	64
6.2. Алгоритм зворотного поширення помилки	66
6.2.1. Пряме поширення сигналу мережею	66
6.2.2. Зворотне поширення для багат шарової мережі з одним нейроном у кожному шарі	69
6.2.3. Мережа з декількома нейронами в кожному шарі	72
6.2.4. Оновлення вагових коефіцієнтів	74
6.2.5. Локальний градієнт нейронів кожного шару	75
Тема 7. Модифікації методу зворотного поширення. Способи навчання. Збіжність. Перенавчання.	81
7.1. Модифікації алгоритму зворотного поширення помилки	81
7.1.1. Інерційний алгоритм	82
7.1.2. Зменшення вагових коефіцієнтів	83
7.1.3. Відкладене навчання	84
7.1.4. Метод RProp	84
7.2. Способи навчання багат шарових нейронних мереж	85
7.3. Гіперпараметри та збіжність мережі	86
7.4. Перенавчання нейронної мережі	87

Тема 8. Приклади задач із багат шаровими нейронними мережами прямого поширення	90
8.1. Апроксимація заданої функції	90
8.1.1. Постановка задачі	90
8.1.2. Кроки реалізації	90
8.1.3. Приклад програмної реалізації	93
8.2. Навчання агента грати в «пінг-понг»	97
8.2.1. Постановка задачі	97
8.2.2. Реалізація гри	98
8.3. Розпізнавання рукописних цифр	101
8.3.1. Постановка задачі	101
8.3.2. Етапи навчання та тестування	101
8.3.3. Код алгоритму для відцентрування та переформатування будь-якого зображення	107
Тема 9. Автоматизація навчання нейромереж з використанням пакета TensorFlow	109
9.1. Початок роботи	109
9.2. Дослідження даних	111
9.3. Побудова моделі	112
9.4. Скомпілювання моделі	114
9.5. Навчання моделі	114
9.6. Оцінка точності	115
9.7. Прогнозування	115

Розділ 3. Асоціативна пам'ять, класифікація без учителя та кластеризація 120

Тема 10. Організація асоціативної пам'яті за допомогою мережі Хопфілда	120
10.1. Задача класифікації мережею Хопфілда	120
10.2. Структура мережі Хопфілда	122
10.3. Навчання мережі Хопфілда	123
10.4. Стійкість мережі	125
10.5. Приклади роботи мережі Хопфілда	126
10.5.1. Один вектор для запам'ятовування	126
10.5.2. Робота мережі з декількома еталонними векторами	128

Тема 11. Використання мережі Хемінга для класифікації .	131
11.1. Архітектура нейронної мережі Хемінга	131
11.2. Прихований шар Хемінга	133
11.3. Прихований шар MAXNET	136
11.4. Принцип роботи тришарової нейронної мережі Хемінга	138
11.5. Застосування мережі Хемінга для класифікації цифр	140
Тема 12. Нейронна мережа <i>Mexican Hat</i> для збільшення контрастності зображень	146
12.1. Архітектура нейронної мережі <i>Mexican Hat</i>	147
12.2. Алгоритм функціонування мережі <i>Mexican Hat</i>	147
12.3. Застосування мережі <i>Mexican Hat</i>	149
Тема 13. Нейронна мережа Кохонена в задачах кластеризації	152
13.1. Архітектура мережі Кохонена	152
13.2. Навчання нейронної мережі	155
13.3. Приклад застосування нейронної мережі Кохонена . .	160
13.3.1. Фіксована кількість нейронів вихідного шару .	160
13.3.2. Нейронна мережа з невизначеною структурою	169
13.4. Карти Кохонена, що самоорганізуються	176
13.4.1. Структура карти Кохонена	177
13.4.2. Приклади використання карт Кохонена	183

Розділ 4. Глибоке навчання. Комп'ютерний зір.

Згорткові нейронні мережі.	191
Тема 14. Глибокі нейронні мережі та глибоке навчання . .	191
14.1. Сенс прихованих шарів	191
14.2. Застосування методів глибокого навчання	196
Тема 15. Згорткові нейронні мережі	201
15.1. Структура згорткової мережі та основні парадигми .	201
15.2. Побудова згорткової нейромережі	209
15.2.1. Налаштування параметрів згортковій мережі .	211
15.2.2. Прямий хід сигналу по нейронній мережі . . .	213
15.2.3. Зворотний хід: корегування значень вагових коєфіцієнтів	216
15.3. Лістинг коду для побудови мережі	221
15.4. Застосування побудованої мережі для розпізнавання зображень	228

Розділ 5. Рекурентні нейронні мережі: робота з текстом	233
Тема 16. Рекурентні нейронні мережі в задачах оброблення текстів	233
16.1. Архітектура RNN та особливості їх використання . . .	234
16.2. Побудова рекурентної нейронної мережі	236
16.3. Застосування рекурентних нейронних мереж для генерації тексту	237
16.4. Архітектури рекурентних нейронних мереж для генерації тексту	239
16.5. Проблеми та виклики	240
16.6. Приклади застосування генерації тексту	241
Тема 17. Аналіз емоційного забарвлення тексту	243
17.1. Побудова нейронної мережі	243
17.2. Попереднє оброблення даних	245
17.3. Фаза прямого поширення	247
17.4. Фаза зворотного поширення	249
17.4.1. Крос-ентропія	250
17.4.2. Параметри нейронної мережі	251
17.5. Тестування нейронної мережі	257
 Список літератури	 261

ПЕРЕДМОВА

Цей навчальний посібник містить основні теоретичні та математичні концепції побудови штучних нейронних мереж різної архітектури, приклади їх використання для завдань класифікації, кластеризації та розпізнавання образів. Розглянуто типові приклади одношарових та багатошарових нейронних мереж, методи їх навчання, а також приклади задач для кожного типу нейронної мережі з їх розв'язуванням. Для багатошарових мереж наведено приклад програмної реалізації з використанням мови програмування Python.

Для полегшення сприйняття матеріалу, викладеного в цьому навчальному посібнику вважається, що читач ознайомлений з основами лінійної алгебри, зокрема, з аспектами роботи з векторами та матрицями; основами математичного аналізу, статистичного аналізу та методами оптимізації.

Посібник складається з п'яти основних розділів. У першому розділі викладені основні концепції щодо будови біологічного нейрона та його математичної моделі, розглянуто модель штучного нейрона та типи активаційних функцій, архітектуру структуру та класифікацію штучних нейронних мереж, одношарові нейронні мережі та методи їх навчання. Другий розділ присвячено задачам навчання багатошарових нейронних мереж. Детально розглянуто метод градієнтного спуску, метод зворотного поширення помилки та його модифікації. Наприкінці розділу наведено приклади завдань з використанням багатошарових нейронних мереж прямого поширення. У третьому розділі розглянуто основні типи нейронних мереж, які використовуються у завданнях відновлення інформації, класифікації без учителя та кластеризації з прикладами застосування. У четвертому розділі приділено увагу завданням глибокого навчання в задачах комп'ютерного зору. На прикладі задачі розпізнавання надані основні моменти щодо побудови згорткової нейронної мережі та результати її навчання. Останній розділ присвячено рекурентним нейронним мережам в завданнях роботи з текстом.

Автор висловлює вдячність Олександрю Алфімову – директору ІТ компанії «СРCS» та backend-розробнику компанії «СРCS» Ігорю Князю за консультації та допомогу в структуруванні цього навчального посібника.

Розділ 1. Основні поняття теорії нейронних мереж. Нейромережеві моделі

Тема 1. Біологічний нейрон та його математична модель.

Штучний нейрон. Активаційна функція

1.1. Біологічний нейрон

Нейрон – це нервова клітина, яка є основним будівельним блоком для нервової системи. Нейрони багато в чому схожі з іншими клітинами, але існує одна важлива відмінність нейрона від інших клітин: від нейронів залежить передавання інформації по всьому тілу. Ці вузькоспеціалізовані клітини здатні на передавання інформації як хімічним, так і електричним шляхом. Налічують багато різних видів нейронів, що виконують різні функції в людському тілі. Нейрон (нервова клітка) складається з тіла клітини діаметром від 3 до 120 мкм (у середньому 10–30 мкм) – соми (soma), і двох типів зовнішніх відгалужень: аксона (аксон, грец. Вісь) і дендритів (dendrites, грец. Дерево). У тілі клітини розташовано ядро (nucleus), що містить інформацію про властивості нейрона. Ядро оточене плазмою, що продукує необхідні для нейрона матеріали. Тіло нервової клітини зовні обмежено мембраною з подвійного шару ліпідів (жирові сполуки). Нейрон одержує сигнали (імпульси) від інших нейронів через дендрити (приймачі) та передає сигнали, згенеровані тілом клітки, уздовж аксона (передавач), який наприкінці розгалужується на волокна (strands). На закінченнях волокон є невеликі потовщення, що називають синаптичними закінченнями

(synapses). Структуру біологічного нейрона подано на рисунку 1.1. Аксони та дендрити служать для зв'язку між різними нейронами. Нейрон може мати кілька дендритів і зазвичай лише один аксон. Один нейрон може мати зв'язки з багатьма (до 20 тисяч) іншими нейронами. Різні нейрони мають різне співвідношення довжини аксона та дендритів.

Місцем генерації збудження в більшості нейронів є аксонний горбок – утворення в місці відходження аксона від тіла. У всіх нейронів цю зону називають критичною. Аксон може поширюватися дуже далеко від тіла клітини. Наприклад, аксон нейрона спинного мозку може досягати 1,2 м в довжину і, розпочавшись від кінця хребта, йде до самих м'язів великого пальця ступні.

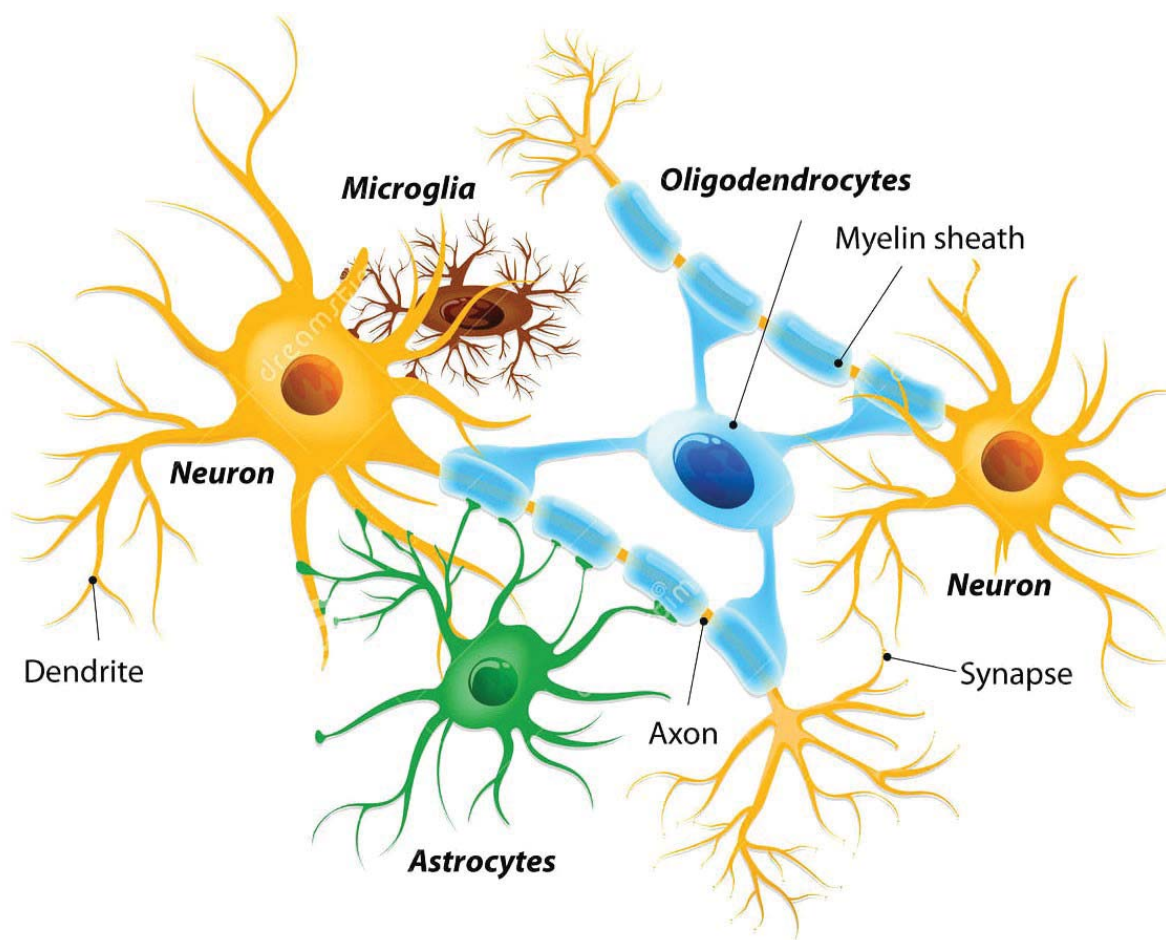


Рисунок 1.1 – Структура біологічного нейрона

На відміну від аксонів, дендритів зазвичай багато, вони тонкі й постійно розгалужуються прямо біля тіла клітини. Зазвичай у нейроні задіяно кілька сотень дендритів (дендритне дерево). Вони

формують зв'язки, що називаються «синапси», з іншими нейронами і є системою «проводів» мозку. Вони формуються розумовими процесами, впливом навколишнього середовища, навчанням і життєвим досвідом. Установлено, що в дорослої людини розвивається приблизно 1 трильйон дендритів у мозку, що при фізичному вимірі становило би більше як 160 тис. км. Нервові стовбури, або нерви, складаються з великої кількості аксонів і дендритів, об'єднаних у загальне цільною оболонкою. Самі тіла нейронів не розкидані хаотично, а утворюють скупчення, які називають:

- нервовими центрами, якщо тіла нейронів розміщені в головному або спинному мозку;
- ганглії, якщо тіла нейронів розміщені поза головним і спинним мозком.

1.1.1. Структурна класифікація нейронів

Залежно від кількості й розташування дендритів та аксона нейрони поділяють на види, зазначені нижче.

1. **Безаксонні нейрони** – невеликі клітини, що згруповані поблизу спинного мозку в міжхребцевих гангліях, які не мають анатомічних ознак поділу відростків на дендрити й аксони. Усі відростки в клітині дуже схожі. Функціональне призначення безаксонних нейронів слабо вивчене.

2. **Уніполярні нейрони** – нейрони з одним відростком, що містяться, наприклад у сенсорному ядрі трійчастого нерва в середньому мозку.

3. **Біполярні нейрони** – нейрони, що мають один аксон та один дендрит, розміщені в спеціалізованих сенсорних органах – сітківці ока, нюховому епітелії та цибулині, слуховому й вестибулярному гангліях.

4. **Псевдо-уніполярні нейрони** – є унікальними у своєму роді. Від тіла відходить один відросток, який відразу Т-подібно ділиться. Цей єдиний тракт структурно являє собою аксон, хоча по одній із гілок збудження йде не від, а до тіла нейрона. Структурно дендритами є розгалуження на кінці цього (периферичного) відростка. Критичною зоною є початок цього розгалуження (знаходиться поза тілом клітини).

5. **Мультиполярні нейрони** – нейрони з одним аксоном і кількома дендритами. Даний вид нервових клітин переважає в центральній нервовій системі

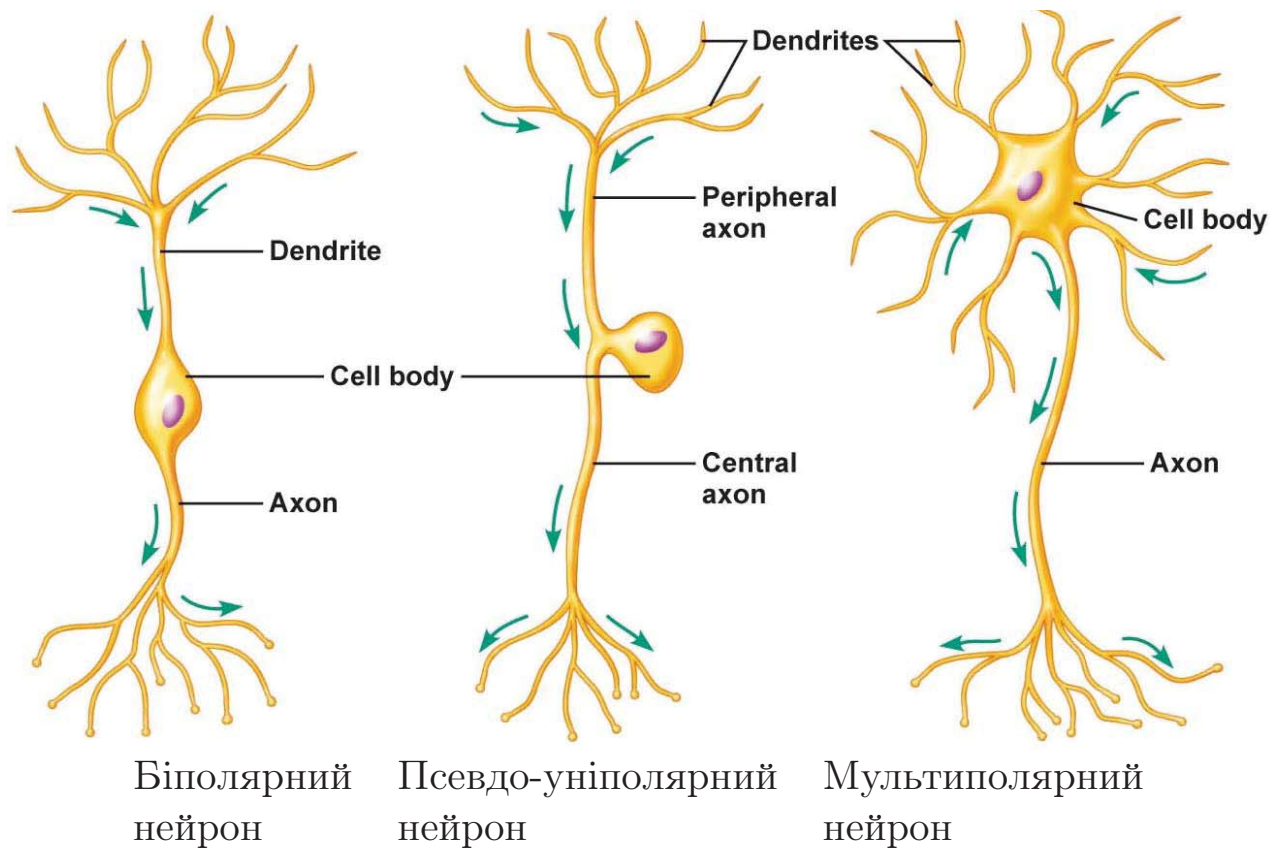


Рисунок 1.2 – Види нейронів

1.1.2. Типи нейронів

Залежно від покладених на нейрони завдань та обов'язків їх поділяють на три категорії.

1. **Аферентні нейрони** (чутливі, сенсорні або рецепторні). Ці нейрони приймають і передають імпульси від рецепторів «до центру», тобто центральної нервової системи. Рецептори – це спеціально навчені клітини органів почуттів, м'язів, шкіри й суглобів, які вміють виявляти фізичні або хімічні зміни всередині та зовні організму, перетворювати їх на імпульси, і передавати до сенсорних нейронів. Отже, сигнали йдуть від периферії до центру.

2. **Еферентні нейрони** (ефекторні, рухові або моторні). Ці нейрони несуть сигнали, що виходять із головного або спинного

мозку, до органів, що виконують роботу, зокрема м'язів, залоз, тощо. Сигнали йдуть від центру до периферії.

3. Асоціативні нейрони (вставні або інтернейрони). Ці нейрони одержують сигнали від сенсорних нейронів і посилають ці імпульси далі до інших проміжних нейронів, або відразу до моторних нейронів.

У сенсорних нейронах дендрити з'єднані з рецепторами, а аксони – з іншими нейронами (вставними). У рухових нейронів навпаки, дендрити з'єднані з іншими нейронами (вставними), а аксони – з якимось ефектором, тобто стимулятором скорочення певного м'язу або секреції залози. Відповідно, у проміжних нейронів і дендрити, і аксони з'єднуються з іншими нейронами.

Найпростіший шлях, за яким може йти нервовий імпульс, буде складатися з трьох нейронів: одного сенсорного, одного вставного й одного моторного. Наприклад, на прийомі в невропатолога він стукає молоточком по коліну. Це є найпростіший рефлекс: коли він б'є по колінному сухожиллю, прикріплений до нього м'яз розтягується й сигнал від чутливих клітин (рецепторів), що знаходяться в ньому передається по сенсорних нейронах до спинного мозку. У спинному мозку сенсорні нейрони контактують або через проміжні, або безпосередньо з моторними нейронами, які у відповідь посилають імпульси назад до м'яза, змушуючи його скорочуватися, а ногу – розправлятися.

1.1.3. Передавання нервового імпульсу

Передача нервового імпульсу від нейрона до нейрона досить незвичайна й цікава. Нервовий імпульс здебільшого поширюється лише в одному напрямку – із кількох дендритів до тіла клітини, і від тіла, по єдиному аксону до м'яза або якогось органу або дендритів наступного нейрона.

Синаптичне закінчення аксона не торкається іншого нейрона, до якого надходить збудження. Між синаптичним закінченням і дендритом сприймаючої клітини існує невеликий проміжок, що називають синаптичною щілиною, а саме – з'єднання синапсом. У ході синаптичної передачі амплітуда й частота сигналу можуть регулюватися. Одні синапси викликають деполяризацію нейрона, інші –



Рисунок 1.3 – Схема передавання нервового імпульсу

гіперполяризацію; перші є збуджувальними, інші – гальмівними. Зазвичай для збудження нейрона необхідно подразнення від декількох збуджувальних синапсів. Коли нервовий імпульс, проходячи по аксону, досягає синаптичного закінчення, він запускає виділення особливої хімічної речовини, що називають нейромедіатором (медіатором). Медіатор проникає через синаптичну щілину й стимулює наступний нейрон, передаючи сигнал від одного нейрона до іншого.

Нейромедіатор (або нейротрансмітер) – «посередник» хімічного походження, який бере участь у передаванні, посиленні й модуляції сигналів між нейронами та іншими клітинами (наприклад, м'язової тканини) в організмі. У більшості випадків нейромедіатор вивільняється з термінальних гілок аксонів після того, як потенціал дії досягає синапсу. Далі нейромедіатор перетинає синаптичну щілину й досягає рецептора інших клітин або нейронів. А потім у процесі, який називають зворотним захопленням, він зв'язується з рецептором і поглинається нейроном.

Нейромедіатори відіграють важливу роль у житті людини. Учені поки не змогли дізнатися точну кількість нейромедіаторів, але їм вдалося ідентифікувати понад 100 хімічних речовин. Вплив хвороби або, наприклад, наркотиків на нейротрансмітери призводить до різних несприятливих наслідків для організму.

1.1.4. Класифікація нейромедіаторів

Залежно від їх функції нейромедіатори можна розділити на два типи.

1. **Збуджувальні.** Цей тип нейромедіаторів чинить збудливу дію на нейрон. Вони збільшують імовірність того, що нейрон буде генерувати потенціал дії. До основних збудливих нейротрансмітерів відносять адреналін і норадреналін.

2. **Пригнічувальні.** Ці нейротрансмітери чинять гальмівну дію на нейрон; вони зменшують імовірність того, що буде вироблено потенціал дії. Основними нейромедіаторами інгібуючого типу вважають серотонін і гамма-аміномасляну кислоту (або ГАМК). Деякі нейротрансмітери можуть надавати збудливого й гальмівного ефекту залежно від типу рецепторів, якими володіє постсинаптичний нейрон.

Коли нейромедіатор, проникає через синапс, він вступає в контакт із рецепторними нервовими закінченнями іншого нейрона. Цей нейрон називають постсинаптичним, із рецепторними закінченнями, що знаходяться, здебільшого, на його дендритах і тілі. Речовина-передавач (медіатор) викликає або деполяризацію (збудження) постсинаптичного нейрона, або гіперполяризацію (гальмування).

Чи буде постсинаптичний нейрон генерувати потенціал дії (передавати повідомлення далі), залежить від сумарного впливу на нього з боку пресинаптичних нейронів. Кожен нейрон може мати сотні синапсів, і через одні до нього можуть приходити збуджувальні сигнали, а через інші – гальмівні. Нейрон підсумовує всі ці дії і залежно від результату або генерує потенціал дії, або ні.

Головний мозок використовує багато нейромедіаторів. Упродовж багатьох років вважали, що кожен нейрон вивільняє з усіх своїх кінцевих бляшок лише один певний медіатор. Зараз відомо, що більшість нейронів можуть вивільняти 2–3 медіатори, а деякі – навіть 5–6.

1.2. Штучний нейрон

Історично першою роботою, яка заклала теоретичний фундамент для створення штучних моделей нейронів і нейронних мереж, прийнято вважати опубліковану в 1943 р. статтю Уоррена Мак-Каллока та Уолтера Піттса «Логічне числення ідей, що належить до нервової активності». Основним принципом теорії Мак-Каллока і Піттса є те, що довільні явища, які стосуються вищої нервової діяльності, можуть бути проаналізовані та зрозумілі як деяка активність у мережі, що складається з логічних елементів, які приймають тільки два стани – «все або нічого». Водночас для будь-якого логічного виразу, за наведених авторами умов, може бути знайдена мережа логічних елементів, що має поведінку, описану цим виразом.

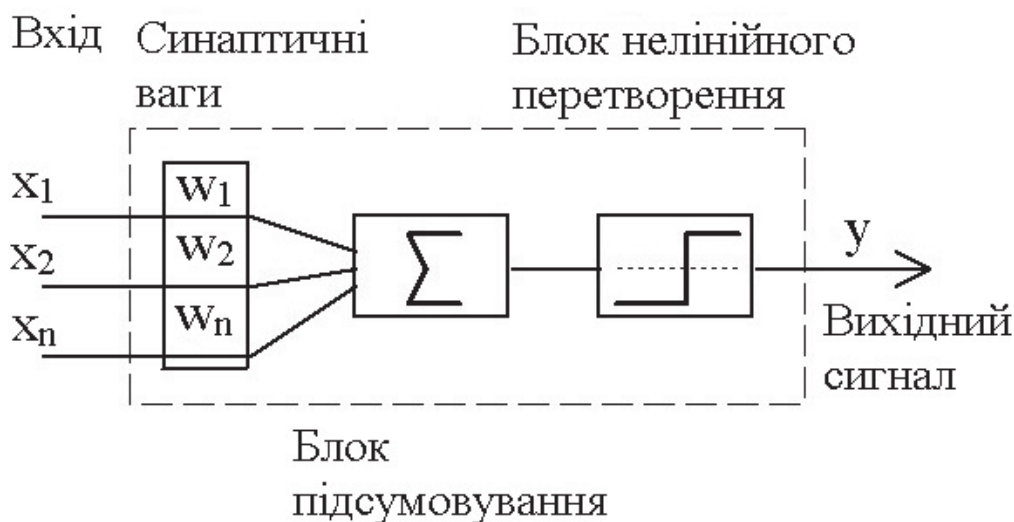


Рисунок 1.4 – Функціональна схема формального нейрона Мак-Каллока і Піттса

За модель такого логічного елемента, який у подальшому назвали «формальний нейрон», було запропоновано схему, наведену на рисунку 1.4. Формальний нейрон представляє математичну модель простого процесора, що має кілька входів і один вихід. Вектор вхідних сигналів (що надходять через «дендрити») перетворюється нейроном на вихідний сигнал (що поширюється по «аксону») з використанням трьох функціональних блоків: локальної пам'яті, блоку підсумовування та блоку нелінійного перетворення.

Вектор локальної пам'яті містить інформацію про вагові коефіцієнти із якими вхідні сигнали будуть інтерпретовані нейроном. Ці змінні ваги є аналогом чутливості пластичних синаптичних контактів. Вибором ваг досягається та чи інша інтегральна функція нейрона.

У **блоці підсумовування** відбувається накопичення загального вхідного сигналу (зазвичай позначають символом net), що дорівнює зваженій сумі входів

$$net = \sum_{i=1}^n w_i x_i.$$

У моделі Мак-Каллок і Піттса відсутні тимчасові затримки вхідних сигналів, тому значення net визначає повне зовнішнє збудження, яке сприймається нейроном. Відгук нейрона далі описаний за принципом «все або нічого», тобто змінна піддається нелінійному пороговому перетворенню, за якого вихід (стан активації нейрона) y встановлюють рівним одиниці, якщо $net > \Theta$, і $y = 0$ в іншому випадку. Значення порога Θ (зазвичай дорівнює нулю) також зберігається в локальній пам'яті.

Формально нейрони можуть бути об'єднані в мережі методом замикання виходів одних нейронів на входи інших, і на думку авторів моделі, така кібернетична система з належно обраними вагами може бути довільною логічною функцією.

Більшість моделей нейронних мереж ґрунтуються на різних модифікаціях формального нейрона.

1.2.1. Розширена модель штучного нейрона

У наявних на цей час пакетах програм штучні нейрони називають «процесорами» або «елементами оброблення» й мають значно більше можливостей, ніж формальний нейрон, описаний вище. На рисунку 1.5 зображено детальну схему розширеної моделі штучного нейрона.

Модифіковані входи передаються в блок підсумовування де, можна обрати багато різних операцій, такі як сума, середнє, найбільше, найменше, OR, AND, тощо, які могли б виробляти деяку

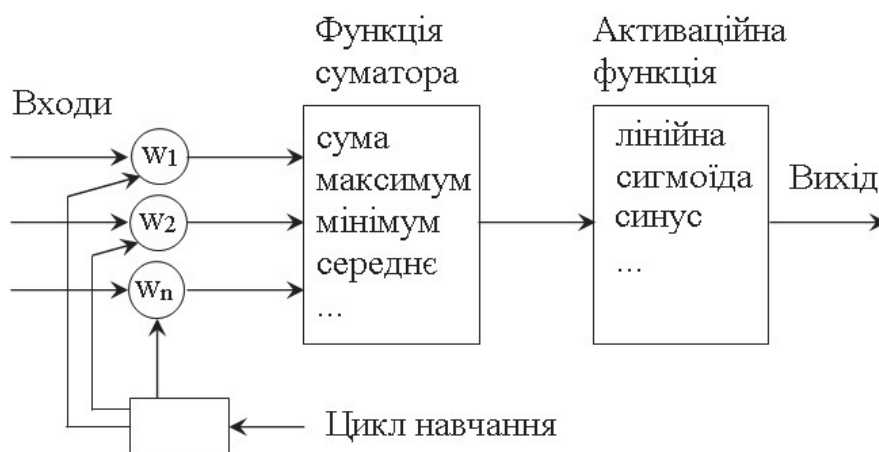


Рисунок 1.5 – Схема розширеної моделі штучного нейрона

кількість різних значень. Крім того, більшість комерційних програм дозволяють інженерам-програмістам створювати власні функції сумування за допомогою підпрограм, закодованих мовою високого рівня.

Вихід суматора надсилається до активаційної функції та скеровує весь ряд на дійсний вихід (0 або 1, -1 або 1 , або якесь інше число) за допомогою певного алгоритму. В існуючих нейромережах за активаційні функції можуть використовувати сигмоїд, гіперболічний тангенс та ін. Приклад того, як працює активаційна функція показано на рисунку 1.6. Після оброблення сигналу, нейрон на виході має результат активаційної функції, що надходить на входи інших нейронів або до зовнішнього з'єднання, як це передбачено структурою нейромережі. Усі штучні нейромережі конструюють із базового формувального блоку – штучного нейрона. Наявні різноманітності й фундаментальні відмінності є підставою мистецтва талановитих розробників для реалізації ефективних нейромереж.

Підставою для деталізації моделі нейронного елемента можна вважати встановлення нових фактів в області нейрофізіології, зокрема:

- наявність декількох місць синаптичного контакту;
- дихотомічне розгалуження дендритів різних порядків, що відповідає логічним операціям «І», «АБО», «Виключаюче АБО», виділення максимального або мінімального сигналу в технічних аналогах;

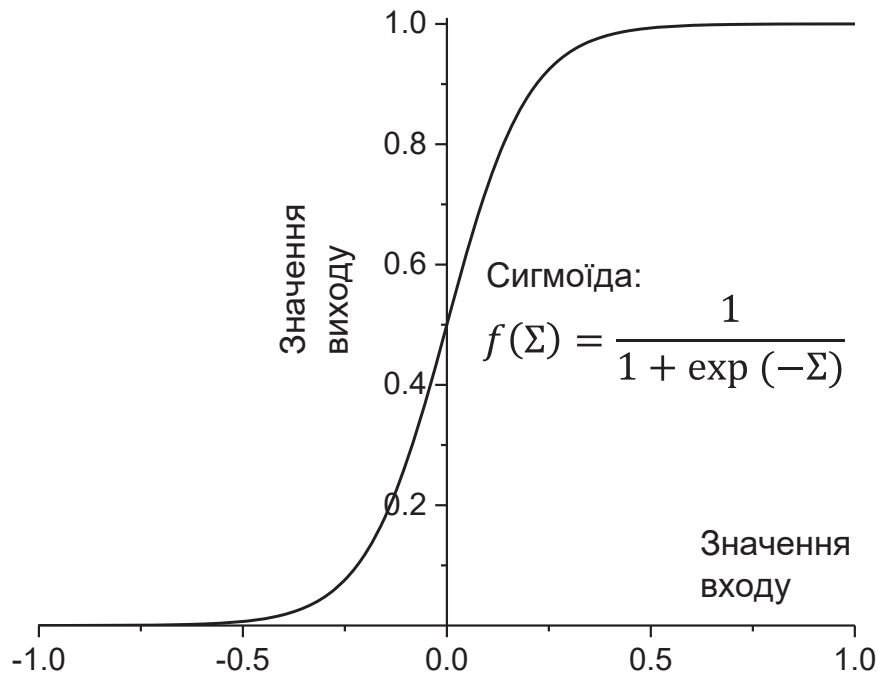


Рисунок 1.6 – Сигмоїдна функція активації

- різні діаметри стовбурових дендритів, гілок, що безпосередньо прилягають до тіла нейрона, причому величина діаметра визначає ступінь важливості інформації, що проходить через дендрит;
- наявність «доріжок» на поверхні соми, які проходять від основних стовбурних дендритів до аксона, що обумовлює наявність паралельних шляхів оброблення інформації й надає можливість застосування логічних операцій над сигналами, що надходять від різних стовбурових дендритів;
- особливості функціонування аксонного горбка; власне аксонний горбок установлює активаційну функцію нейрона, яка має набагато складнішу форму від прийнятих у нейромережових технологіях сигмоїдальних або лінійних активаційних функцій;
- наявність дихотомічного галуження аксона; у вузлах галуження відбувається керування проходженням сигналу, що залежить від співвідношення діаметрів різних гілок аксона (при математичному моделюванні ці особливості можна реалізувати за допомогою логічних операцій);
- наявність зворотного аксосоматичного зв'язку, що вже знайшов свою реалізацію під час побудови рекурентних нейромереж.

Поглиблені знання відносно будови біологічного нейрона як ефективного перетворювального інструмента можна розглядати як джерело базових ідей та концепцій зі створення нових парадигм нейромереж не лише на сьогодні, але й на віддалену перспективу.

1.3. Компоненти штучного нейрона

Незалежно від розташування та функціонального призначення всі штучні нейронні елементи мають спільні компоненти. Розглянемо сім основних компонент штучного нейрона.

Компонента 1. Вагові коефіцієнти

Під час функціонування нейрон одночасно одержує багато вхідних сигналів. Кожен вхід має свою власну синаптичну вагу, що надає входу впливу, необхідному для функції суматора елемента оброблення. Ваги є мірою сили вхідних зв'язків і моделюють різноманітні синаптичні сили біологічних нейронів. Ваги суттєвого входу підсилюються й навпаки вага несуттєвого входу примусово зменшується, що визначає інтенсивність вхідного сигналу. Ваги можуть змінюватися відповідно до навчальних прикладів, топології мережі та навчальних правил.

Компонента 2. Функція суматора

Першим кроком дії нейрону є обчислення зваженої суми всіх входів. Математично, вхідні сигнали та відповідні їм ваги репрезентовані векторами $(x_{10}, x_{20}, \dots, x_{n0})$ та $(w_{10}, w_{20}, \dots, w_{n0})$. Добуток цих векторів є загальним вхідним сигналом. Спрощеною функцією суматора є множення кожної компоненти вектора x на відповідну компоненту вектора w : $\text{вхід}_1 = x_{10} \cdot w_{10}$, $\text{вхід}_2 = x_{20} \cdot w_{20}$, і знаходження суми всіх добутків: $\text{вхід}_1 + \text{вхід}_2 + \dots + \text{вхід}_n$. Результатом є єдине число, а не багатоелементний вектор. Функція суматора може бути складнішою, наприклад, вибір мінімуму, максимуму, середнього арифметичного, добутку або виконувати інший нормалізуючий алгоритм. Вхідні сигнали та вагові коефіцієнти можуть комбінуватися багатьма способами перед надходженням до активаційної функції. Особливі алгоритми для комбінування входів нейронів визначаються обраними архітектурою та парадигмою мережі.

Компонента 3. Активаційна функція

Результат функції суматора є зваженою сумою вхідних сигналів, що перетворюється на вихідний сигнал через алгоритмічний процес відомий як активаційна функція. В активаційній функції для визначення виходу нейрона загальну суму порівнюють з деяким порогом. Якщо сума є більшою за значення порогу, елемент оброблення генерує сигнал, в протилежному разі сигнал не генерується або генерується гальмівний сигнал. Переважно застосовують нелінійну активаційну функцію, оскільки лінійні (прямолінійні) функції обмежені й вихід є просто пропорційним входу. Застосування лінійних активаційних функцій було проблемою в ранніх моделях мереж. На рисунку нижче зображено типові активаційні функції. Типові активаційні функції наведено в таблиці 1.1.

Для простої активаційної функції нейромережа може видавати 0 та 1, 1 та -1 або інші числові комбінації. Активаційна функція в таких випадках є «жорстким обмежувачем» або пороговою функцією.

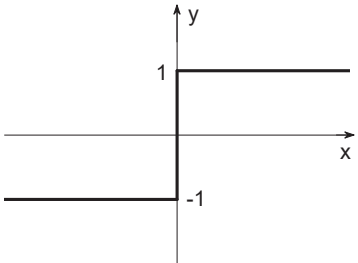
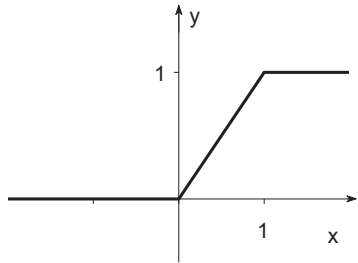
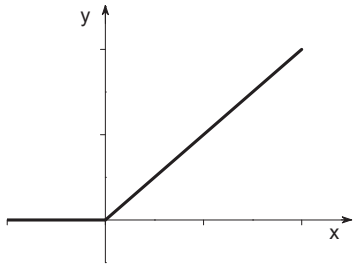
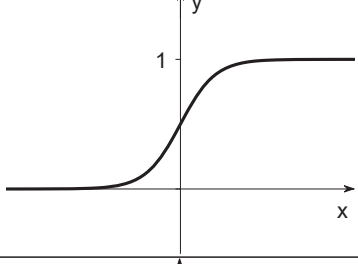
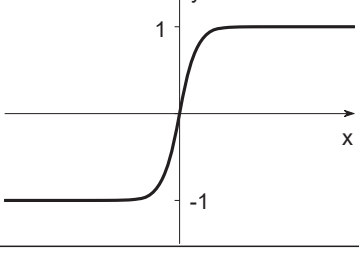
Інший тип активаційної функції – лінійна з насиченням віддзеркалює вхід усередині заданого діапазону й діє як жорсткий обмежувач за його межами. Це лінійна функція, що відсікається до мінімальних та максимальних значень, роблячи її нелінійною.

Активаційна функція *ReLU* (Rectified Linear Unit) – заміняє усі негативні значення на нуль і передає позитивні значення без змін. Ця функція придатна для будь-яких завдань, не вимагаючи великих обчислювальних потужностей.

Серед нелінійних активаційних функцій виділяють сигмоїдну функцію (або логістичну функцію), що наближує мінімальне та максимальне значення в асимптотах $[0, 1]$. Вона застосовується у задачах бінарної класифікації, де потрібно передбачити можливість приналежності до одного з двох класів та конвертує вхідні значення у ймовірності, які можуть бути інтерпретовані як ймовірність приналежності до позитивного класу. Гіперболічний тангенс схожий до сигмоїдної функції, проте обмежений в діапазоні $[-1, 1]$.

Зрештою, для різних нейромереж можуть вибиратися інші активаційні функції. Перед надходженням до активаційної функції до вхідного сигналу деколи додають однорідно розподілений випад-

ковий шум, джерело та кількість якого визначається режимом навчання. У літературі цей шум, згадано як «температура» штучних нейронів, що надає математичній моделі елемент реальності.

Назва	Графік	Рівняння	Область значень
Жорстка порогова функція		$y = \begin{cases} -1, & x < 0 \\ 1, & x \geq 0 \end{cases}$	$\{-1, 1\}$
Лінійна функція з насиченням		$y = \begin{cases} 0, & x < 0 \\ x, & 0 < x < 1 \\ 1, & x \geq 1 \end{cases}$	$[0, 1]$
ReLU (Rectified Linear Unit)		$y = \begin{cases} 0, & x \leq 0 \\ x, & x > 0 \end{cases}$	$[0, x]$
Сигмоїдна функція		$y = \frac{1}{1+e^{-x}}$	$[0, 1]$
Тангенс гіперболічний		$y = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$[-1, 1]$

Таблиця 1.1 – Типові активаційні функції

Компонента 4. Масштабування

Після активаційної функції вихідний сигнал проходить додаткове оброблення: результат активаційної функції множать на масштабувальний коефіцієнт і додають зміщення.

Компонента 5. Вихідна функція

За аналогією з біологічним нейроном кожний штучний нейрон має один вихідний сигнал, який передається до сотень інших нейронів. Переважно вихід прямо пропорційний результату активаційної функції. У деяких мережевих топологіях результати активаційної функції змінюються для створення змагання між сусідніми нейронами. Нейронам дозволено змагатися між собою, блокуючи дії нейронів, що мають слабкий сигнал. Змагання (конкуренція) може відбуватися між нейронами, що знаходяться на одному або різних прошарках. По-перше, конкуренція визначає, який штучний нейрон буде активним і забезпечить вихідний сигнал. По-друге, конкуруючі виходи допомагають визначити, який нейрон буде брати участь у процесі навчання.

Компонента 6. Функція похибки

У більшості мереж, що застосовують контрольоване навчання, обчислюють різницю між одержаним та бажаним виходом. Похибка відхилення (біжуча похибка) перетворюється функцією похибки відповідно до заданої мережної архітектури. У базових архітектурах похибку відхилення використовують безпосередньо, у деяких парадигмах використовують квадрат або куб похибки зі збереженням знака. Після проходження всіх шарів похибка поширюється назад до попереднього шару й може бути безпосередньо похибкою або похибкою, масштабованою певним чином залежно від типу мережі (наприклад похідною від активаційної функції). Це поширюване назад значення враховується у наступному циклі навчання.

Компонента 7. Функція навчання

Метою функції навчання є налаштування змінних ваг з'єднань на входах кожного елемента оброблення відповідно до певного алгоритму навчання для досягнення бажаного результату.

Тема 2. Архітектура, структура та класифікація штучних нейронних мереж

Згідно з класичним визначенням нейронною мережею називають деяку послідовність нейронів, об'єднаних між собою синапсами. Якщо програма має структуру нейронної мережі, з'являється можливість на машинному рівні проаналізувати вхідні дані із запам'ятовуванням результату.

Нейромережа – це зв'язка нейронів. Кожен із цих нейронів одержує дані, обробляє їх, а потім передає іншому нейрону. І кожен нейрон обробляє сигнали однаково. Але як же ми одержуємо різний результат? За це відповідають синапси, які з'єднують нейрони один з одним. Кожен нейрон здатний мати багато синапсів, які послаблюють або посилюють сигнал. Нейрони здатні змінювати свої характеристики впродовж певного часу. Одержання правильних результатів перетворення вхідної інформації на виході нейронної мережі досягають правильним вибором параметрів синапсів.

2.1. Архітектура з'єднань штучних нейронів

Об'єднуючись у мережі, нейрони утворюють системи оброблення інформації, які забезпечують ефективну адаптацію моделі до постійних змін із боку зовнішнього середовища. У процесі функціонування мережі відбувається перетворення вхідного вектора сигналів у вихідний. Конкретний вид перетворення визначається як архітектурою нейромережі, так і характеристиками нейронних елементів, засобами керування та синхронізації інформаційних потоків між нейронами. Важливим фактором ефективності мережі є встановлення оптимальної кількості нейронів, типів зв'язків між ними та відповідних правил передавання інформації. Під час опису нейромереж використовують кілька усталених термінів, які в різних джерелах можуть мати різне трактування, зокрема:

- структура нейромережі – спосіб зв'язків нейронів у нейромережі;
- архітектура нейромережі – структура нейромережі та типи нейронів;
- парадигма нейромережі – спосіб навчання та використання; іноді також уміщує поняття архітектури.

На основі однієї архітектури можуть бути реалізовані різні парадигми нейромережі й навпаки. Серед відомих архітектурних рішень виділяють групу слабозв'язаних нейронних мереж у разі, коли кожний нейрон мережі зв'язаний лише із сусідніми. Якщо входи кожного нейрона зв'язані з виходами усіх решти нейронів, тоді мова йде про повнозв'язані нейромережі.

2.2. Структура нейронної мережі

Кожна нейронна мережа містить перший шар нейронів, який називають вхідним. Цей шар не виконує будь-яких перетворень та обчислень, його завдання полягає в тому, щоб приймати та розподіляти вхідні сигнали по решті нейронів. І цей шар єдиний, що є загальним для всіх типів нейромереж, а критерієм для поділу є подальша структура. Зовнішній вхідний вектор подається на вхідний шар нейронної мережі (рецептори). Виходами нейронної мережі є вихідні сигнали останнього шару (ефектори). Крім вхідного та вихідного шарів, нейромережа має один або декілька прихованих шарів нейронів, які не мають контактів із зовнішнім середовищем. Виділяють два типи структур штучних нейронних мереж.

1. Одношарова структура нейронної мережі є структурою взаємодії нейронів, у якій сигнали з вхідного шару відразу спрямовані на вихідний шар, який, власне кажучи, не лише перетворює сигнал, але й відразу ж видає відповідь. Як було зазначено, 1-й вхідний шар лише приймає та розподіляє сигнали, а необхідні обчислення відбуваються вже в другому шарі. Вхідні нейрони є об'єднаними з основним шаром за допомогою синапсів із різними вагами, що забезпечують якість зв'язків. Найпростіша одношарова мережа складається з групи нейронів, що утворюють шар, як показано на рисунку 2.1.

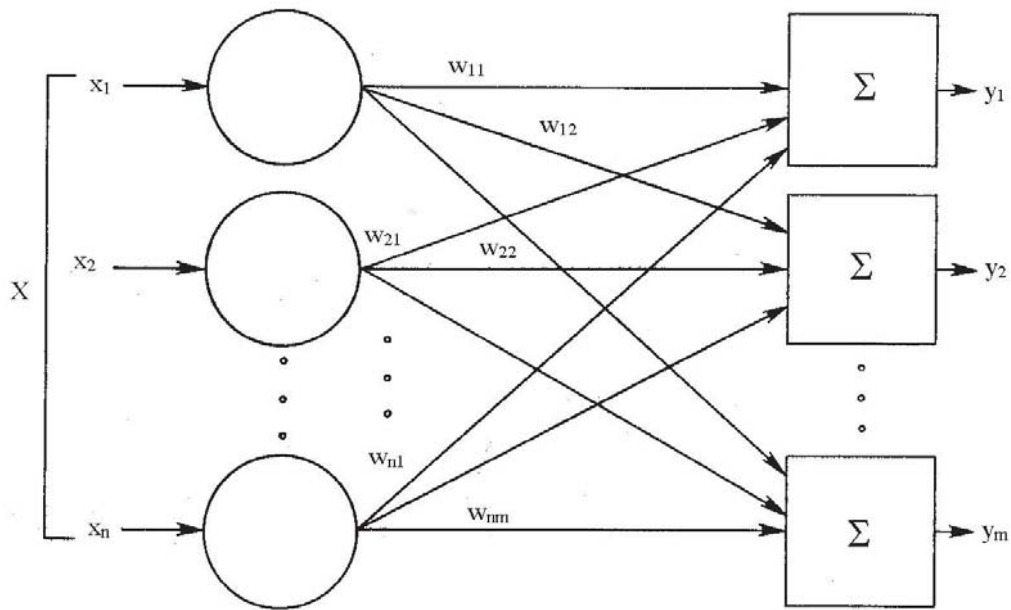


Рисунок 2.1 – Одношарова нейронна мережа

Вершини – кола виконують розподілення вхідних сигналів. Вони не виконують жодних обчислень і тому не вважаються шаром. Кожен елемент із множини входів X окремим ваговим коефіцієнтом з'єднаний із кожним штучним нейроном. А кожен нейрон видає зважену суму входів у мережу. У штучних і біологічних мережах багато зв'язків можуть бути відсутніми, всі зв'язки показані з метою узагальненості. Можуть бути актуальними також з'єднання між виходами й входами елементів у шарі.

Зручно вважати ваги w_{ij} елементами матриці $W(n \times m)$. Отже, обчислення вихідного вектора net зводиться до матричного множення $net = X \times W$, де net та X – вектори-рядки.

2. Багатошарова нейронна мережа – мережа, в якій крім вихідного та вхідного шарів, є ще кілька прихованих проміжних шарів. Кількість цих шарів залежить від ступеня складності нейронної мережі. Вона здебільшого мірою нагадує структуру біологічної нейронної мережі. Такі види були розроблені зовсім недавно, раніше всі процеси були реалізовані за допомогою одношарових нейронних мереж. Відповідні рішення мають великі можливості, якщо порівнювати з одношаровими, адже в процесі оброблення даних кожен проміжний шар – це проміжний етап, на якому здійснюють оброблення та розподіл інформації.

Багатошарові мережі можуть утворюватися каскадами шарів. Вихід одного шару є входом наступного. Багатошарові мережі можуть привести до збільшення обчислювальної потужності лише в тому разі, якщо активаційна функція між шарами буде нелінійною. Обчислення виходу шару полягає в множенні вхідного вектора на першу вагову матрицю з подальшим множенням (якщо відсутня нелінійна активаційна функція) результуючого вектора на другу вагову матрицю. Отже, будь-яка багатошарова лінійна мережа може бути замінена еквівалентною одношаровою мережею, як показано на рисунку 2.2.

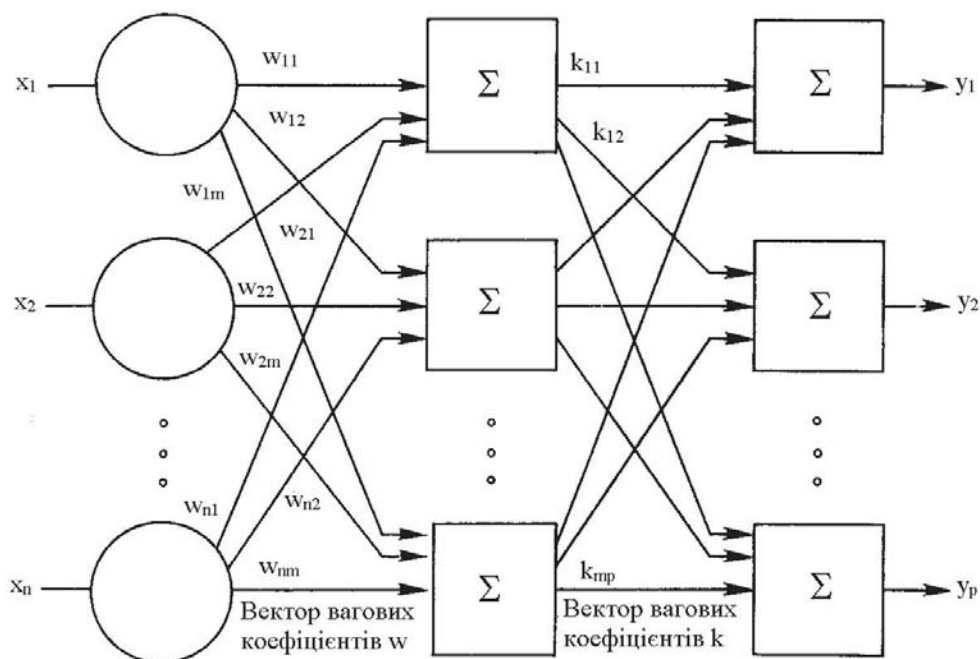


Рисунок 2.2 – Багатошарова нейронна мережа

Зв'язки між нейронами різних шарів називають проєктивними. Зв'язки скеровані від вхідних шарів до вихідних називають аферентними, в іншому випадку, при зворотному напрямку зв'язків – еферентними. Зв'язки між нейронами одного шару відносять до бічних (латеральних).

Спосіб, яким нейрони з'єднуються між собою має значний вплив на роботу мережі. Більшість програмних моделей дозволяють користувачу додавати, вилучати та керувати з'єднаннями як завгодно. Корегуючи параметри, зв'язки можна робити як збуджувальними, так і гальмівними.

2.3. Класифікація нейронних мереж

Крім кількості шарів, нейронні мережі можна класифікувати за напрямом розподілення інформації щодо синапсів між нейронами.

1. Нейронні мережі прямого поширення (односпрямовані). У цій структурі сигнал переміщається виключно в напрямку від вхідного шару до вихідного. Рух сигналу в протилежному напрямку не здійснюється і в принципі неможливий. Розроблення цього плану поширені й сьогодні успішно вирішують завдання розпізнавання образів, прогнозування та кластеризації.

2. Рекурентні нейронні мережі (зі зворотними зв'язками). Тут сигнал рухається як у прямому, так і зворотному напрямку. Водночас результат виходу здатний повертатися на вхід. Вихід нейрона визначається ваговими коефіцієнтами та вхідними сигналами, плюс доповнюється попередніми виходами, які знову повертаються на вхід. Цим нейромережам властива функція короткочасної пам'яті, з огляду на те, що сигнали відновлюються й доповнюються під час їх оброблення.

3. Мережа радіально базисних функцій у математичному моделюванні – це штучна нейронна мережа, яка використовує радіальні базисні функції як функції активації. Виходом мережі є лінійна комбінація радіальних базисних функцій входу та параметрів нейрона. Мережі радіальних базисних функцій мають багато застосувань, зокрема такі, як апроксимацію функції, прогнозування часових рядів, задачі класифікації та керування системою.

4. Карти, що самоорганізуються – це один із різновидів нейромережових алгоритмів. Основною відмінністю даної технології від нейромереж, які навчаються за алгоритмом зворотного поширення, є те, що під час навчання використовують метод навчання без учителя, тобто його результат залежить від структури вхідних даних. Нейронні мережі даного типу часто застосовують для вирішення найрізноманітніших завдань, від відновлення прогалин у даних до аналізу даних та пошуку закономірностей.

Додатково, нейромережі класифікують за критеріями, наведеними нижче.

1. Залежно від типів нейронів:
 - однорідні;
 - гібридні.
2. Залежно від методу навчання нейронних мереж:
 - навчання з учителем;
 - навчання без учителя;
 - навчання з підкріпленням.
3. За типом вхідної інформації:
 - аналогові;
 - двійкові;
 - образні.
4. За характером налаштування синапсів:
 - із фіксованими зв'язками;
 - із динамічними зв'язками.

Тема 3. Навчання штучної нейронної мережі

Після визначення архітектури нейронної мережі її потрібно «навчити», тобто підібрати такі значення її ваг, щоб вона працювала належним чином. Процес навчання нейронної мережі полягає в налаштуванні її внутрішніх параметрів під конкретну задачу. Навчити нейронну мережу – значить, повідомити їй, чого від неї вимагають. Алгоритм роботи нейронної мережі є ітеративним, його кроки називають **епохами** або циклами.

Виділяють чотири типи навчання нейронної мережі:

- навчання з учителем – нейронна мережа навчається на розміченому наборі даних і передбачає відповіді, що використовуються для оцінювання точності алгоритму на навчальних даних;
- навчання без учителя – модель використовує нерозмічені дані, зокрема алгоритм самостійно намагається витягти ознаки і залежності;
- навчання з частковим залученням учителя використовує невелику кількість розмічених даних та великий набір нерозмічених;
- навчання з підкріпленням – тренує алгоритм за допомогою системи заохочень; агент одержує зворотний зв'язок у вигляді винагород за правильні дії.

Які входні поля (ознаки) необхідно використовувати? Спочатку здійснюють евристичний вибір, далі кількість входів може бути змінено. Складність може викликати питання про кількість прикладів у наборі даних. Уся інформація, яку нейронна мережа має про завдання, міститься в наборі прикладів, тому якість навчання нейронної мережі безпосередньо залежить від кількості прикладів у навчальній вибірці, а також від того, наскільки повно ці приклади описують це завдання. Так, наприклад, безглуздо використовувати нейронну мережу для передбачення фінансової кризи, якщо в навчальній вибірці не репрезентовано жодної кризи. Для повноцінного навчання нейронної мережі потрібна репрезентативна й доволі велика вибірка із сотні (а краще тисячі) прикладів. Кількість

необхідних прикладів залежить від складності розв'язуваної задачі. У разі збільшення кількості ознак кількість прикладів зростає нелінійно, ця проблема має назву «прокляття розмірності». За недостатньої кількості даних рекомендовано використовувати лінійну модель.

Розробник повинен надати можливості вибору кількості шарів у мережі й кількості нейронів у кожному шарі. Далі необхідно призначити такі значення ваг і зсувів, які зможуть мінімізувати похибку рішення. Від якості навчання нейронної мережі залежить її здатність вирішувати поставлені перед нею завдання.

3.1. Навчання з учителем

Навчання з учителем (**supervised learning**) передбачає наявність повного набору розмічених даних для тренування моделі всіх етапах її побудови.

Процес навчання здійснюють на навчальній вибірці (множині). Навчальна вибірка містить набір даних про предметну область (об'єкт, явище, процес), поділену на вхідні значення й відповідні їм вихідні значення набору даних.

Наприклад, після пред'явлення зображення літери «А» на вхід нейронної мережі вона видає деяку відповідь, необов'язково правильну. У навчальній множині міститься вірна (бажана) відповідь, а сенсом навчання є те, щоб на виході нейронної мережі з міткою «А» рівень сигналу був максимальний. У ході навчання нейронна мережа знаходить певні залежності вихідних полів від вхідних. Зазвичай як бажаний вихід в задачі класифікації беруть набір $(1, 0, 0, \dots)$, де 1 стоїть на виході з міткою «А», а 0 – на всіх інших виходах. Обчислюючи різницю між бажаною та реальною відповідями мережі, утворюється функція похибки. Функція похибки це цільова функція, дозволяє оцінити якість роботи нейронної мережі й потребує мінімізації в процесі керованого навчання нейронної мережі. Алгоритмом навчання є набір формул, що дозволяє за функцією похибки обчислити необхідні поправки для вагових коефіцієнтів зв'язків нейронної мережі.

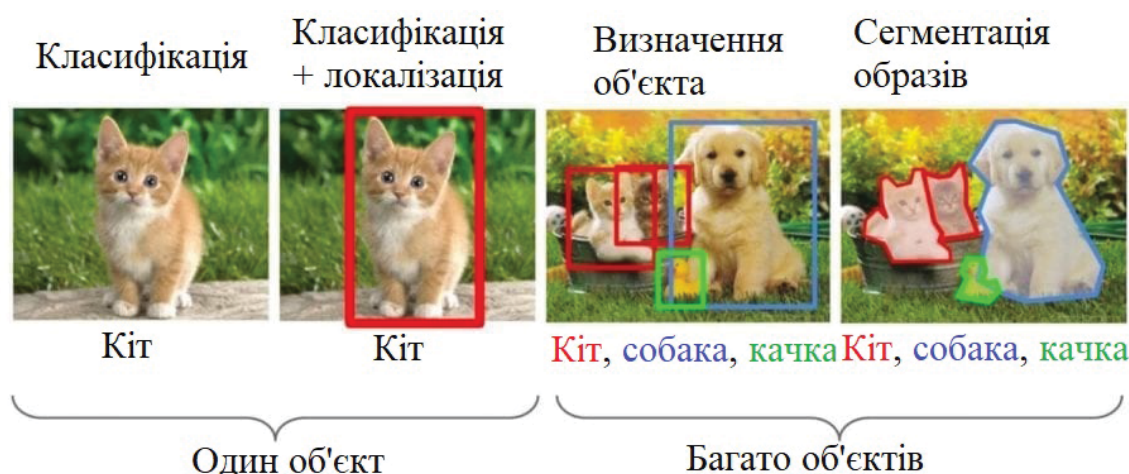


Рисунок 3.1 – Приклад навчання з учителем – класифікація (ліворуч), та подальше її використання для сегментації та розпізнавання об'єктів (праворуч)

Наявність повністю розміченого датасету означає, що кожним прикладам у навчальному наборі співвідносна відповідь, яку алгоритм повинен одержати. Розмічений датасет із фотографій кольорів навчить нейронну мережу, де зображені троянди, ромашки чи нарциси. Коли мережа одержує нове фото, вона порівняє його з прикладами навчального датасету, щоб передбачити відповідь.

Здебільшого навчання з учителем застосовують для вирішення двох типів завдань: класифікації та регресії.

У задачах класифікації алгоритм передбачає дискретні значення, що відповідають номерам класів, до яких належать об'єкти. У датасеті з фотографіями тварин кожне зображення матиме відповідну мітку — «кішка», «собака» або «качка». Якість алгоритму оцінюють тим, як точно він може правильно класифікувати нові фото із кішками та собаками (рисунок 3.1).

Завдання регресії пов'язані з безперервними даними. Один із прикладів, лінійна регресія, обчислює очікуване значення змінної y , ураховуючи конкретні значення x .

Більш утилітарні завдання машинного навчання використовують велику кількість змінних. Як приклад, нейронна мережа передбачає ціну квартири на основі її площі, розташування та доступності громадського транспорту. Алгоритм виконує роботу експерта, що розраховує ціну квартири, беручи за основу ті самі дані.

Отже, навчання з учителем найбільше підходить для завдань, коли є значний набір достовірних даних для навчання алгоритму. Але так буває далеко не завжди. Нестача даних – проблема, що найчастіше трапляється в машинному навчанні.

3.2. Навчання без учителя

Ідеально розмічені та чисті дані дістати нелегко. Тому іноді перед алгоритмом стоїть завдання знайти наперед невідомі відповіді. Ось де потрібне навчання без учителя.

У разі навчання без учителя (**unsupervised learning**) модель має набір даних і не має явних вказівок, що з ним робити. Нейронна мережа намагається самостійно знайти кореляції даних, витягуючи корисні ознаки та аналізуючи їх (рисунок).

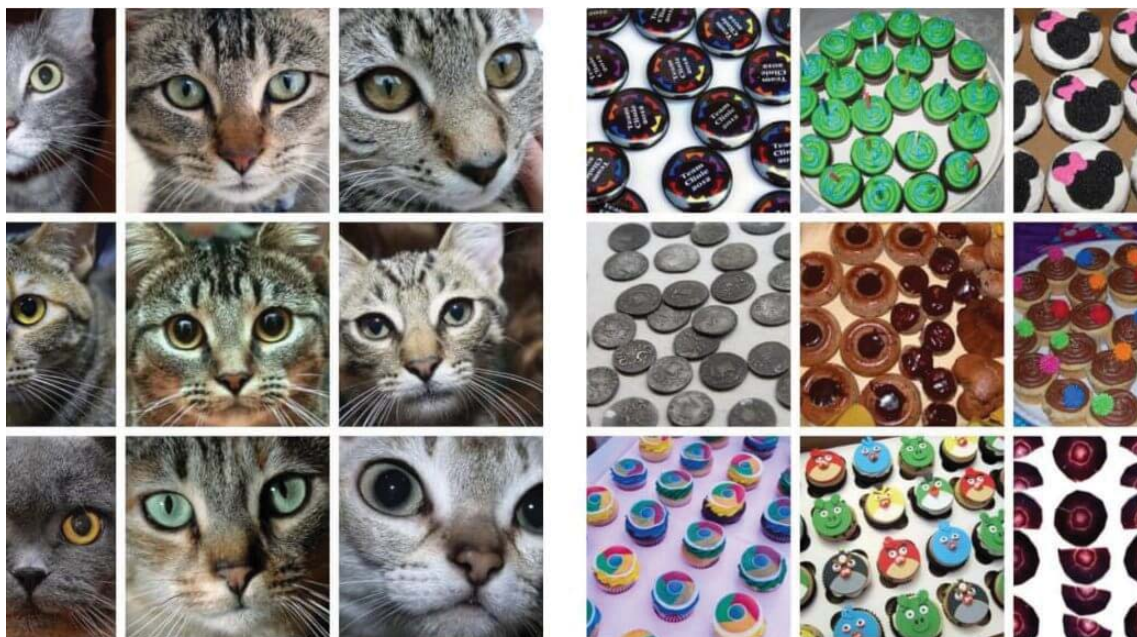


Рисунок 3.2 – Кластеризація даних з використанням загальних ознак

Залежно від завдання модель систематизує дані по-різному.

Кластеризація. Навіть без спеціальних знань експерта-орнітолога можна подивитися на колекцію фотографій та розділити їх на групи за видами птахів, беручи до уваги колір пера, розмір або форму дзьоба. Саме в цьому полягає кластеризація – найпоширені-

ше завдання для навчання без учителя. Алгоритм підбирає схожі дані, знаходячи загальні ознаки та групує їх разом.

Виявлення аномалій. Банки можуть виявити шахрайські операції, виявляючи незвичайні дії в купівельній поведінці клієнтів. Наприклад, підозріло, якщо одну кредитну картку використовують у Каліфорнії та Данії в той самий день. У схожий спосіб навчання без учителя використовують для знаходження викидів даних.

Асоціації. Ви берете в онлайн-магазині підгузки, яблучне пюре та дитячу склянку-непроливайку, а сайт порекомендує вам додати нагрудник та радіоняню до замовлення. Це приклад асоціацій: деякі характеристики об'єкта корелюють з іншими ознаками. Розглядаючи пару ключових ознак об'єкта, модель може передбачити інші, із якими існує зв'язок.

Автоенкодери. Автоенкодери приймають вхідні дані, кодуєть їх, а потім намагаються відтворити початкові дані з одержаного коду. Не так багато реальних ситуацій, коли використовують простий автоенкодер. Але необхідно додати шари та можливості розширяться: використовуючи зашумлені та вихідні версії зображень для навчання, автоенкодери можуть видаляти шум із відеоданих, зображень або медичних сканів, щоб підвищити якість даних.

У навчанні без учителя складно обчислити точність алгоритму, оскільки в даних відсутні «правильні відповіді» чи мітки. У таких ситуаціях, надаючи моделі свободу дій для пошуку залежностей, можна одержати добрі результати.

3.3. Навчання з частковим залученням учителя

Це золота середина. Навчання нейромережі з частковим залученням учителя (**semi-supervised learning**) характеризується своєю назвою: навчальний датасет містить як розмічені, так і нерозмічені дані. Цей метод особливо корисний, коли важко витягти з даних важливі ознаки або розмітити всі об'єкти.

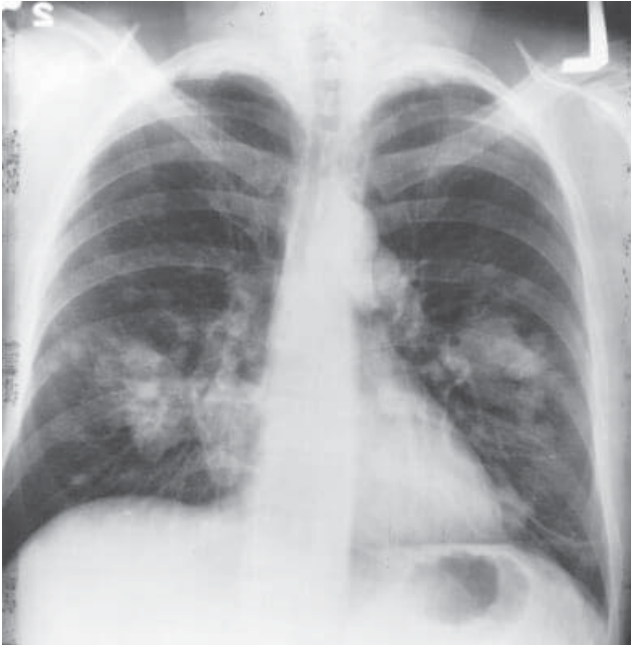


Рисунок 3.3 – Типовий знімок легень

Навчання з частковим залученням учителя часто використовують для вирішення медичних завдань, де невелика кількість розмічених даних може привести до значного підвищення точності. Цей метод машинного навчання є поширеним для аналізу медичних зображень, таких як скани комп'ютерної томографії або МРТ (рисунок 3.3). Досвідчений рентгенолог може розмітити невелику підмножину сканів, на яких виявлено пухлини та захворювання. Але вручну розмічати всі скани

є надто трудомістким й дорогим завданням. Тим не менш, нейронна мережа може одержати інформацію з невеликої частки розмічених даних і поліпшити точність передбачень порівняно з моделлю, що навчається винятково на нерозмічених даних.

Популярний метод навчання, що потребує невеликий набір розмічених даних, полягає у використанні генеративно-змагальної мережі чи GAN. Одна з мереж, генератор, намагається створити нові об'єкти даних, що імітують навчальну вибірку. Інша мережа, дискримінатор, оцінює, чи ці згенеровані дані є реальними чи підробленими. Мережі взаємодіють і циклічно вдосконалюються, оскільки дискримінатор намагається краще відокремлювати підробки від оригіналів, а генератор намагається створювати переконливі підробки навчання із частковим залученням учителя

3.4. Навчання з підкріпленням

Відеоігри ґрунтуються на системі стимулів. Завершіть рівень та одержіть нагороду. Переможіть усіх монстрів та заробите бонус. Потрапили в пастку – кінець гри – треба оминати. Ці стимули допомагають гравцям зрозуміти, як краще діяти в наступному

раунді гри. Без зворотного зв'язку люди просто приймали б випадкові рішення та сподівалися перейти на наступний ігровий рівень.

Навчання нейромережі з підкріпленням (**reinforcement learning**) діє за тим самим принципом. Відеоігри – популярне тестове середовище для досліджень (рисунок 3.4).

Агенти намагаються знайти оптимальний спосіб досягнення мети чи покращення продуктивності для конкретного середовища. Коли агент робить дії, що сприяють досягненню мети, він одержує нагороду. Глобальна мета – передбачати такі кроки, щоб заробити максимальну нагороду зрештою.

Під час ухвалення рішення агент вивчає зворотний зв'язок, нові тактики та рішення здатні привести до більшого виграшу. Цей підхід використовує довгострокову стратегію – як і в шахах: наступний найкращий хід може допомогти виграти зрештою. Тож агент намагається максимізувати сумарну нагороду.

Це ітеративний процес. Чим більше рівнів зі зворотного зв'язку, тим краще стає стратегія агента. Такий підхід є особливо корисним для навчання роботів, що керують автономними транспортними засобами або інвентарем на складі.

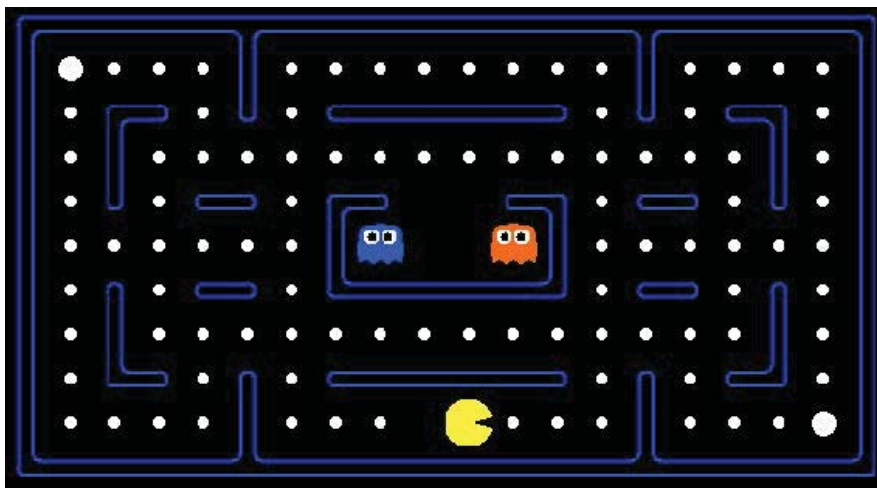


Рисунок 3.4 – Приклад гри для навчання з підкріпленням

Так само, як учні в школі, кожен алгоритм навчається по-різному. Але завдяки різноманітності доступних методів питання в тому, щоб вибрати відповідний і навчити вашу нейронну мережу розбиратися в середовищі.

3.5. Перенавчання нейронної мережі

Під час навчання нейронних мереж часто виникає проблема перенавчання (**overfitting**). Перенавчання, або надмірно близька підгонка – зайва точна відповідність нейронної мережі до конкретного набору навчальних прикладів, при якому мережа втрачає здатність до узагальнення.

Перенавчання виникає в разі занадто довгого навчання, недостатньої кількості навчальних прикладів або переускладненої структури нейронної мережі. Перенавчання пов'язано з тим, що вибір навчальної (тренувальної) множини є випадковим. Із перших кроків навчання відбувається зменшення похибки. На наступних кроках для зменшення похибки (цільової функції) параметри підлаштовують під особливості навчальної множини. Однак при цьому відбувається «підлаштування» не під загальні закономірності ряду, а під особливості його частини – навчальної підмножини. Водночас точність прогнозу зменшується.

Один із варіантів боротьби з перенавчанням мережі – поділ навчальної вибірки на дві множини (навчальну й тестову). На навчальній множині відбувається навчання нейронної мережі. На тестовій множині здійснюється перевірка побудованої моделі. Ці множини не повинні перетинатися.

З кожним кроком параметри моделі змінюються, однак постійне зменшення значення цільової функції відбувається саме на навчальній множині. У разі розбиття множини на дві можна спостерігати зміну похибки прогнозу на тестовій множині паралельно зі спостереженнями над навчальною множиною. Певну кількість кроків похибка прогнозу зменшується на обох множинах. Однак на певному етапі похибка на тестовій множині починає зростати, водночас похибка на навчальній множині продовжує зменшуватися. Цей момент вважають кінцем реального або справжнього навчання, з нього й починається перенавчання.

Тема 4. Персептрон. Навчання персептрону. Лінійна роздільність і персептронне подання

4.1. Персептрон Розенблата

Однією з перших штучних мереж, здатних до сприйняття та формування реакції до сприйнятого сигналу, з'явився персептрон Розенблата у 1957 р. Персептрон розглядався його автором не як конкретний технічний обчислювальний пристрій, а як модель роботи мозку. Потрібно зауважити, що після кількох десятиліть досліджень сучасні роботи зі штучних нейронних мереж рідко мають таку мету.

Найпростіший класичний персептрон містить нейроподібні елементи трьох типів (див. рис. 4.1):

- *S*-елементи формують сітківку сенсорних клітин, які приймають двійкові сигнали від зовнішнього світу;
- *A*-елементи подають шар асоціативних елементів, до яких надходять сигнали з *S*-елементів;
- *R*-елементи з фіксованими ваговими коефіцієнтами формують сигнал реакції персептрону на вхідний стимул.

Потрібно зауважити, що лише асоціативні елементи, які є формальними нейронами, виконують нелінійне оброблення інформації та мають вагові коефіцієнти зв'язків, які можуть змінюються.

Розенблат називав таку нейронну мережу тришаровою, проте за сучасною термінологією зазначену мережу зазвичай називають одношаровою, оскільки має лише один шар нейропроцесорних елементів. Одношаровий персептрон характеризується матрицею синаптичних зв'язків W від *S*-елементів до *A*-елементів. Елемент w_{ij} матриці W відповідає зв'язку, що веде від *i*-го *S*-елемента до *j*-го *A*-елемента.

У Корнельській авіаційній лабораторії було розроблено електротехнічну модель персептрона MARK-1, що містила 8 вихідних

S-елементи A-елементи R-елементи

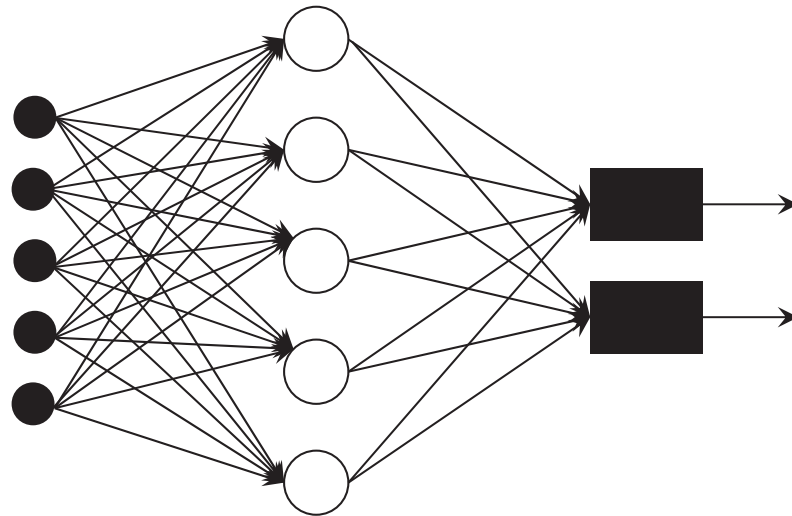


Рисунок 4.1 – Персептрон Розенблата

R-елементів та 512 *A*-елементів, які можна було поєднувати в різних комбінаціях. На цьому персептроні було проведено серію експериментів із розпізнавання букв алфавіту та геометричних образів.

У роботах Розенблата було зроблено висновок про те, що нейронна мережа розглянутої архітектури буде здатна до відтворення будь-якої логічної функції, проте, як було показано пізніше М. Мінським та С. Пейпертом (М. Minsky, S. Papert, 1969), цей висновок виявився неточним. Були виявлені принципові непереборні обмеження одношарових персептронів, і згодом стали переважно розглядати багатошаровий варіант персептрону, у якому є кілька шарів процесорних елементів.

На сьогодні одношаровий персептрон представляє здебільшого історичний інтерес, проте на його прикладі можуть бути вивчені основні поняття та прості алгоритми навчання нейронних мереж. Типовий одношаровий персептрон показано на рисунку 4.2. Потрібно зазначити, що персептрон може поділяти об'єкти лінійно-роздільної вибірки лише на два класи «0» та «1» і характеризується жорсткою пороговою функцією, як показано на рисунку 4.2. Такий персептрон здатен розпізнавати найпростіші образи. Окремий нейрон обчислює зважену суму елементів вхідного сигналу та пропускає результат через жорстку порогову функцію, вихід якої

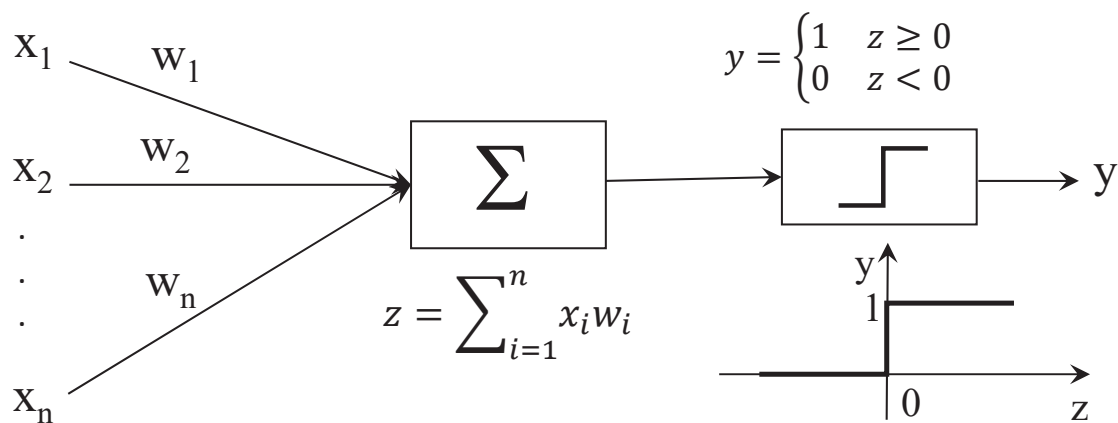


Рисунок 4.2 – Елементарний перцептрон Розенблата

дорівнює 1 чи 0. Залежно від значення вихідного сигналу приймають рішення:

- 1 – вхідний сигнал належить класу «1»;
- 0 – вхідний сигнал належить класу «0».

4.2. Навчання перцептрона

Навчання мережі полягає в підлаштуванні вагових коефіцієнтів $w_{i,j}$ кожного нейрона. Нехай є набір пари векторів (x_i, y_i) , $i = 1, \dots, N$, що подає навчальну вибірку. Будемо називати нейронну мережу такою, що навчена на цій навчальній вибірці, якщо в разі подачі на входи мережі кожного вектора x_i на виходах щоразу виходить відповідний вектор y_i .

Запропонований Ф. Розенблатом метод навчання полягає в ітераційному підстроюванні матриці вагових коефіцієнтів, яке послідовно зменшує помилку у вихідних векторах. Алгоритм містить кроки, описані далі.

Крок 1. Початкові значення вагових коефіцієнтів усіх нейронів фіксуються такими, що дорівнюють нулю: $\mathbf{w}_{t=0} = 0$, або обираються випадково.

Крок 2. Мережі на вхід подається вхідний вектор \mathbf{x}_i . Мережа генерує відповідь $\hat{y}_i \neq y_i$.

Крок 3. Розраховують вектор помилки $\delta_i = (y_i - \hat{y}_i)$, яку робить мережа на виході. Подальша ідея полягає в тому, що зміна

вектора вагових коефіцієнтів в області малих помилок повинна бути пропорційна помилці на виході, й дорівнює нулю, якщо помилка дорівнює нулю.

Крок 4. Модифікується вектор вагових коефіцієнтів за такою формулою $\mathbf{w}_{t+1} = \mathbf{w}_t + \eta \mathbf{x}_i \cdot \delta_i$. Тут $0 < \eta < 1$ – темп навчання.

Крок 5. Кроки 2 – 4 повторюються для всіх векторів навчальної вибірки. Один цикл послідовного пред'явлення усієї вибірки називають епохою.

Навчання завершується після кількох епох, коли:

- ітерації зійдуться, тобто вектор вагових коефіцієнтів перестає змінюватися;
- повна підсумована по всіх векторах абсолютна помилка стане меншою за деяке мале значення.

Формула, використувувана на кроці 4, враховує такі обставини:

- модифікуються лише ті компоненти матриці вагових коефіцієнтів, які відповідають ненульовим значенням входів;
- знак збільшення вагового коефіцієнта відповідає знаку помилки, тобто позитивна помилка ($\delta > 0$, значення виходу менше від необхідного) веде до посилення зв'язку;
- навчання кожного нейрона відбувається незалежно від навчання інших нейронів, що відповідає важливому з біологічної точки зору принципу локальності навчання.

Зазначений метод навчання був названий Ф. Розенблатом «методом корекції зі зворотним передаванням сигналу помилки». Пізніше ширше стала відома назва « δ -правило». Зазначений алгоритм належить до широкого класу алгоритмів навчання з учителем, оскільки відомі як вхідні вектори, так і необхідні вихідні вектори (є вчитель, здатний оцінити правильність відповіді учня).

Доведена Ф. Розенблатом теорема про збіжність навчання за δ -правилом говорить про те, що персептрон здатний навчитися будь-якого навчального набору, який він здатний уявити.

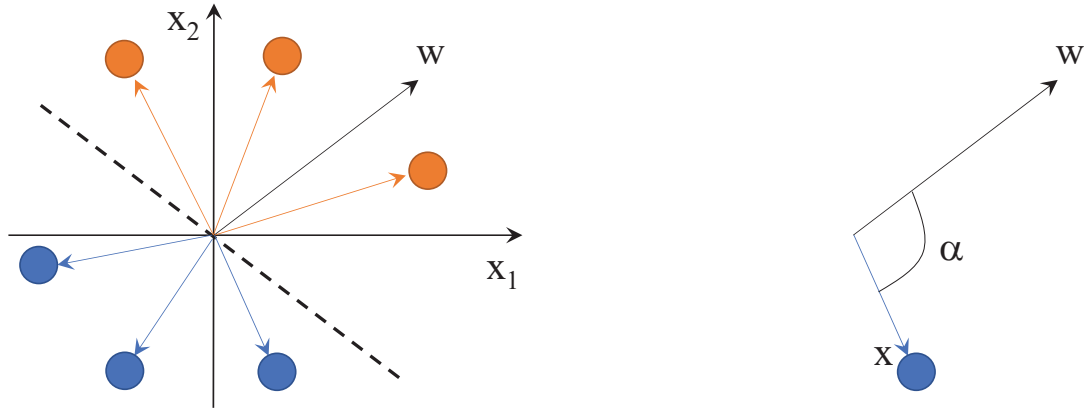


Рисунок 4.3 – Лінійно-роздільна двовимірна вибірка з елементів двох класів «0» та «1» (зліва) та кут між вектором вагових коефіцієнтів w та обраним об'єктом x

4.3. Геометричне подання алгоритму навчання

Розглянемо випадок лінійно-роздільної вибірки, поданої на рисунку 4.3. Пряма, яка розділяє всі об'єкти $\mathbf{x} = [x_1, x_2, \dots, x_n]$ з ваговими коефіцієнтами $\mathbf{w} = [w_1, w_2, \dots, w_n]$ на два класи, визначається рівнянням

$$\mathbf{x} \cdot \mathbf{w} = 0.$$

Відповідно до правила множення векторів одержуємо:

$$\mathbf{x} \cdot \mathbf{w} = \mathbf{w}^T \mathbf{x} = \sum_{i=1}^n w_i \cdot x_i.$$

Кут α між двома векторами \mathbf{w} та \mathbf{x} (див. рис. 4.3 справа) визначається скалярним добутком цих векторів відповідно до визначення

$$\cos(\alpha) = \frac{\mathbf{w}^T \mathbf{x}}{\|\mathbf{w}\| \|\mathbf{x}\|}.$$

Оскільки $\|\mathbf{w}\| \|\mathbf{x}\| = \text{const} > 0$, то можна подати

$$\cos(\alpha) \propto \mathbf{w}^T \mathbf{x}.$$

Отже, одержуємо певні закономірності

$$\begin{cases} \text{якщо } \mathbf{w}^T \mathbf{x} > 0 & \Rightarrow \cos(\alpha) > 0 & \Rightarrow \alpha < 90^0, \\ \text{якщо } \mathbf{w}^T \mathbf{x} < 0 & \Rightarrow \cos(\alpha) < 0 & \Rightarrow \alpha > 90^0. \end{cases}$$

Отже, для корегування значень вагових коефіцієнтів використовують такий підхід:

<p>для кожної епохи навчання t з урахуванням $\cos(\alpha_t) \propto \mathbf{w}_t^T \mathbf{x}$:</p> <p>якщо для об'єкта з класу «1» $\mathbf{w}_t^T \mathbf{x} < 0$, то</p> <p>$\mathbf{w}_{t+1} = \mathbf{w}_t + \eta \mathbf{x}$</p> <p>$\cos(\alpha_{t+1}) \propto \mathbf{w}_{t+1}^T \mathbf{x}$ $\propto (\mathbf{w}_t + \eta \mathbf{x})^T \mathbf{x}$ $\propto \mathbf{w}_t^T \mathbf{x} + \eta \mathbf{x}^T \mathbf{x}$ $\propto \cos(\alpha_t) + \eta \mathbf{x}^T \mathbf{x}$ $\Rightarrow \cos(\alpha_{t+1}) > \cos(\alpha_t)$;</p>	<p>якщо для об'єкта з класу «0» $\mathbf{w}_t^T \mathbf{x} \geq 0$, то</p> <p>$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \mathbf{x}$</p> <p>$\cos(\alpha_{t+1}) \propto \mathbf{w}_{t+1}^T \mathbf{x}$ $\propto (\mathbf{w}_t - \eta \mathbf{x})^T \mathbf{x}$ $\propto \mathbf{w}_t^T \mathbf{x} - \eta \mathbf{x}^T \mathbf{x}$ $\propto \cos(\alpha_t) - \eta \mathbf{x}^T \mathbf{x}$ $\Rightarrow \cos(\alpha_{t+1}) < \cos(\alpha_t)$.</p>
---	--

У такому разі пряма, що розділяє об'єкти на два класи «0» та «1» у двовимірній площині ознак (штрихова пряма на рисунку 4.3 зліва) може бути одержана так:

$$\mathbf{w}_t^T \mathbf{x} = w_1 x_1 + w_2 x_2 = 0 \quad \Rightarrow \quad x_2 = -\frac{w_1}{w_2} x_1.$$

Отже загалом рівняння прямої має вигляд

$$x_2 = kx_1 \quad k = -\frac{w_1}{w_2}.$$

Зрозуміло, що ця пряма завжди перетинає центр координат. Проте така ситуація є певною ідеалізацією. Розглянемо більш загальний випадок двовимірної вибірки на рисунку 4.4. Бачимо, що для цих вибірок пряма, що правильно поділяє класи не проходить центр координат та описується загальним рівнянням прямої

$$x_2 = kx_1 + b,$$

де параметр b задає зміщення (bias) прямої $x_2 = kx_1$ у певному напрямку. Для навчання персептрона розділяти такі об'єкти, сама модель персептрона повинна бути узагальнена, як показано на рисунку 4.5. Водночас метод навчання персептрона залишається

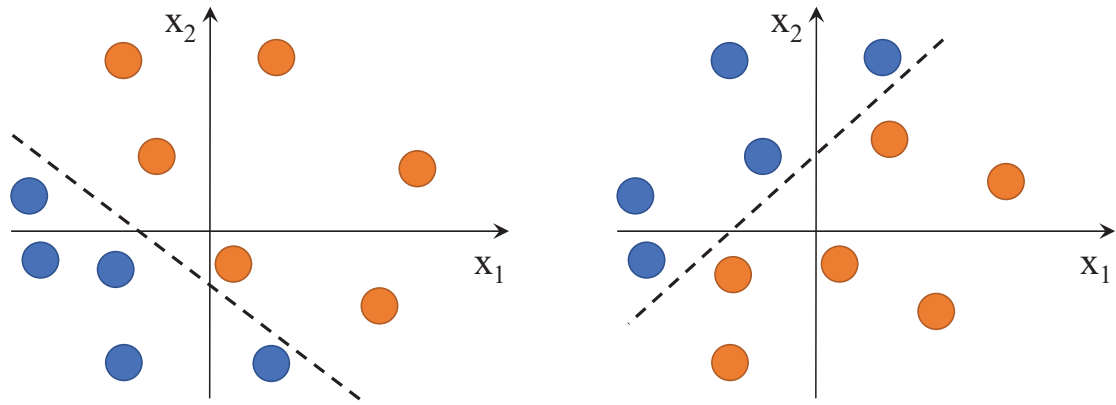


Рисунок 4.4 – Лінійно-роздільні двовимірні вибірки з елементів двох класів «0» та «1»

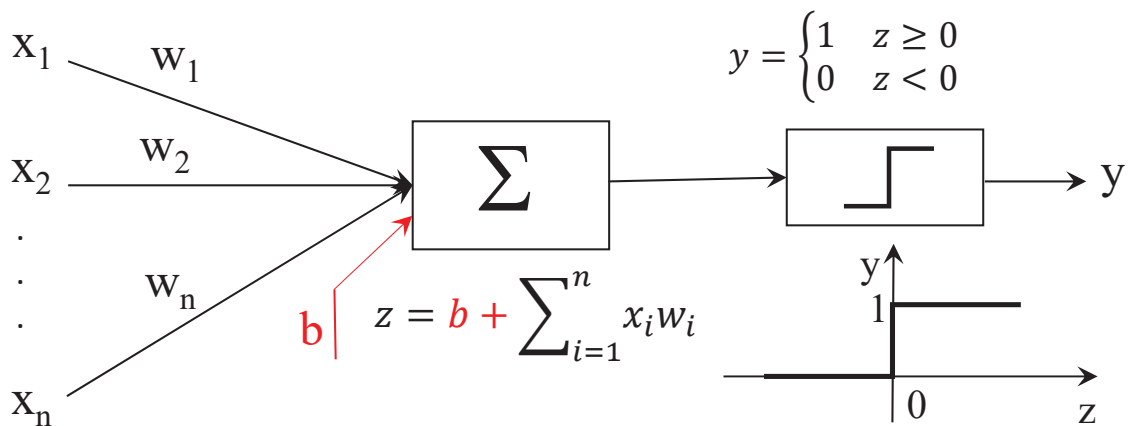


Рисунок 4.5 – Узагальнена модель елементарного перцептрона

незмінним, тоді як розширюється простір ознак об'єктів:

$$\mathbf{x} = [x_1, x_2, \dots, x_n] \Rightarrow \mathbf{x} = [x_0, x_1, x_2, \dots, x_n]$$

з урахуванням умови $x_0 = 1$ для всіх об'єктів вибірки; та модифікується вектор вагових коефіцієнтів

$$\mathbf{w} = [w_1, w_2, \dots, w_n] \Rightarrow \mathbf{w} = [w_0, w_1, w_2, \dots, w_n],$$

ураховуючи $w_0 = b$ відповідно до узагальненої моделі перцептрона (рис. 4.5). Тоді рівняння прямої у двовимірному просторі ознак матиме вигляд

$$w_0 x_0 + w_1 x_1 + w_2 x_2 = 0 \quad \Rightarrow \quad x_2 = -\frac{w_2}{w_1} x_1 - \frac{w_0}{w_1}$$

і зможе розділити об'єкти вибірки подані на рисунку 4.4.

4.4. Лінійна роздільність

Розглянемо три найпростіші приклади використання персептрона, а саме, логічні операції:

- кон'юнкція «ТА» («AND»);
- диз'юнкція «АБО» («OR»);
- виключне «АБО» (eXclusive OR, «XOR»).

• Кон'юнкція «ТА» («AND») на вихід видає просте додавання двох сигналів («логічне 1» або «логічне 0») та подає на вихід активний сигнал («логічна 1», «істина»), якщо активний сигнал наявний в обох входах. В інших випадках така умова на вихід дає пасивні сигнали («логічний 0», «хиба»).

• Диз'юнкція «АБО» («OR») дає на вихід пасивний сигнал («логічний 0», «хиба»), якщо обидва входи пасивні. У інших випадках така умова на вихід дає активні сигнали («логічна 1», «істина»).

• Виключне «АБО» («XOR») – логічний вентиль, який реалізує операцію виключна диз'юнкція або, що те ж саме, логічна еквівалентність з інверсією результату. Активний сигнал («логічна 1», «істина») на виході цього вентиля наявний тоді, коли активний сигнал знаходиться на одному й лише на одному з входів. Якщо ж на обох входах присутні пасивні сигнали («логічний 0», «хиба») або активний сигнал наявний більше, ніж на одному вході, на виході сигнал пасивний.

На рисунку 4.6 наведено таблиці істинності для всіх трьох логічних операцій та їх інтерпретація на площині ознак. Видно, що перші дві логічні операції є окремими прикладами загальної задачі класифікації з лінійно роздільними класами, яку подано на рисунку 4.4, і можуть бути розв'язані за допомогою узагальненого персептрона, модель якого подано на рисунку 4.5. Однак випадок виключеного «АБО» («XOR») є лінійно нероздільним і персептрон не здатен провести пряму в площині двох ознак, яка б повністю та без помилок розділила дані на два класи.

Лінійна нероздільність множини аргументів, що відповідають різним значенням функції означає, що функція виключеного «АБО» («XOR»), що так широко використовується в логічних пристроях, не може бути репрезентована формальним нейроном. Такі скромні можливості нейрона й послужили основою для критики

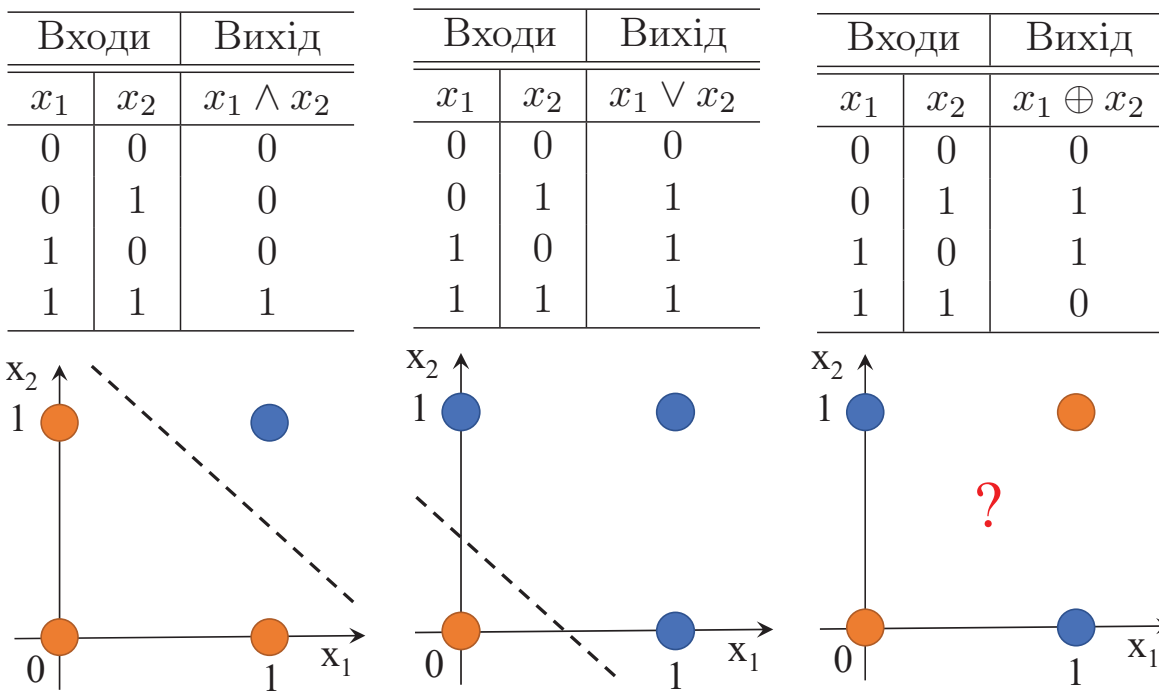


Рисунок 4.6 – Таблиці істинності для трьох логічних операцій «AND», «OR», «XOR» та їх інтерпретація на площині ознак

персептронного напрямку Ф. Розенблатта з боку М. Мінського та С. Пейперта.

За умови зростання числа аргументів ситуація катастрофічна: відносне число функцій, що характеризуються властивістю лінійної роздільності різко зменшується. А значить різко звужується клас функцій, який може бути реалізований персептроном (так званий клас функцій, що має властивість персептронної репрезентації). Відповідні дані наведено в таблиці 4.1. Видно, що одношаровий персептрон у край обмежений у своїх можливостях точно репрезентувати наперед задану логічну функцію. Потрібно зазначити, що пізніше, на початку 70-х років, це обмеження було подолано методом введення кількох шарів нейронів, проте критичне ставлення до класичного персептрона сильно заморозило загальне коло інтересу та наукових досліджень у галузі штучних нейронних мереж.

Кількість змінних N	Загальна кількість можливих логічних функцій ($= 2^{2^N}$)	Лінійно роздільних
1	4	4
2	16	14
3	256	104
4	65 536	1 882
5	> 1000000000	94 572

Таблиця 4.1 – Відсоток лінійно роздільних функцій від загальної кількості можливих логічних функцій

4.5. Сигмоїдальний нейрон

Нейрон сигмоїдального типу має структуру аналогічну до персептрона (див. рис. 4.5) із тією відмінністю, що функція активації є неперервною та може бути виражена у вигляді сигмоїдальної функції (див. таблицю 1.1)

$$f(z) = \frac{1}{1 + e^{-\beta z}}.$$

Параметр β вибирають індивідуально для кожної задачі. Сигмоїдну функцію за різних значень параметра β подано на рисунку 4.7а. Видно, що за малих значеннях параметра β функція є доволі плавна, і зі збільшенням параметра β вона стає більш різкою. У границі $\beta \rightarrow \infty$ сигмоїдна функція перетворюється на жорстку порогову функцію, ідентичну функції активації для персептрона.

Водночас основною відмінністю персептрона від сигмоїдального нейрона є те, що персептрон на виході дає бінарне значення 0 або 1, визначаючи належність кожного об'єкта досліджуваної вибірки до одного з класів «0» або «1»; тоді як сигмоїдальний нейрон як вихідний сигнал видає дійсне число в інтервалі $(0, 1)$ тим самим визначаючи ймовірність того чи іншого об'єкта належати до класу «0» або «1».

Важливою властивістю сигмоїдної функції є те, що її можна продиференціювати. Водночас похідна $df(z)/dz$ є також неперерв-

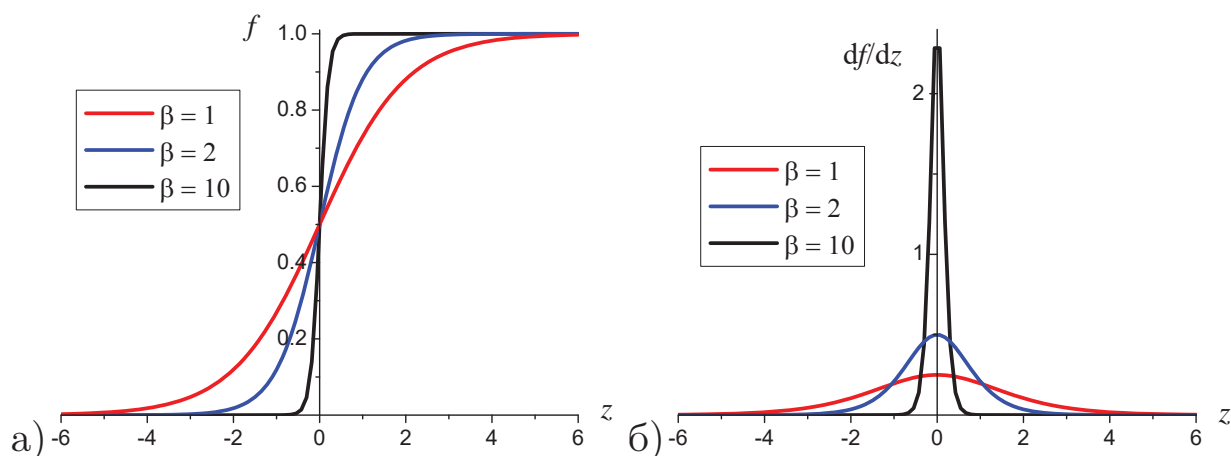


Рисунок 4.7 – Сигмоїдна функція активації: а) та її похідна б) за різних значень параметра β

на

$$\frac{df(z)}{dz} = \beta f(z) (1 - f(z)).$$

Залежність похідної від аргументу має дзвоноподібну форму, показану на рисунку 4.7б за різних значень параметра β . Сигмоїдний нейрон зазвичай навчається з учителем за принципом мінімізації цільової функції E , яку для j -того одиничного навчального набору (\mathbf{x}_j, y_j) визначають у такому вигляді:

$$E = \frac{1}{2}(\hat{y}_j - y_j)^2,$$

де

$$\hat{y}_j = f(z_j) = f\left(\sum_{i=0}^n w_i x_i\right).$$

Застосування неперервної активаційної функції дозволяє використати для навчання такого нейрона градієнтні методи. Найпростіше реалізувати метод градієнтного спуску відповідно до якого оновлення вектора вагових коефіцієнтів $\mathbf{w} = [w_0, w_1, \dots, w_n]^T$ відбувається в напрямку від'ємного градієнта цільової функції. Беручи похідну від цільової функції E за вектором вагових коефіцієнтів \mathbf{w} з урахуванням виразу для очікуваного значення відповіді \hat{y}_j одержуємо:

$$\nabla_j E = \frac{dE}{d\mathbf{w}} = \mathbf{x}_j (\hat{y}_j - y_j) \frac{df(z)}{dz}.$$

У такому разі значення вагових коефіцієнтів оновлюються для кожного j -го об'єкта відповідно до формули

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \mathbf{x}_j (\hat{y}_j - y_j) \frac{df(z)}{dz}, \quad z = \mathbf{w}^T \mathbf{x}_j$$

з темпом навчання $\alpha \in (0, 1)$.

4.6. Приклад застосування одношарового персептрона та сигмоїдального нейрона

Одношаровий персептрон та сигмоїдальний нейрон можна використати для задачі класифікації на два класи за умови лінійної роздільності класів у просторі ознак.

Приклад: існують відцифровані чорно-білі зображення:

«0» – чорний піксель, «1» – білий піксель.

Завдання: навчити персептрон та сигмоїдальний нейрон класифікувати зображення на два класи: відрізнити темні (більше нулів) та світлі (більше одиниць) рисунки.

Етапи розв'язання.

1. Згенерувати N наборів 0 та 1 довжиною M , наприклад,

$$\begin{aligned} & [0, 1, 1, 0, 1, 0, \dots], \\ & [1, 0, 0, 0, 1, 0, \dots], \\ & \quad \vdots \\ & [0, 0, 1, 1, 0, 1, \dots]. \end{aligned}$$

2. Визначити значення цільового вектора y_j для кожного об'єкта (набору) \mathbf{x}_j

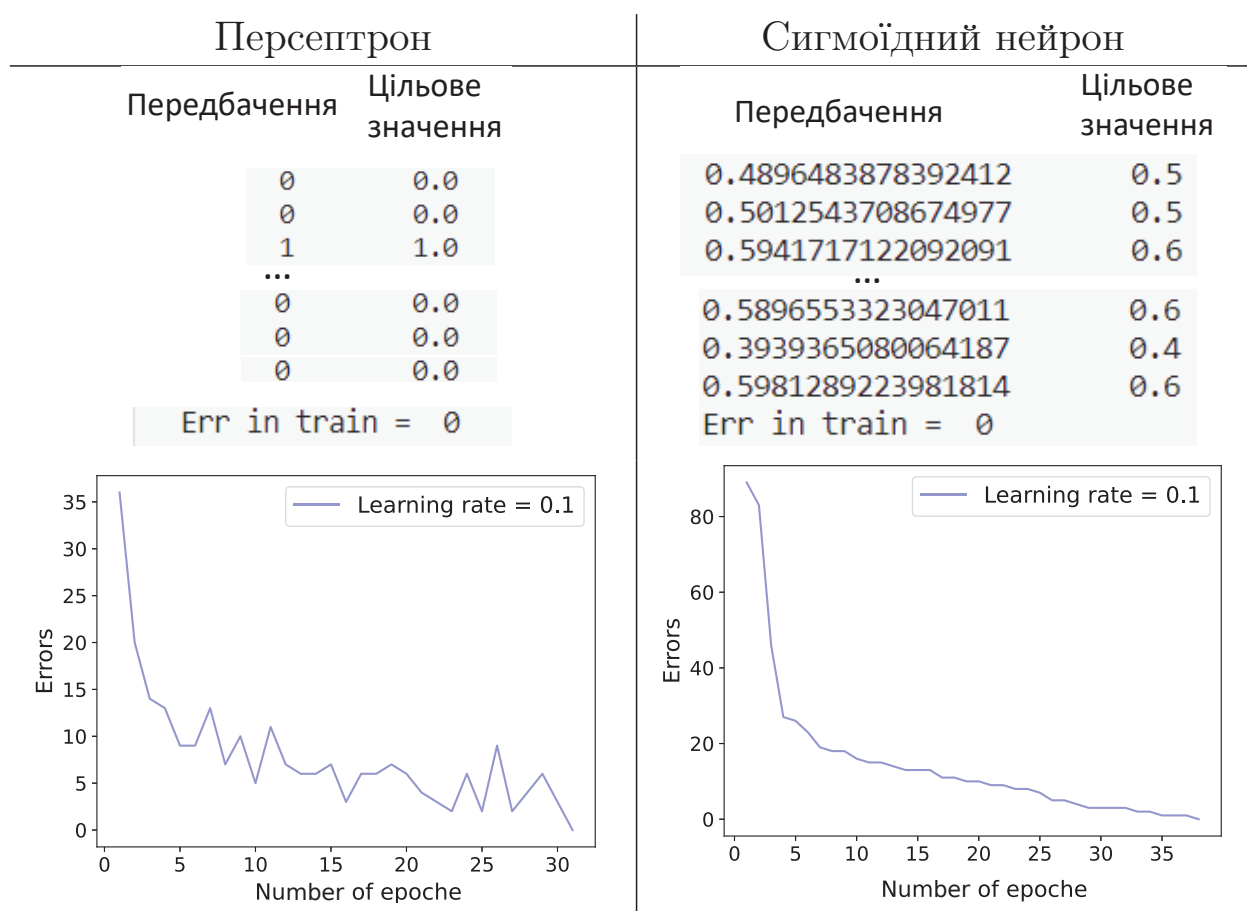
$$y_j = \begin{cases} 0, & \text{якщо } \frac{1}{M} \sum_{i=1}^M x_{ij} < 0.5 \\ 1, & \text{в іншому випадку} \end{cases}.$$

3. До кожного вектора \mathbf{x}_j додати ще один стовпчик (bias coefficient) до матриці ознак для зсуву розділяючої прямої відносно початку координат і встановити значення такими, що дорівнюють 1.

4. Поділити всю вибірку на навчальну та тестову.
5. Зафіксувати кількість епох навчання та крок навчання.
6. Побудувати алгоритм навчання персептрона та сигмоїдальний нейрона за δ -правилом.
7. Порахувати кількість неправильно класифікованих об'єктів на навчальній вибірці. Зупинити процес навчання коли всі об'єкти класифіковано вірно.
8. Перевірити роботу алгоритму на тестовій вибірці. Порахувати кількість неправильно класифікованих об'єктів.

Результат навчання.

Візуалізація залежності кількості помилок класифікації від номера епохи навчання:



Розділ 2. Методи навчання багат шарових нейронних мереж

Тема 5. Багат шарові нейронні мережі прямого поширення. Багат шаровий персептрон. Задача «XOR»

5.1. Багат шарова нейронна мережа

Багат шарова нейронна мережа складається з нейронів, які розміщені на різних рівнях, причому, окрім вхідного та вихідного шарів, є ще як мінімум один внутрішній (прихований) шар. Класичний багат шаровий персептрон, репрезентований Румельхартом, Гінтоном і Вільямсом, це нейронна мережа, яку використовували як для задач класифікації, так і задач регресії. Його можна описати такими складовими:

- лінійна функція, яка агрегує вхідні значення;
- сигмоїдна функція (функція активації);
- порогова функція для процесу класифікації та функція ідентичності для задач регресії;
- функція втрат, що обчислює загальну помилку мережі;
- процедура навчання для налаштування вагових коефіцієнтів мережі.

5.1.1. Лінійна функція

Функція лінійної агрегації така сама, як у персептрона та сигмоїдного нейрона. Але з кількома відмінностями, які змінюють визначення, оскільки ми маємо справу з кількома шарами та блоками оброблення вхідних сигналів. На відміну від одного нейрона, де ми мали справу з вектором вагових коефіцієнтів \mathbf{w} , кількість

елементів якого визначається кількістю вхідних сигналів та додатковим параметром зміщення в багатошарових мережах вагові коефіцієнти об'єднані в матрицю.

Розглянемо спрощений приклад мережі, яка складається з:

- 3-х блоків вхідних сигналів,
- 1-го прихованого шару з 2-ма нейронами,

як показано на рисунку 5.1. Вхідний вектор для нашого першого

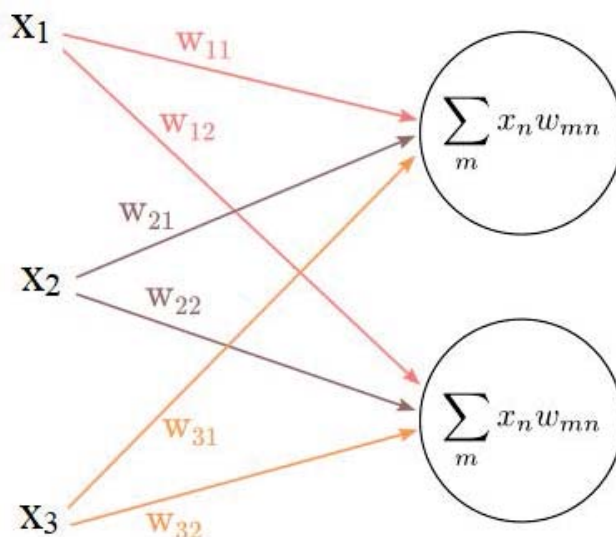


Рисунок 5.1 – Спрощений приклад мережі, що складається з трьох вхідів та одного прихованого шару з двох нейронів

навчального прикладу виглядатиме так: $\mathbf{x} = [x_1, x_2, x_3]$. Оскільки у нас є три вхідні сигнали, з'єднані з прихованими двома нейронами, то матриця вагових коефіцієнтів матиме розмір $[3 \times 2]$ і може бути репрезентована у вигляді

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix}.$$

Вихід лінійної функції дорівнює множенню вектора \mathbf{x} і матриці \mathbf{W} . Для виконання множення в цьому разі нам потрібно транспонувати матрицю \mathbf{W} щоб відповідати кількості стовпців у \mathbf{W} з кількістю рядків у \mathbf{x} , що в результаті дає

$$\mathbf{W}^T \times \mathbf{x} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} w_{11}x_1 & w_{12}x_2 & w_{13}x_3 \\ w_{21}x_1 & w_{22}x_2 & w_{23}x_3 \end{bmatrix}.$$

Цю матричну операцію можна подати так:

$$z_m = f(x_n, w_{mn}) = b + \sum_m x_n w_{mn}. \quad (5.1)$$

Індекс m визначає номер рядка в матриці \mathbf{W}^T та кількість рядків у \mathbf{z} , тоді як індекс n нумерує колонки в матриці \mathbf{W}^T та рядки у \mathbf{x} . Зміщення b додається до суми з метою спростити вивчення належного порогу для функції.

5.1.2. Сигмоїдна функція

Кожен елемент z_m вектора \mathbf{z} стає вхідним сигналом для сигмоїдної функції

$$a_m = \sigma(z_m) = \frac{1}{1 + e^{-z_m}}. \quad (5.2)$$

Результатом (виходом) функції $\sigma(z_m)$ є інший вектор

$$\mathbf{a}^T = [a_1, a_2]$$

розмірності m , елементи якого визначають один вхід для кожного нейрона прихованого шару.

Тут a означає «активація», що є поширеним способом позначення результату прихованих шарів. Цю сигмоїдну функцію, що «огортає» результат лінійної функції, зазвичай називають функцією активації. Ідея полягає в тому, що штучний нейрон «активується» більш-менш так само, як активується біологічний нейрон, коли надходить достатньо сильний вхідний сигнал. На цьому етапі в мережі можна вибрати багато різних нелінійних функцій (див. таб. 1.1). На жаль, принципового способу вибору функцій активації для прихованих шарів немає. Здебільшого це питання проб і помилок.

5.1.3. Результуюча функція

Для задач бінарної класифікації кожен вихідний блок реалізує **порогову функцію**

$$\hat{y} = f(a_m) = \begin{cases} +1, & \text{якщо } a_m > 0.5 \\ -1, & \text{в іншому випадку} \end{cases}.$$

Для задач регресії (проблем, які вимагають дійсного вихідного значення, як прогнозування доходу або тестових балів) кожен вихідний блок реалізує **функцію ідентичності**:

$$\hat{y} = f(a_m) = a_m.$$

Простіше кажучи, функція ідентифікації повертає те саме значення, що й вхід. Справа в тому, що a_m вже є результатом лінійної функції, отже, це значення, яке нам потрібно для таких задач.

Для задач багатокласової класифікації ми можемо використовувати функцію **softmax**:

$$\hat{y} = \sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \text{ for } j = 1, \dots, K.$$

5.1.4. Функція витрат

Функція витрат (цільова функція) є мірою якості мережі. Традиційно функція витрат зазвичай відноситься до вимірювання помилки для окремого навчального прикладу, функція вартості – до сукупної помилки для всього набору даних, а цільова функція є більш загальним терміном, що стосується будь-якої міри загальної помилки в мережі. Наприклад, «середня квадратична помилка», «сума квадратичної помилки» та «двійкова перехресна ентропія». Часто всі ці терміни використовують як синоніми: усі вони стосуються міри продуктивності мережі.

Для різних типів задач зазвичай використовують різні функції вартості. У своїй оригінальній роботі Румельхарт, Хінтон та Вільямс використовували суму квадратів помилок, визначену так:

$$E = \frac{1}{2} \sum_k (a_k - y_k)^2, \quad a_k = \hat{y}_k. \quad (5.3)$$

5.1.5. Пряме поширення

Усі нейронні мережі можна розділити на дві частини: фазу прямого поширення, де інформація «тече» вперед для обчислення прогнозів і помилок; і фаза зворотного поширення, де алгоритм

зворотного поширення обчислює похідні помилок та оновлює вагові коефіцієнти мережі. На рисунку 5.2 показано мережу з двома нейронами вхідного шару, трьома нейронами одного прихованого шару блоками та одним блоком виходу.

Матричне множення вхідних сигналів та вагових коефіцієнтів нейронів прихованого шару

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}^T \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Лінійна функція

$$z_m = b + \sum_m x_n w_{mn}$$

Сигмоїдна активація

$$a_m = \sigma(z_m) = \frac{1}{1 + e^{-z_m}}$$

Матричне множення вхідних сигналів нейронів прихованого шару та нейрона вихідного шару

$$\begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

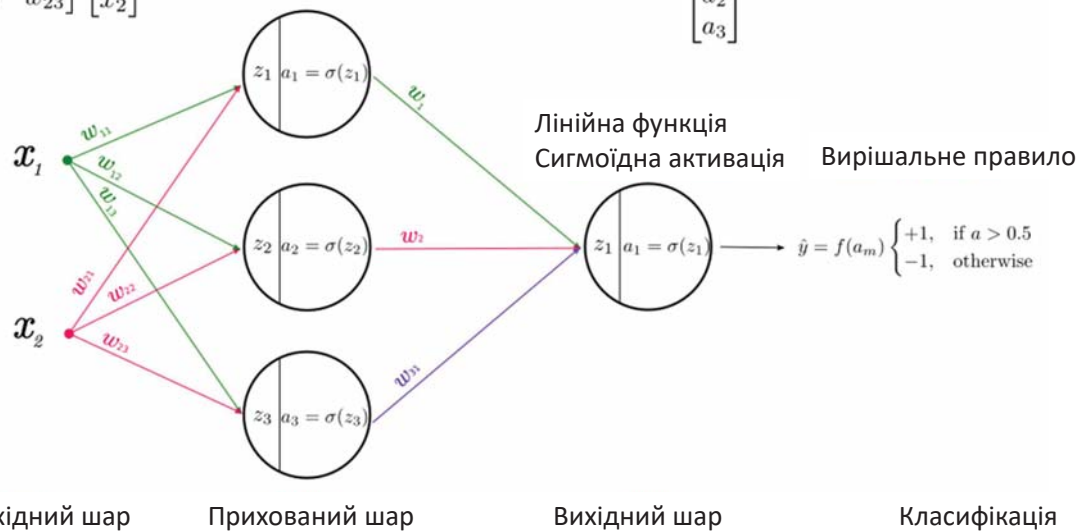


Рисунок 5.2 – Мережа прямого поширення з двома нейронами вхідного шару, трьома нейронами одного прихованого шару блоками та одним блоком виходу

Фаза прямого поширення складається з «ланцюжка» всіх кроків, які ми визначили досі: лінійної, сигмоїдної та порогової функції. Позначимо лінійну функцію як $\lambda()$, сигмоїдну функцію як $\sigma()$, а порогову функцію як $\tau()$ та подамо мережу на рисунку 5.2 у такому вигляді:

$$\hat{y} = \tau \left(\sigma^{(2)} \left(\lambda^{(2)} \left(\sigma^{(1)} \left(\lambda^{(1)} (x_n w_{mn}) \right) \right) \right) \right)$$

Усі нейронні мережі можна репрезентувати у вигляді композиції функцій, де кожен крок є вкладеним у наступний крок. Наприклад, ми можемо додати додатковий прихований шар до мережі на рисунку 5.2

$$\hat{y} = \tau \left(\sigma^{(3)} \left(\lambda^{(3)} \left(\sigma^{(2)} \left(\lambda^{(2)} \left(\sigma^{(1)} \left(\lambda^{(1)} (x_n w_{mn}) \right) \right) \right) \right) \right) \right)$$

5.2. Двошаровий персептрон

Як було показано в попередній лекції, одношаровий персептрон здатен проводити класифікацію лінійно-роздільних класів, як, наприклад, логічні операції «AND» або «OR». Проте, на практиці частіше трапляються складніші завдання.

5.2.1. Задача лінійної нероздільності класів

Наприклад, уявімо, що класи наших образів розподілені так як показано на рисунку 5.3. Видно, що в цьому разі неможливо

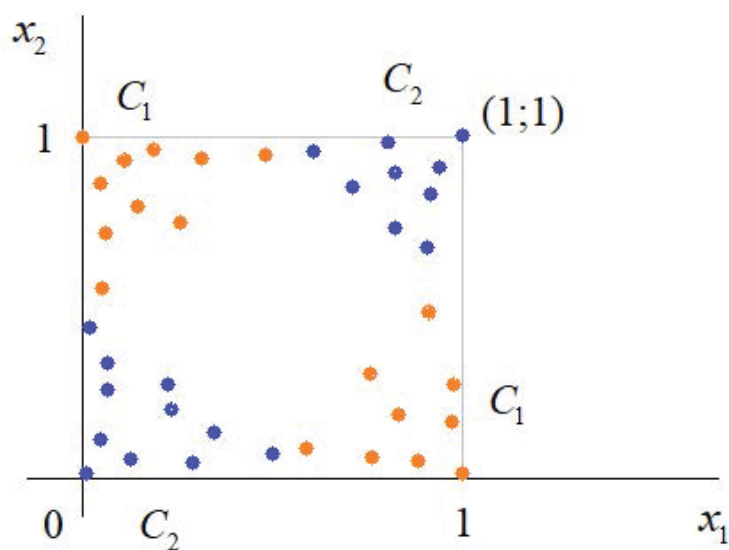


Рисунок 5.3 – Приклад вибірки з лінійно нероздільними класами

провести одну пряму для правильної класифікації. Для цього, наприклад, можна провести дві прямі, як показано на рисунку 5.4. У такому разі всі об'єкти, що знаходяться між цими прямими можна віднести до одного класу (C_1), а об'єкти, що знаходяться поза прямими – до другого класу (C_2).

Вибірка на рисунках 5.3 та 5.4 є узагальненням логічного правила виключеного «АБО» («XOR»), якщо на вхід подаються лише сигнали 0 або 1, та мітки класів приймають значення 0 або 1 (див. рис. 4.6 справа).

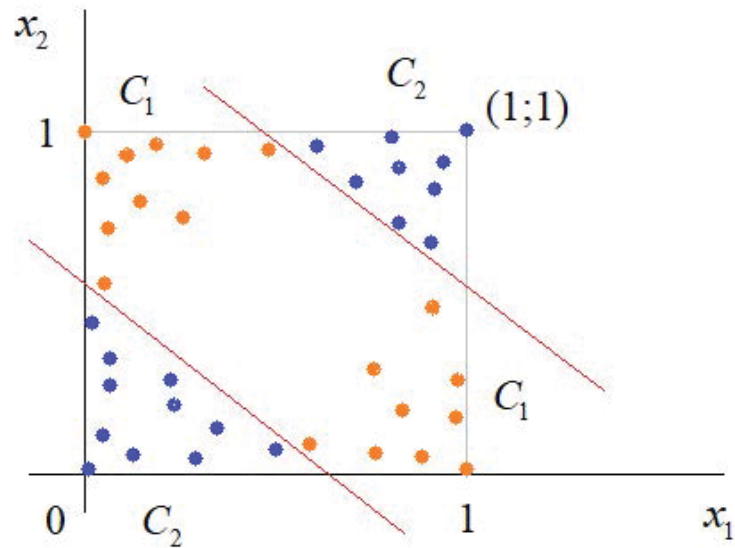


Рисунок 5.4 – Розділення вибірки за допомогою двох прямих

5.2.2. Побудова двошарового персептрона для задачі «XOR»

Легко показати, що для побудови класифікатора для логічної операції «XOR» достатньо використати двошаровий персептрон із двома нейронами на внутрішньому шарі й кожна розділова лінія може бути репрезентована окремим нейроном, а потім, результат їх класифікації об'єднується результуючим нейроном вихідного шару. Типову нейронну мережу подано на рисунку 5.5.

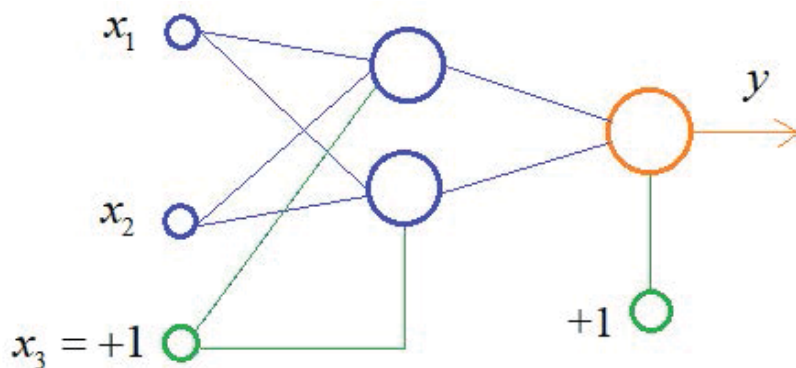


Рисунок 5.5 – Типова нейронна мережа для побудови класифікатора для логічної операції «XOR»

Активаційна функція кожного нейрона має вигляд як і для одношарового персептрона

$$f(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}.$$

Залишилося визначити значення вагових коефіцієнтів цієї нейронної мережі для вирішення поставленої задачі класифікації. Для початку, припустимо, що перший нейрон прихованого шару буде формувати роздільну пряму

$$x_2 = -1 \cdot x_1 + 1.5.$$

Ураховуючи нашу формулу для роздільної прямої для одношарового персептрона

$$x_2 = -\frac{w_1}{w_2}x_1 - \frac{w_3}{w_2},$$

вагові коефіцієнтів першого нейрона для x_1 , x_2 можна покласти рівними

$$w_1 = w_2 = 1,$$

а вагу третього зв'язку покласти рівним

$$w_3 = -b \cdot w_2 = -1.5.$$

У результаті ми одержали пряму, яка формує поділ на площині, як показано на рисунку 5.6. Другий нейрон прихованого шару буде формувати поділ прямою

$$x_2 = -1 \cdot x_1 + 0.5$$

і значення вагових коефіцієнтів його зв'язків можна взяти рівними

$$w_1 = w_2 = 1, \quad \text{та} \quad w_3 = -0.5.$$

Одержуємо картину поділу вибірки як показано на рисунку 5.7. Тепер нам потрібно об'єднати результати їх роботи, щоб вийшла область, яка повністю поділяє вибірку. Якщо результати просто скласти, то одержимо поділ, як показано на рисунку 5.8 зліва й

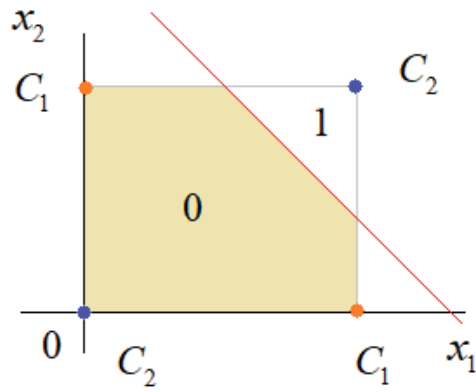


Рисунок 5.6 – Поділ вибірки «XOR» за допомогою першого нейрона прихованого шару

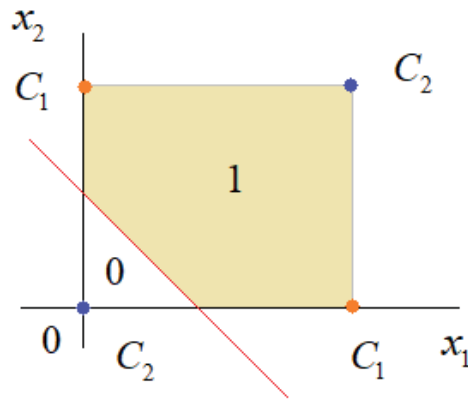


Рисунок 5.7 – Поділ вибірки «XOR» за допомогою другого нейрона прихованого шару.

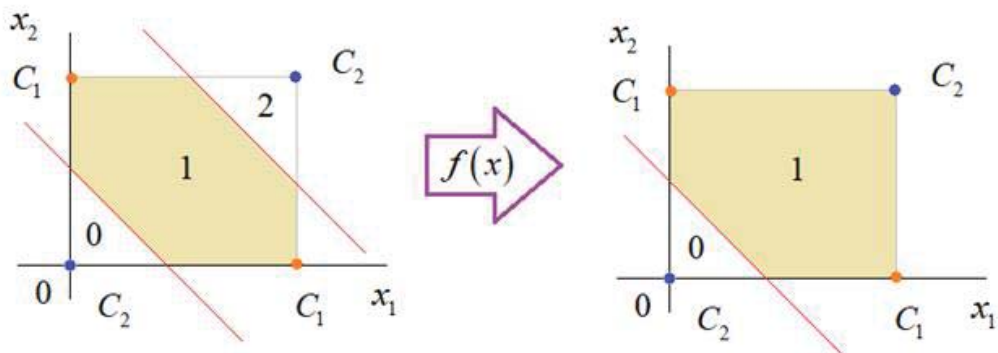


Рисунок 5.8 – Результат додавання роботи двох нейронів

на виході активаційної функції побачимо результат класифікації за допомогою лише другого нейрона (порівняй рисунок 5.8 справа та рисунок 5.7). Якщо з другого відняти перше, то на вході вихідно-

го нейрона отримуватимемо картину, як показано на рисунку 5.9. Для надійності змістимо ці значення на -0.5 та остаточно одер-

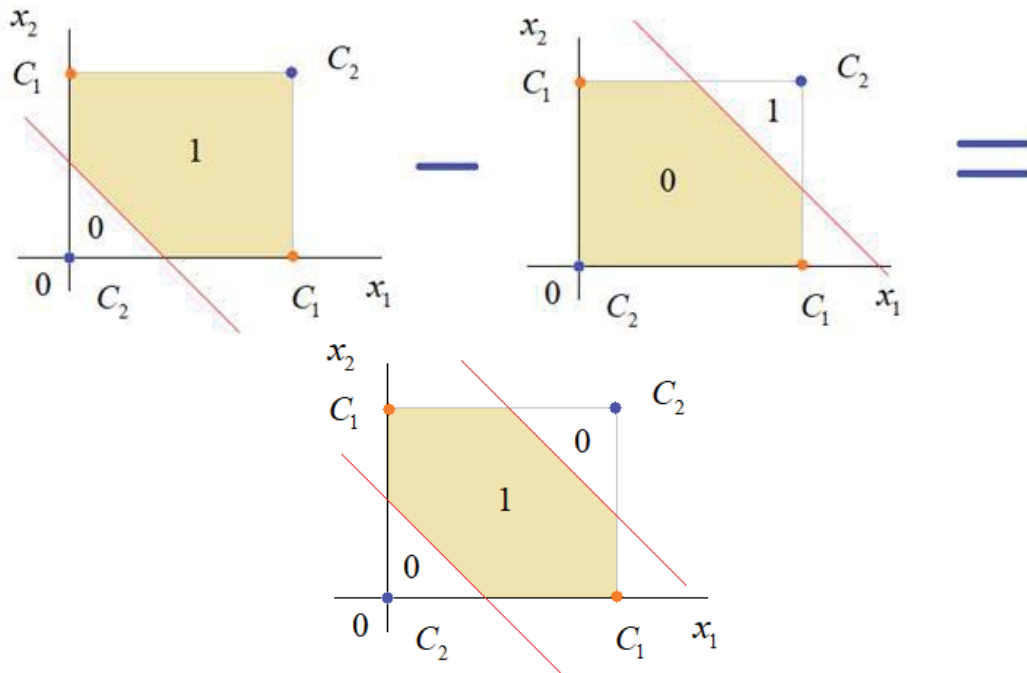


Рисунок 5.9 – Результат додавання роботи двох нейронів

жимо значення вагових коефіцієнтів як показано на рисунку 5.10. Цей приклад добре ілюструє, що додаючи нові нейрони, ми можемо

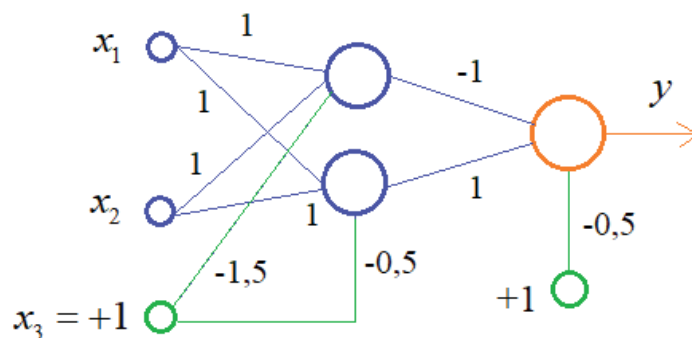


Рисунок 5.10 – Оптимальні значення вагових коефіцієнтів для класифікатора для проблеми «XOR»

одержувати все більш складні форми роздільних опуклих областей, одержані комбінацією ліній, що розділяють, або гіперплощин. Це приводить до більш складних схем класифікації, що йде далеко за межі лінійно-роздільних образів.

Тема 6. Метод градієнтного спуску та алгоритм зворотного поширення ПОМИЛКИ

Для того щоб нейронна мережа давала правильні відповіді, її потрібно навчати, однак наскільки б простою не була відповідь, її реалізація в сенсі простоти залишає бажати кращого. Існує кілька методів навчання нейронних мереж. Найбільш широко вживаними є такі:

- метод зворотного поширення (Backpropagation);
- метод пружного поширення (Resilient propagation або Rprop);
- генетичний Алгоритм (Genetic Algorithm).

Основним алгоритмом є метод зворотного поширення помилки, який використовує алгоритм градієнтного спуску.

6.1. Метод градієнтного спуску

6.1.1. Поверхня помилок

Градієнтний спуск – це спосіб знаходження локального мінімуму або максимуму функції за допомогою руху вздовж градієнта. Для початку з'ясуємо що таке градієнт і де він присутній у нашій нейронній мережі. Для цього розглянемо схематичний графік, де за віссю абсцис будуть значення вагові коефіцієнти певного нейрона w_{ij} , а за віссю ординат – помилка ϵ , що відповідає цьому ваговому коефіцієнту (рисунок 6.1).

Функція $f(w)$ є залежністю помилки від вибраної ваги. Оскільки нейронна мережа може бути порівняна із сильно нелінійною функцією, то й морфологія залежності помилки від вагового коефіцієнта, тобто поверхня помилок багат шарового персептрона, є не випуклою. Це означає, що існує кілька «долин», що характеризуються локальними мінімумами, а також глобальний мінімум. Локальні мінімуми – це точки мінімальної похибки для окремої ділян-

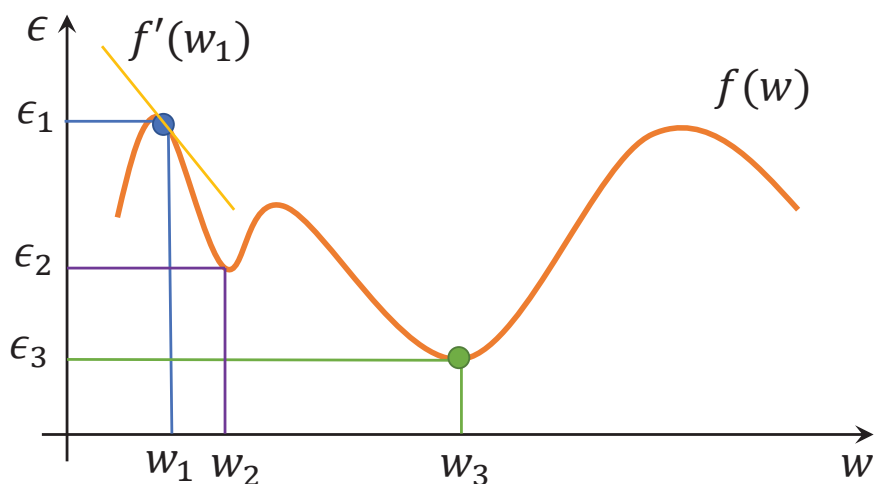


Рисунок 6.1 – Залежність помилки нейронної мережі від вагового коефіцієнта

ки поверхні помилок (наприклад значення $w = w_2$ на рисунку 6.1). Глобальний мінімум – це точка, де помилка (тобто значення функції вартості) є мінімальною для всієї поверхні помилок. На рисунку 6.1 це точка (w_3, ϵ_3) або, іншими словами, те місце, де графік підходить ближче до осі абсцис. Ця точка означатиме, що вибравши значення вагового коефіцієнта $w = w_3$ ми одержимо найменшу помилку ϵ_3 і внаслідок цього – найкращий результат з усіх можливих. Рисунок 6.2 ілюструє ці концепції на тривимірній поверхні. Вертикальна вісь репрезентує похибку поверхні, а дві інші осі – різні комбінації вагових коефіцієнтів для мережі. На рисунку видно, що різні комбінації вагових коефіцієнтів дають різні значення похибки.

Знайти точку глобального мінімуму на поверхні помилок можна за допомогою методу градієнтного спуску (жовтим відрізком на графіку на рисунку 6.1 позначений градієнт). Відповідно, у кожного вагового коефіцієнту в нейромережі буде свій графік і градієнт і в кожного треба знайти глобальний мінімум.

6.1.2. Ітеративна процедура пошуку мінімуму

Градієнт – це вектор, що визначає крутість схилу та вказує його напрямком щодо якоїсь із точок на кривій, у разі, коли w – число, або поверхні (гіперповерхні), коли w є вектором з двома чи

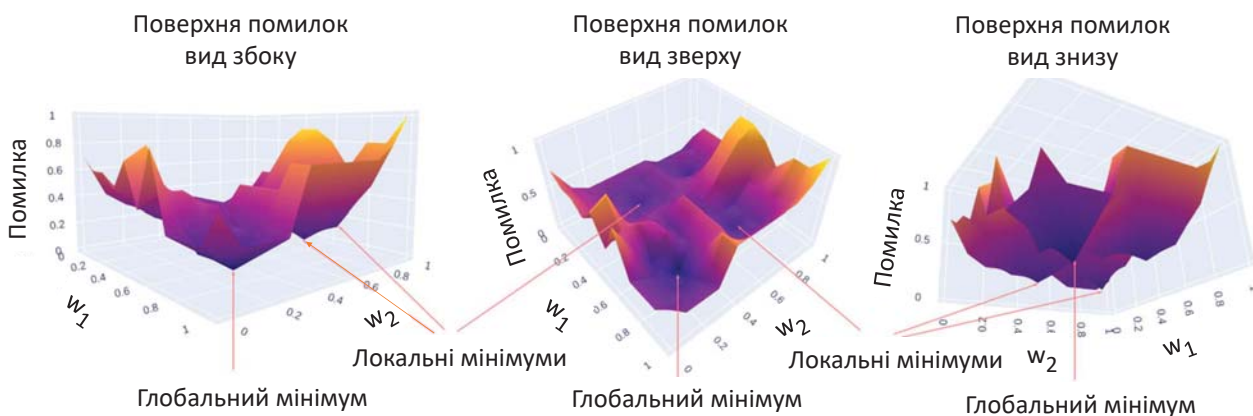


Рисунок 6.2 – Поверхня помилок багат шарового перцептрона

більше координатами. Щоб знайти градієнт потрібно взяти похідну від залежності $f(w)$ у точці, що визначається значенням певного вагового коефіцієнта (для прикладу w_1 , як показано на рисунку 6.1). Рухаючись у напрямку цього градієнта, ми будемо плавно скочуватися в низину (у найближчий мінімум залежності $f(w)$). Після певної кількості ітерацій руху в напрямку градієнта ми досягнемо цього мінімуму, як продемонстровано на схематичному рисунку 6.3, де пронумеровано ітерації процедури спуску по кривій $f(w)$.

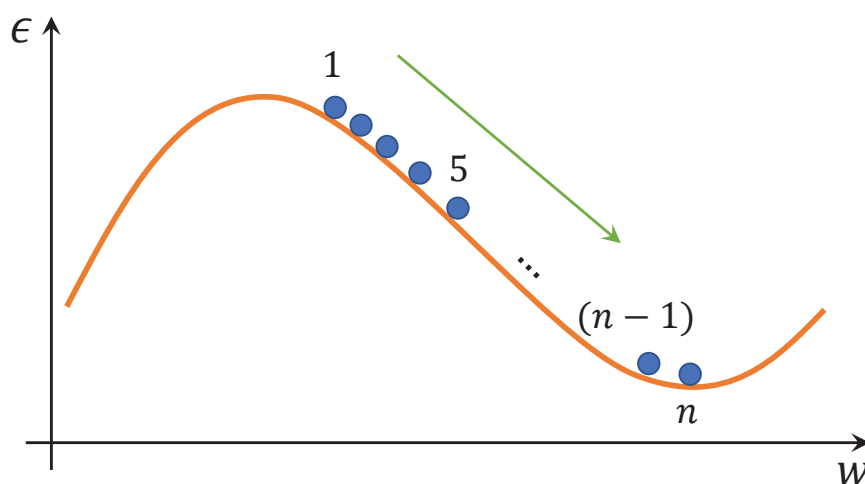


Рисунок 6.3 – Схема руху в напрямку градієнту до найближчого мінімуму функції $f(w)$

Оскільки під час ініціалізації вагових коефіцієнтів нейронної мережі їм присвоюють випадкові значення, то початкове зна-

чення помилки може бути досить високим, як, наприклад, ϵ_1 (див. рис. 6.1). Це значення і є точкою відліку. Стартуючи із цього значення потрібно спуститися вниз до мінімуму за допомогою градієнта. Для цього на шляху вниз у кожній точці обчислюють градієнт, що буде показувати напрям спуску й під час зміни нахилу коригувати його. Якщо схил буде прямим, то після n -ої кількості таких дій (ітерацій) ми дістанемося до мінімуму, але здебільшого графік функції буде хвилястий (як на рисунку 6.1) і досягнутий мінімум може видатися локальним. Потрапляння в локальний мінімум загрожує тим, що ітераційна процедура зміни значення вагового коефіцієнта w зупиниться на значенні w_2 , водночас помилка стане незмінною, набуваючи значення ϵ_2 . Отже, в результаті ітераційної процедури ми досягнемо певної рівноваги, яку не зможемо порушити. Такий результат призведе до того, що наша мережа не надасть нам правильної відповіді.

6.1.3. Збурення й відхилення від локальної рівноваги

Для того, щоб уникнути застрягання в локальних мінімумах використовують певне збурення – відхилення від рівноваги. Таке збурення дозволить покинути точку рівноваги й за наявності поблизу локального максимуму пройти його та продовжити ітераційну процедуру пошуку глобального мінімуму. Ілюстрацію впливу збурення (моменту) наведено на рисунку 6.4. Тут стрілками наведено рух щодо залежності помилки від вагового коефіцієнта.

Таким чином, за відсутності збурення ми «застрягнемо» в локальному мінімумі, як показано на рисунку 6.4а. Урахування збурення дозволить пройти локальний мінімум, запобігаючи зупинки роботи ітераційної процедури пошуку глобального мінімуму залежності помилки від вагового коефіцієнта. Проте, за умови недостатнього збурення ітеративна процедура поверне нас назад до локального мінімуму (червона стрілка), як показано на рисунку 6.4б. Завеликі значення збурення можуть призвести до того, що подолавши всі локальні мінімуми й діставшись глобального мінімуму ми можемо проскочити глобальний мінімум (див. рис. 6.4в) і, якщо поруч із ним є ще й локальний мінімум, потрапити до нього. У кінцевому результаті це не так важливо, оскільки рано чи пізно ми

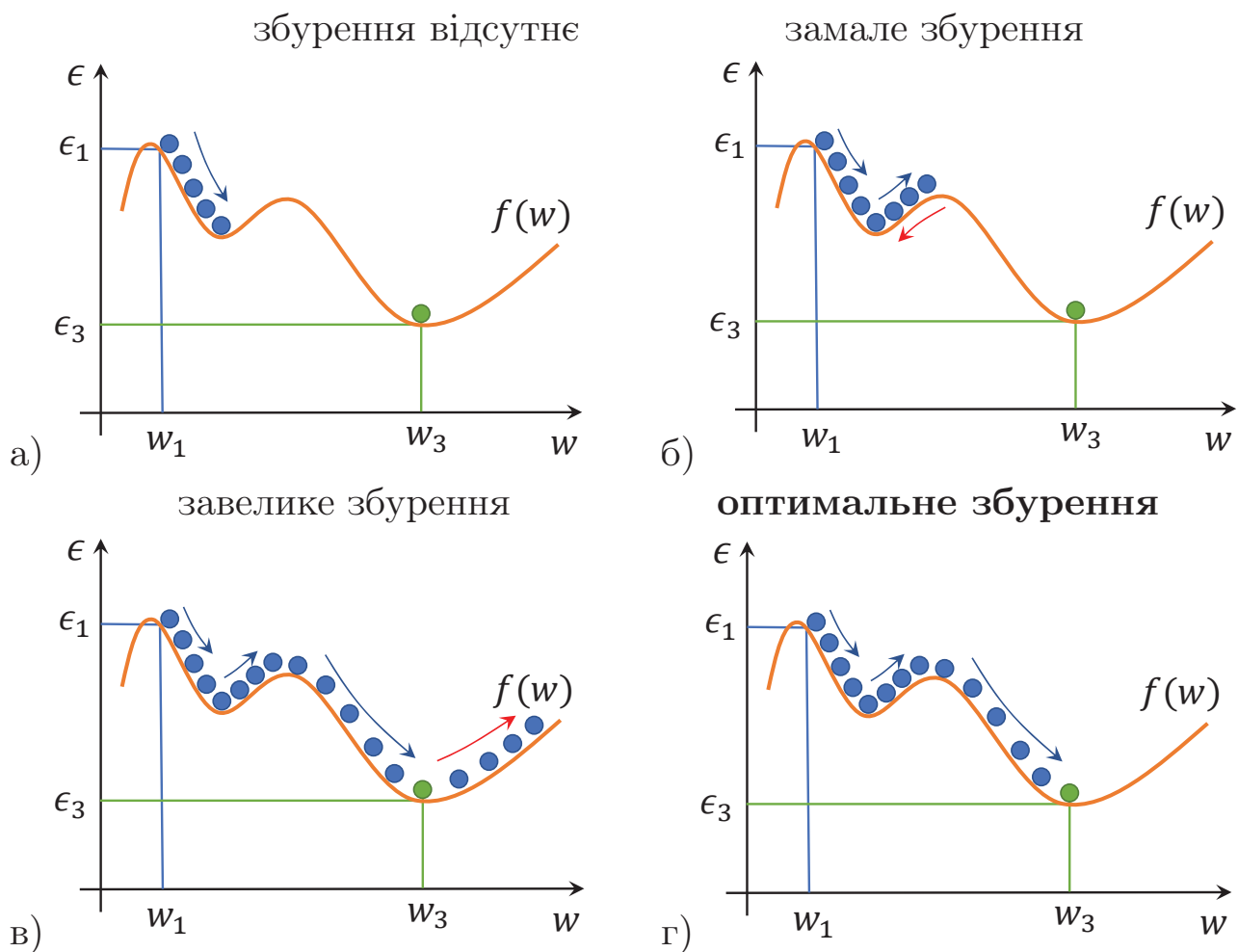


Рисунок 6.4 – Схема руху в напрямку градієнту до найближчого мінімуму функції $f(w)$: а) збурення відсутнє; б) замале збурення; в) завелике збурення; г) оптимальне збурення

все одно повернемося назад у глобальний мінімум, але необхідно пам'ятати, що чим більшим є значення збурення, тим більше буде розмах із яким ми будемо аналізувати морфологію залежності помилки від вагового коефіцієнта. Оптимальне значення збурення дозволить швидко і без зайвих рухів дістатися до глобального мінімуму, як наведено на рисунку 6.4г.

6.2. Алгоритм зворотного поширення помилки

Алгоритм зворотного поширення помилки використовують для пошуку набору вагових коефіцієнтів, що мінімізують помилку за допомогою градієнтного спуску.

Метою використання алгоритму зворотного поширення помилки є дізнатися, як змінюється помилка під час зміни вагових коефіцієнтів мережі на мізерну величину. Далі будемо використовувати верхній індекс L щоб індексувати зовнішню функцію в мережі. Наприклад, a^L індексує останню сигмоїдну функцію активації на вихідному рівні, $a^{(L-1)}$ індексує попередню функцію активації на прихованому шарі, і $x^{(L-2)}$ індексує об'єкти у вхідному шарі (які є єдиними речами в цьому шарі).

6.2.1. Пряме поширення сигналу мережею

Процедура корегування значень вагових коефіцієнтів та зміщень (у загальному випадку) починається з одержання результату роботи мережі та розрахунку помилки на об'єкті навчальної вибірки. Для прикладу розглянемо просту мережу з двома входами, двома прихованими шарами та одним виходом, схема якої показана на рисунку 6.5.

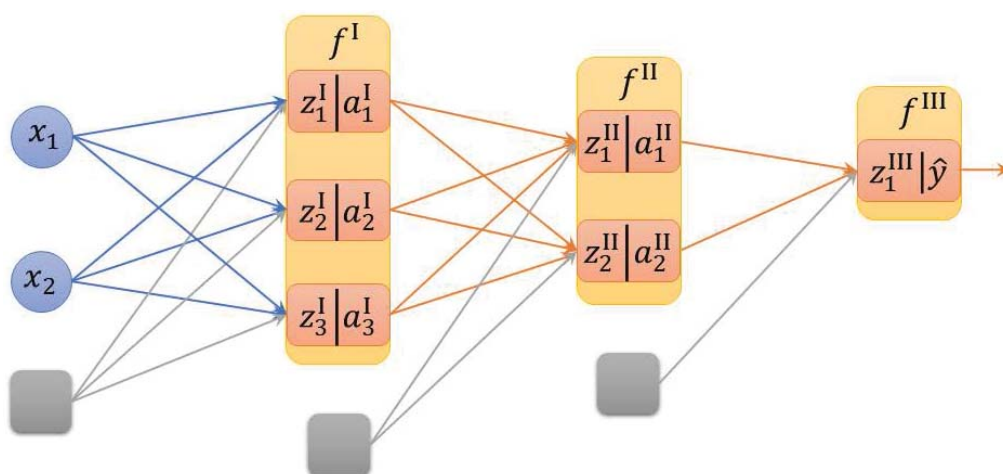
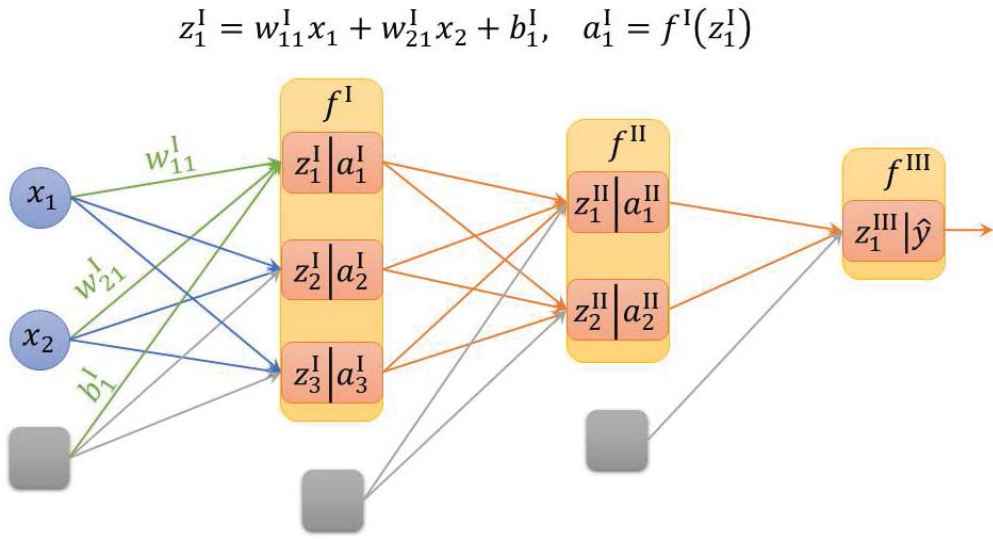
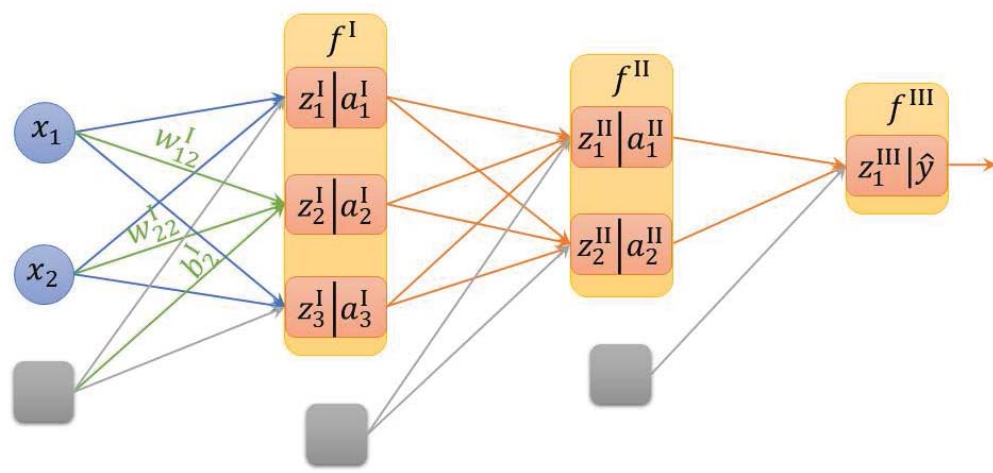


Рисунок 6.5 – Схема багатошарової нейронної мережі.



$$z_2^I = w_{12}^I x_1 + w_{22}^I x_2 + b_2^I, \quad a_2^I = f^I(z_2^I)$$



$$z_3^I = w_{13}^I x_1 + w_{23}^I x_2 + b_3^I, \quad a_3^I = f^I(z_3^I)$$

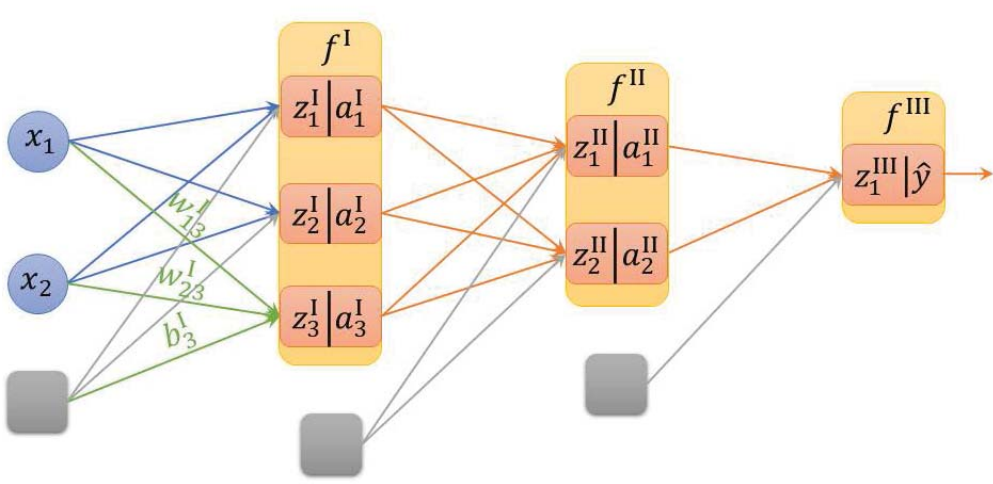
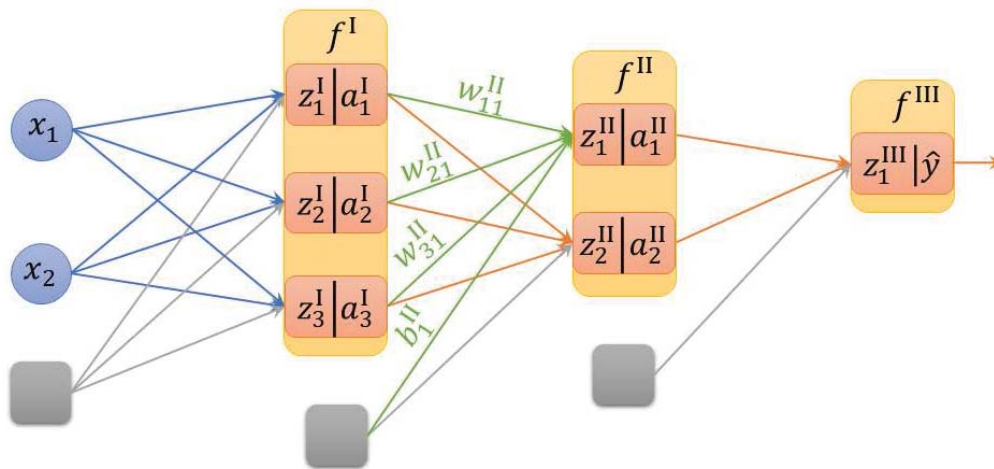


Рисунок 6.6 – Схема розрахунку входів та виходів нейронів першого прихованого шару

Сумуючи добутки вагових коефіцієнтів w_{ij}^I зі входами x_i та коефіцієнтів зміщення b_j^I одержуємо входи нейронів першого прихованого шару z_j^I ; застосовуючи активаційну функцію f^I першого прихованого шару одержуємо вихід a_j^I кожного j -го нейрона першого прихованого шару, як це продемонстровано на рисунку 6.6.

Проводячи аналогічну процедуру розраховуємо виходи нейронів другого прихованого шару a_j^{II} , як показано на рисунку 6.7, та вихід усієї мережі \hat{y} , як показано на рисунку 6.8.

$$z_1^{II} = w_{11}^{II} a_1^I + w_{21}^{II} a_2^I + w_{31}^{II} a_3^I + b_1^{II}, \quad a_1^{II} = f^{II}(z_1^{II})$$



$$z_2^{II} = w_{12}^{II} a_1^I + w_{22}^{II} a_2^I + w_{32}^{II} a_3^I + b_2^{II}, \quad a_2^{II} = f^{II}(z_2^{II})$$

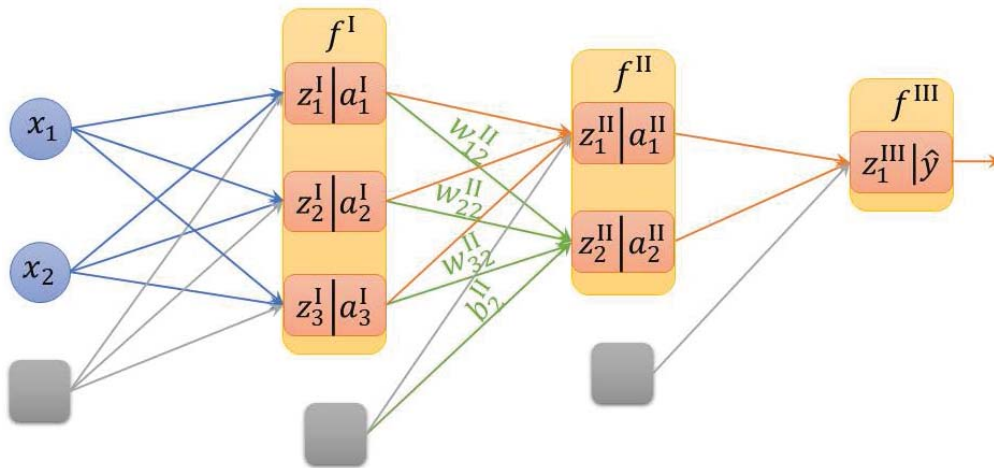


Рисунок 6.7 – Схема розрахунку входів та виходів другого прихованого шару

На початковому етапі, звісно, одержаний результат \hat{y} роботи мережі буде відрізнятись від бажаного – відомого значення y цільо-

$$z_1^{\text{III}} = w_{11}^{\text{III}}a_1^{\text{II}} + w_{21}^{\text{III}}a_2^{\text{II}} + b_1^{\text{III}}, \quad \hat{y} = f^{\text{III}}(z_1^{\text{III}})$$

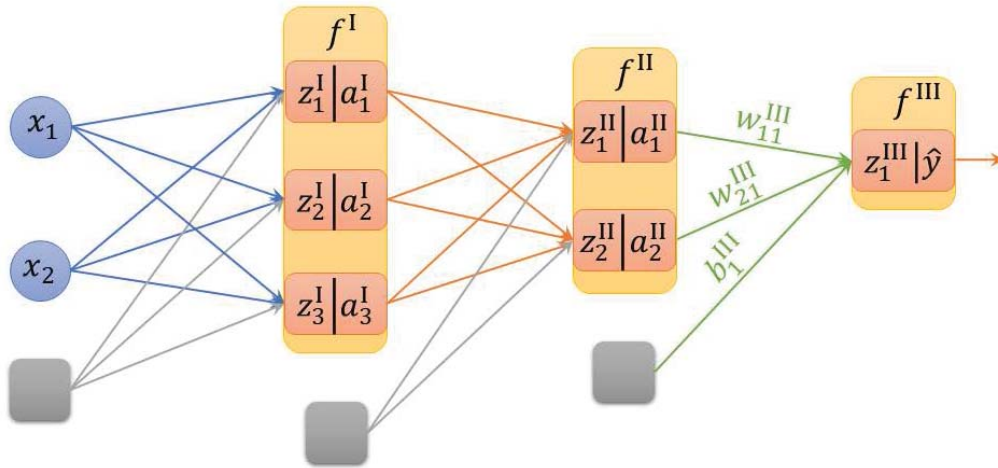


Рисунок 6.8 – Схема розрахунку результату роботи мережі

вого вектора y для цього об'єкту навчальної вибірки. Розраховуємо помилку мережі роботи δ , як показано на рисунку 6.9.

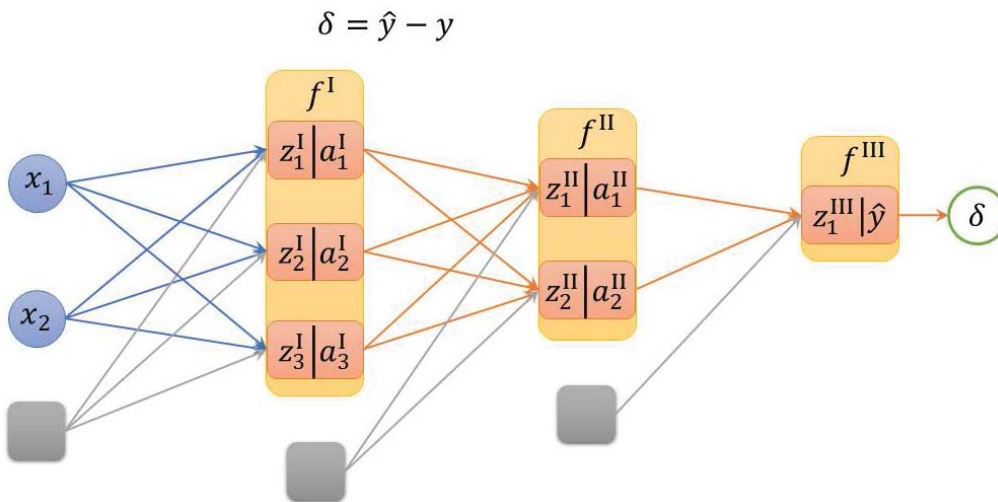


Рисунок 6.9 – Схема розрахунку помилки мережі

6.2.2. Зворотне поширення для багатошарової мережі з одним нейроном у кожному шарі

Для спрощення математичного подання далі не будемо використовувати символ суми та опустимо нижній індекс k . Багатошаровий перцептрон з одним входом, одним нейроном у прихованому

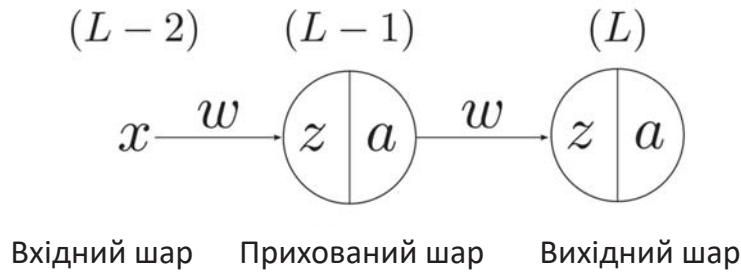


Рисунок 6.10 – Багатошарова мережа з одним входом, одним нейроном у прихованому шарі та одним виходом

шарі та одним виходом наведено на рисунку 6.10. Розберемо алгоритм зворотного поширення для цієї спрощеної мережі, а потім розширимо для випадку мережі з кількома нейронами. Для побудови алгоритму необхідно враховувати таке:

- помилка E залежить від значення сигмоїдної функції активації a ;
- значення функції активації сигмоїдної функції a залежить від значення лінійної функції z ;
- значення лінійної функції z залежить від величини вагових коефіцієнтів w .

Отже, можна простежити зміну залежності від вагових коефіцієнтів. Це означає, що ми повинні відповісти на три запитання в ланцюжку.

1. Як зміниться помилка E , коли ми змінимо активацію a на невелику величину?
2. Як зміниться активація a коли ми змінимо активацію z на невелику величину?
3. Як зміниться активація z коли ми змінимо значення вагових коефіцієнтів w на невелику величину?

Таку послідовність можна математично виразити за допомогою ланцюгового правила обчислення в такий спосіб

$$\frac{\partial E}{\partial w^{(L)}} = \frac{\partial E}{\partial a^{(L)}} \times \frac{\partial a^{(L)}}{\partial z^{(L)}} \times \frac{\partial z^{(L)}}{\partial w^{(L)}} \quad (6.1)$$

Це є правило диференціювання (взяття часткової похідної) від складної функції. Розглянемо кожен частину окремо. Похідна

похибки (5.3) від сигмоїдної функції активації a

$$\frac{\partial E}{\partial a^{(L)}} = a^{(L)} - y.$$

Далі похідна сигмоїдної функції (5.2) від лінійної функції z

$$\frac{\partial a^{(L)}}{\partial z^{(L)}} = a^{(L)} \left(1 - a^{(L)}\right).$$

Нарешті похідна лінійної функції (5.1) відносно вагових коефіцієнтів

$$\frac{\partial z^{(L)}}{\partial w^{(L)}} = a^{(L-1)}.$$

Отже, похідна (6.1) набуває вигляду

$$\frac{\partial E}{\partial w^{(L)}} = \left(a^{(L)} - y\right) \times a^{(L)} \left(1 - a^{(L)}\right) \times a^{(L-1)}. \quad (6.2)$$

На цьому етапі ми з'ясували, як змінюється помилка, коли ми змінюємо вагові коефіцієнти $w^{(L)}$, що з'єднують прихований і вихідний шари. Далі нам потрібно знати, як змінюється помилка, коли ми регулюємо вагові коефіцієнти $w^{(L)}$, що з'єднують вхідний і прихований шари. Для цього застосовуємо правило ланцюга знову й знову, поки не досягнемо результату

$$\frac{\partial E}{\partial w^{(L-1)}} = \frac{\partial E}{\partial a^{(L)}} \times \frac{\partial a^{(L)}}{\partial z^{(L)}} \times \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \times \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \times \frac{\partial z^{(L-1)}}{\partial w^{(L-1)}}. \quad (6.3)$$

Беручи частинкові похідні замість рівняння (6.3) одержуємо

$$\begin{aligned} \frac{\partial E}{\partial w^{(L-1)}} &= \left(a^{(L)} - y\right) \times a^{(L)} \left(1 - a^{(L)}\right) \\ &\times w^{(L)} \times a^{(L-1)} \left(1 - a^{(L-1)}\right) \times x^{(L-1)}. \end{aligned} \quad (6.4)$$

Відповідно до визначення лінійної функції (5.1) для похідної цільової функції (5.3) від коефіцієнта зміщення b для першого та другого шарів маємо

$$\frac{\partial E}{\partial b^{(L)}} = \left(a^{(L)} - y\right) \times a^{(L)} \left(1 - a^{(L)}\right) \quad (6.5)$$

та

$$\begin{aligned} \frac{\partial E}{\partial b^{(L-1)}} &= \left(a^{(L)} - y \right) \times a^{(L)} \left(1 - a^{(L)} \right) \\ &\times w^{(L)} \times a^{(L-1)} \left(1 - a^{(L-1)} \right). \end{aligned} \quad (6.6)$$

Отже, кожного разу, коли ми навчаємо нейронну мережу зі зворотним поширенням помилки, нам потрібно обчислити похідні для всіх вагових коефіцієнтів і зміщень усіх нейронів в усіх прихованих шарах і вихідного шару.

6.2.3. Багатошарова мережа з декількома нейронами в кожному шарі

Ми розглянули мережу з одним нейроном на кожному шарі, проте практично всі нейронні мережі мають більше ніж один нейрон на кожному шарі. Єдина відмінність між виразами, наведеними в попередньому розділі, полягає в кількох додаткових індексах. Наприклад, ми можемо використовувати індекс j щоб індексувати нейрони у вихідному шарі, індекс k для індексування нейронів у прихованому шарі та індекс i щоб індексувати нейрони у вхідному шарі. Також необхідно враховувати індекси для вагових коефіцієнтів. Для будь-якої мережі з декількома нейронами на кожному шарі кількість вагових коефіцієнтів буде більшою за кількість нейронів. Це означає, що нам знадобляться два індекси для ідентифікації кожного вагового коефіцієнта. Це видно в матриці вагових коефіцієнтів на рисунку 5.2. Ми будемо індексувати вагові коефіцієнти як $w_{n1,n2}$, де $n1$ – нейрон із якого іде сигнал, $n2$ – нейрон до якого іде сигнал. Наприклад, вагові коефіцієнти в шарі (L) одержують індекси w_{jk} , як показано на рисунку 6.11.

Ураховуючи всі ці позначення, наше початкове рівняння (6.1) для похідної похибки з ваговими коефіцієнтами для шару (L) перепишемо у вигляді

$$\frac{\partial E}{\partial w_{jk}^{(L)}} = \frac{\partial E_i}{\partial a_j^{(L)}} \times \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \times \frac{\partial z_j^{(L)}}{\partial w_{jk}^{(L)}}. \quad (6.7)$$

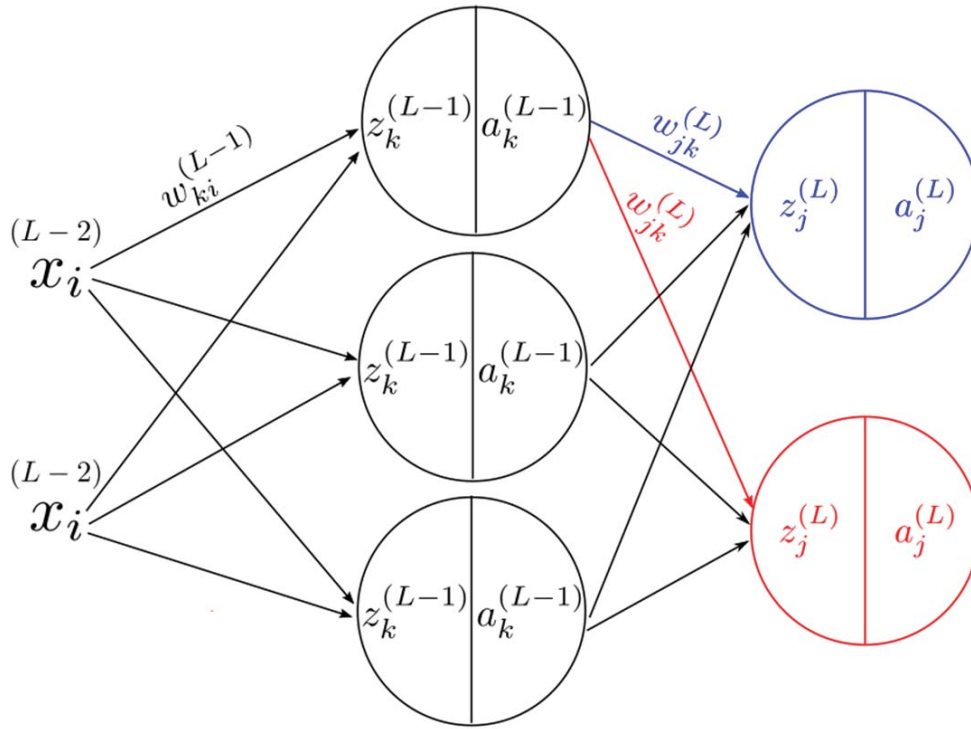


Рисунок 6.11 – Багатошаровий перцептрон із кількома нейронами на кожному шарі

З урахуванням усіх похідних маємо

$$\frac{\partial E}{\partial w_{jk}^{(L)}} = \left(a_j^{(L)} - y \right) \times a_j^{(L)} \left(1 - a_j^{(L)} \right) \times a_j^{(L-1)}. \quad (6.8)$$

Друге, що потрібно врахувати, це що кожна сигмоїдна активація a з $(L - 1)$ шарів впливає на помилку через кілька шляхів (припускаючи мережу з кількома вихідними нейронами). На рисунку 6.11 це показано синіми та червоними підключеннями до вихідного шару. Щоб відобразити це, ми додаємо символ підсумовування, і вираз для похідної помилки від сигмоїдної активації стає таким:

$$\frac{\partial E}{\partial a_k^{(L-1)}} = \sum_j \frac{\partial E}{\partial a_j^{(L)}} \times \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \times \frac{\partial z_j^{(L)}}{\partial a_k^{(L-1)}}. \quad (6.9)$$

Тепер, урахуваючи як нові індекси, так і підсумовування для $\frac{\partial E}{\partial a_k^{(L-1)}}$, ми можемо ще раз застосувати ланцюгове правило для обчислення

похідних похибок для вагових коефіцієнтів w на шарі $(L - 1)$ як

$$\begin{aligned} \frac{\partial E}{\partial w_{ki}^{(L-1)}} &= \left(\sum_j \frac{\partial E}{\partial a_j^{(L)}} \times \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \times \frac{\partial z_j^{(L)}}{\partial a_k^{(L-1)}} \right) \\ &\times \frac{\partial a_k^{(L-1)}}{\partial z_k^{(L-1)}} \times \frac{\partial z_k^{(L-1)}}{\partial w_k^{(L-1)}}. \end{aligned} \quad (6.10)$$

Замінюючи фактичні похідні для кожного виразу, ми одержуємо

$$\begin{aligned} \frac{\partial E}{\partial w_{ki}^{(L-1)}} &= \left(a_j^{(L)} - y \right) \times a_j^{(L)} \left(1 - a_j^{(L)} \right) \\ &\times w_{jk}^{(L)} \times a_k^{(L-1)} \left(1 - a_k^{(L-1)} \right) \times x_i^{L-1}. \end{aligned} \quad (6.11)$$

Аналогічно, використовуючи нові індекси для похідної помилки від зміщення одержуємо

$$\frac{\partial E}{\partial b_j^{(L)}} = \left(a_j^{(L)} - y \right) \times a_j^{(L)} \left(1 - a_j^{(L)} \right) \quad (6.12)$$

та

$$\begin{aligned} \frac{\partial E}{\partial b^{(L-1)}} &= \left(a_j^{(L)} - y \right) \times a_j^{(L)} \left(1 - a_j^{(L)} \right) \\ &\times w_{jk}^{(L)} \times a_k^{(L-1)} \left(1 - a_k^{(L-1)} \right). \end{aligned} \quad (6.13)$$

Отже, ми одержали вирази для всіх елементів алгоритму зворотного поширення помилки.

6.2.4. Оновлення вагових коефіцієнтів

Використовуючи обчислені градієнти для всіх вагових коефіцієнтів і зміщень можемо оновлювати значення вагових коефіцієнтів та зміщення. Це роблять за стандартної процедури градієнтного спуску. Так для вагових коефіцієнтів і параметра зміщення на шарі (L) маємо

$$\begin{aligned} w_{jk}^{(L)} &= w_{jk}^{(L)} - \eta \times \frac{\partial E}{\partial w_{jk}^{(L)}} \\ b^{(L)} &= b^{(L)} - \eta \times \frac{\partial E}{\partial b^{(L)}}. \end{aligned} \quad (6.14)$$

Для шару $(L - 1)$ маємо такі рівняння:

$$\begin{aligned} w_{ki}^{(L-1)} &= w_{ki}^{(L-1)} - \eta \times \frac{\partial E}{\partial w_{ki}^{(L-1)}} \\ b^{(L-1)} &= b^{(L-1)} - \eta \times \frac{\partial E}{\partial b^{(L-1)}}. \end{aligned} \quad (6.15)$$

Тут η визначає крок навчання.

Зворотне поширення помилки – ефективний та популярний алгоритм навчання багатшарових нейронних мереж, за його допомогою вирішують численні практичні завдання.

Багатокритеріальна задача оптимізації в методі зворотного поширення сприймається як набір однокритеріальних завдань – на кожній ітерації відбуваються зміни значень параметрів мережі, які поліпшують роботу лише з одним прикладом навчальної вибірки. Такий підхід істотно зменшує швидкість навчання. Модифікації алгоритму зворотного поширення пов'язані з використанням різних функцій похибки, різних процедур визначення напрямку та величини кроку.

6.2.5. Локальний градієнт нейронів кожного шару

Процедуру оновлення значень вагових коефіцієнтів w та зміщень b можна подати за допомогою уведення локального градієнта нейронів кожного шару. Аналіз формул для похідних (6.8) і (6.12) дозволяє зробити висновок про те, що вираз

$$\left(a_j^{(L)} - y \right) \times a_j^{(L)} \left(1 - a_j^{(L)} \right)$$

є локальним градієнтом для нейронів вихідного шару, який можна подати в спрощеному вигляді $\delta \left(df^{III} / dz_1^{III} \right)$ у випадку одного нейрону на вихідному шарі, як на рисунку 6.9 з урахуванням $a_j^{(L)} = \hat{y}$, та де-якої активаційної функції f^{III} . У такому разі в рівняннях (6.11) і (6.13) локальний градієнт для кожного нейрона k внутрішнього шару має вигляд

$$\delta \frac{df^{III}}{dz_1^{III}} w_{k1}^{III} \frac{df^{II}}{dz_k^{II}}. \quad (6.16)$$

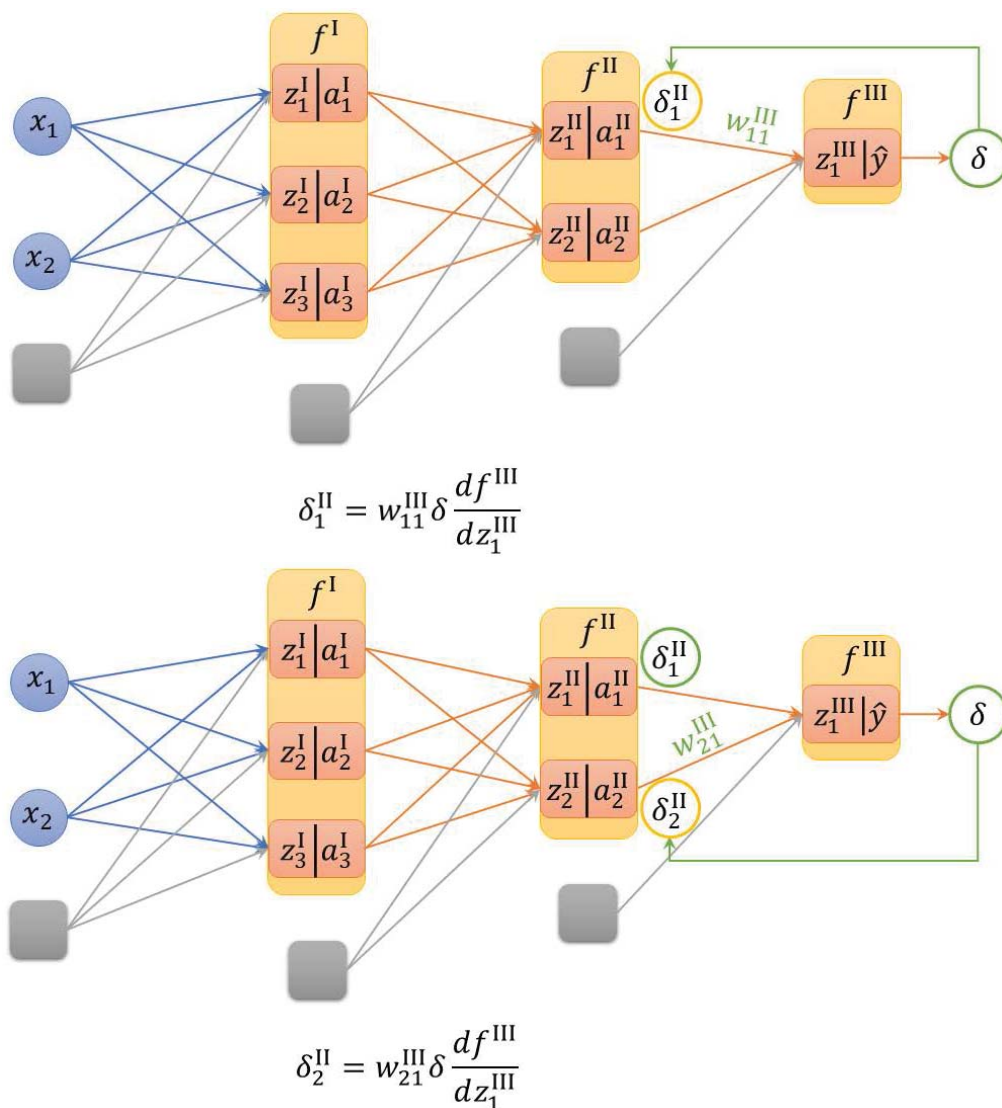


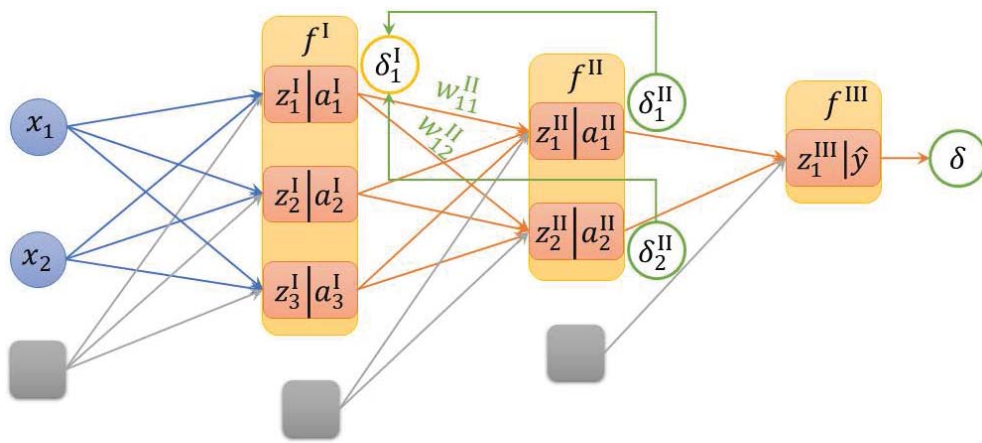
Рисунок 6.12 – Схема розрахунку помилок на кожному нейроні другого прихованого шару

Відповідно до одержаного виразу можна визначити, що помилка на кожному k -му нейроні на внутрішньому шарі з індексом II є

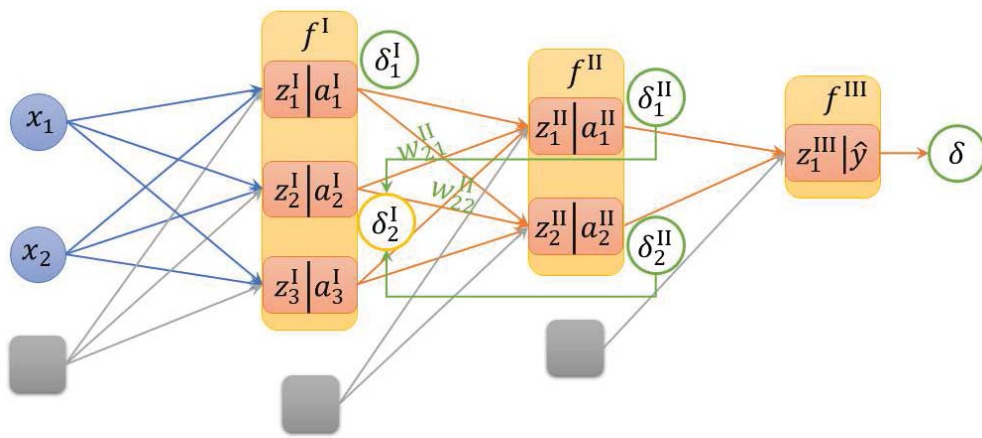
$$\delta_k^{\text{II}} = \delta \frac{df^{\text{III}}}{dz_1^{\text{III}}} w_{k1}^{\text{III}}. \quad (6.17)$$

Розрахунок помилок на кожному нейроні другого прихованого шару (передостаннього) для нейронної мережі з рисунка 6.5 відповідно до рівняння (6.17) проводять за схемою, поданою на рисунку 6.12.

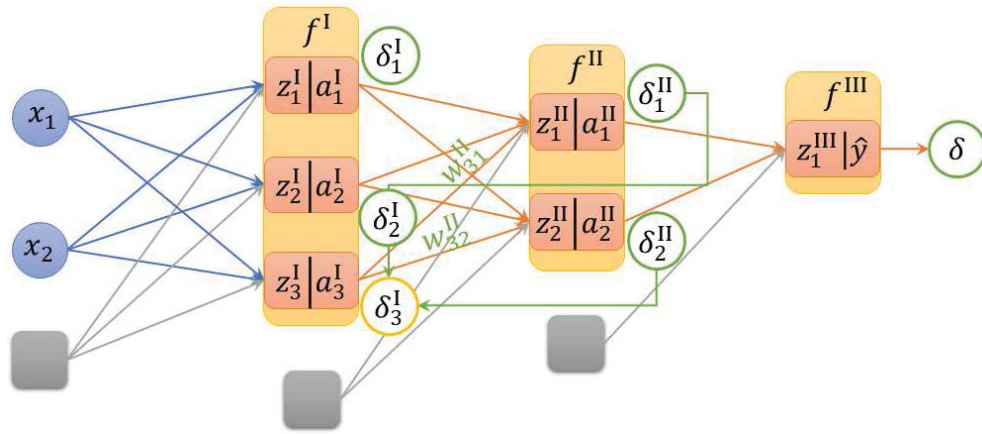
За аналогією, враховуючи, що на шарі з індексом II більше одного нейрона, проводимо розрахунок помилок δ_k^{I} для всіх k нейронів першого прихованого шару, як показано на рисунку 6.13.



$$\delta_1^I = w_{11}^{II} \delta_1^{II} \frac{df^{II}}{dz_1^{II}} + w_{12}^{II} \delta_2^{II} \frac{df^{II}}{dz_2^{II}}$$



$$\delta_2^I = w_{21}^{II} \delta_1^{II} \frac{df^{II}}{dz_1^{II}} + w_{22}^{II} \delta_2^{II} \frac{df^{II}}{dz_2^{II}}$$



$$\delta_3^I = w_{31}^{II} \delta_1^{II} \frac{df^{II}}{dz_1^{II}} + w_{32}^{II} \delta_2^{II} \frac{df^{II}}{dz_2^{II}}$$

Рисунок 6.13 – Схема розрахунку помилок на кожному нейроні першого прихованого шару

У такий спосіб ми можемо розрахувати всі помилки на усіх нейронах прихованих шарів (див. рис. 6.14).

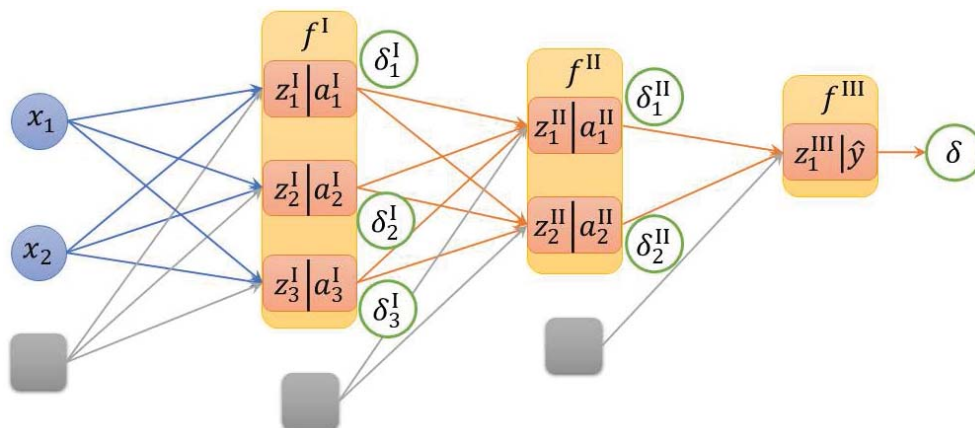
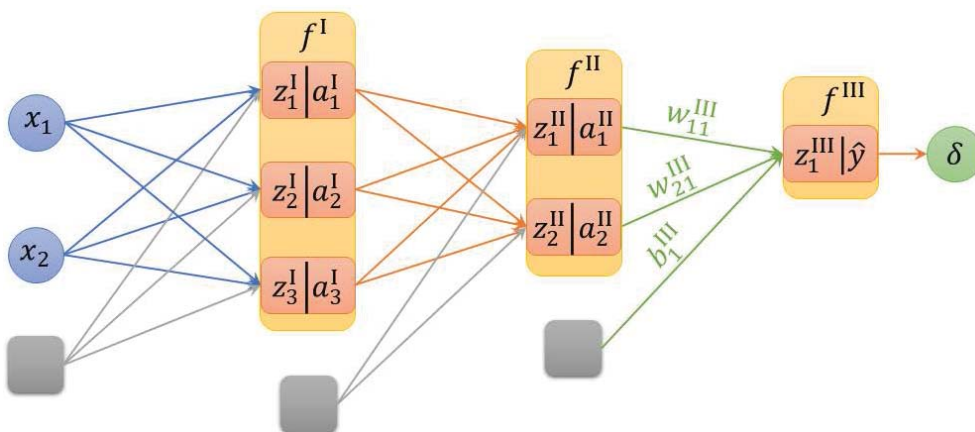


Рисунок 6.14 – Розраховані помилки на кожному нейроні

Останній етап – коригування вагових коефіцієнтів та зміщень відповідно до простих співвідношень, як показано на рисунку 6.15, із використанням загальної помилки роботи мережі.



$$w_{k1}^{III} = w_{k1}^{III} - \eta \delta \frac{df^{III}}{dz_1^{III}} a_k^{II}, \quad b_1^{III} = b_1^{III} - \eta \delta \frac{df^{III}}{dz_1^{III}}$$

Рисунок 6.15 – Коригування вагових коефіцієнтів і зміщень вихідного шару

Формули для коригування вагових коефіцієнтів і зміщень для всіх інших (прихованих) шарів є топологічно однаковими та враховують помилку на конкретному нейроні кожного шару. Схему розрахунків нових значень вагових коефіцієнтів w та зміщень b

для нейронів другого прихованого шару (передостаннього в поданій мережі) наведено на рисунку 6.16.

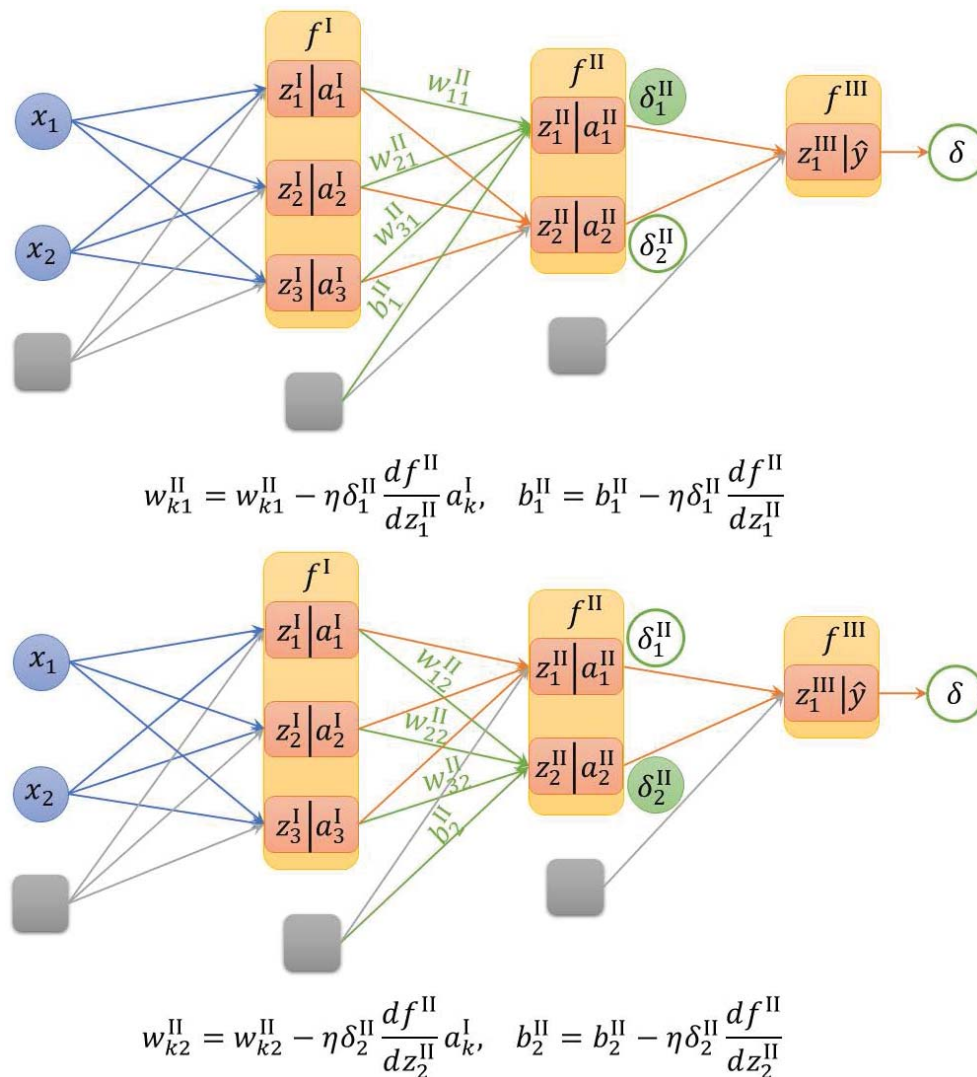
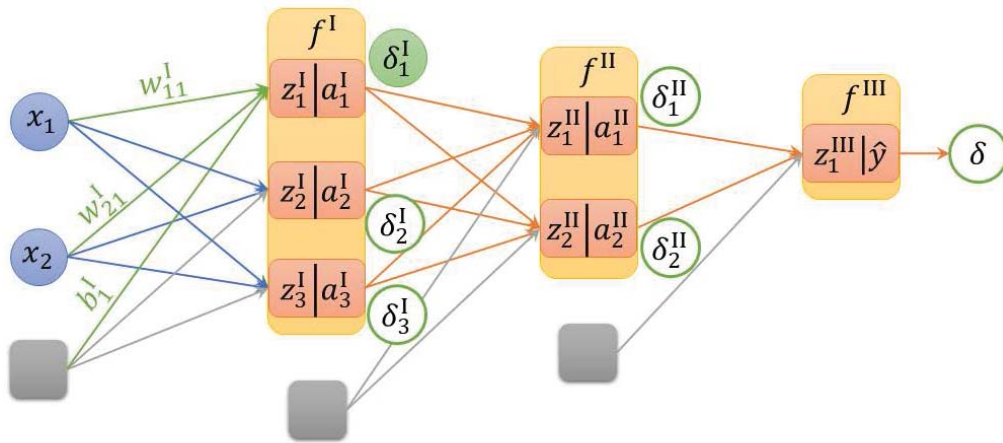
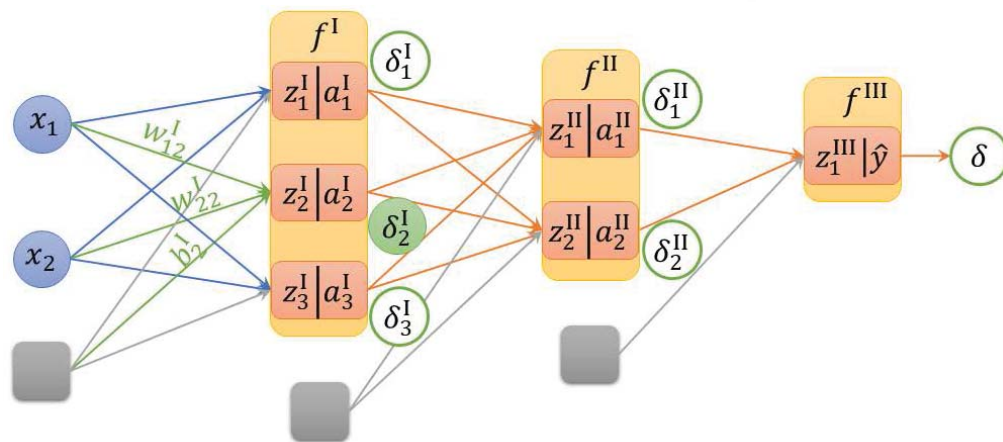


Рисунок 6.16 – Коригування вагових коефіцієнтів і зміщень другого прихованого шару

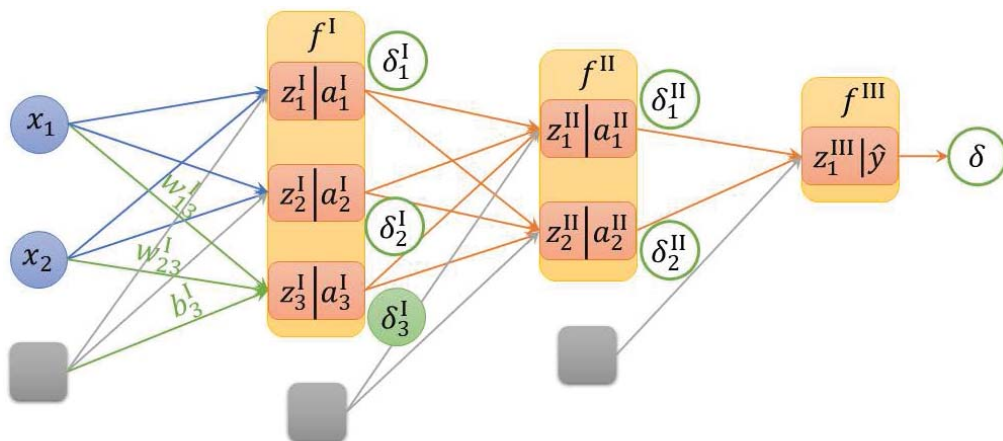
Зрештою, використовуючи помилки на нейронах першого прихованого шару проводимо коригування вагових коефіцієнтів та зміщень цього шару за аналогічними формулами, як це показано на рисунку 6.17. Відмінністю в позначеннях у формулах для розрахунку вагових коефіцієнтів для другого та першого прихованих шарів є те, що в другому прихованому шарі на вхід подаються результати активаційної функції першого прихованого шару a_k^I , тоді як на вхід до першого прихованого шару подаються безпосередньо вхідні сигнали x_k мережі.



$$w_{k1}^I = w_{k1}^I - \eta \delta_1^I \frac{df^I}{dz_1^I} x_k, \quad b_1^I = b_1^I - \eta \delta_1^I \frac{df^I}{dz_1^I}$$



$$w_{k2}^I = w_{k2}^I - \eta \delta_2^I \frac{df^I}{dz_2^I} x_k, \quad b_2^I = b_2^I - \eta \delta_2^I \frac{df^I}{dz_2^I}$$



$$w_{k3}^I = w_{k3}^I - \eta \delta_3^I \frac{df^I}{dz_3^I} x_k, \quad b_3^I = b_3^I - \eta \delta_3^I \frac{df^I}{dz_3^I}$$

Рисунок 6.17 – Коригування вагових коефіцієнтів і зміщень першого прихованого шару

Тема 7. Модифікації методу зворотного поширення. Способи навчання. Збіжність алгоритму навчання. Перенавчання

7.1. Модифікації алгоритму зворотного поширення помилки

Під час навчання нейронної мережі методом зворотного поширення помилки може виявитися низка проблем, притаманних градієнтним методам оптимізації.

Проблема 1. Збіжність алгоритму. Ця проблема пов'язана з вибором розміру кроку (коефіцієнта швидкості навчання), що є найважливішим параметром, що впливає швидкість навчання мережі. Однією з важливих умов збіжності алгоритму є вибір оптимального кроку, що визначає темп навчання. Одним зі способів підвищення швидкості є використання адаптивних алгоритмів вибору кроку в процесі навчання.

Проблема 2. Параліч мережі. У процесі навчання мережі значення вагових коефіцієнтів можуть бути дуже великими. Наслідком цього буде зміщення робочих точок на сигмоїдах в область насичення, де значення вихідної функції нейронів дуже великі, а значення похідної цієї функції дуже малі. Уникнути цього можна зменшенням коефіцієнта швидкості навчання, але збільшується час навчання.

Проблема 3. Локальні пастки. Метод використовує градієнтний спуск поверхнею помилки, безперервно підлаштовуючи ваги до мінімуму помилки. Поверхня помилки складної мережі зазвичай сильно порізана та складається з пагорбів, долин, складок та ярів у просторі високої розмірності. Мережа може потрапити в локальний мінімум (неглибоку долину), тоді як поруч є глибший мінімум. У точці локального мінімуму всі напрями (також, як і в

глобальному мінімумі) ведуть угору, і мережа не здатна з нього самостійно вибратися.

У таких випадках зазвичай рятують стохастичні методи. Мережу «вибивають» із пастки локального мінімуму шляхом присвоєння параметрам мережі випадкових значень із заданого діапазону, після чого продовжується звичайна процедура градієнтного спуску.

Алгоритм зворотного поширення помилки діє ітеративно. На кожному кроці на вхід мережі по чергово подаються всі навчальні приклади, вихідні значення мережі порівнюють із бажаними значеннями та обчислюють похибку. Значення похибки, а також градієнта поверхні станів використовують для корекції вагових коефіцієнтів, і дії повторюються. Процес навчання припиняють або якщо пройдено певну кількість епох, або якщо похибка досягає певного рівня малості, або якщо похибка перестає зменшуватися.

7.1.1. Інерційний алгоритм

Відповідно до методу градієнтного спуску загалом вагові коефіцієнти оновлюються за таким правилом

$$w(t+1) = w(t) - \eta \frac{\partial E}{\partial w(t)}, \quad (7.1)$$

де t позначає епоху навчання. Тоді величину зміни значень вагових коефіцієнтів $\Delta w(t) = w(t+1) - w(t)$ дають виразом

$$\Delta w(t) = -\eta \frac{\partial E}{\partial w}. \quad (7.2)$$

Інерційний алгоритм ґрунтується на додаванні «інерційності» в алгоритм зворотного поширення помилки шляхом уведення моменту інерційності $\mu \in (0, 1)$, коли вплив градієнта на зміну вагових коефіцієнтів накопичується з часом

$$\Delta w(t) = -\eta \frac{\partial E}{\partial w} + \mu \Delta w(t-1). \quad (7.3)$$

Якісно вплив моменту на процес навчання можна пояснити так: припустимо, що градієнт $\frac{\partial E}{\partial w}$ змінюється плавно, тому впродовж деякого часу його зміною можна знехтувати (ми знаходимося далеко

від мінімуму поверхні помилки). Тоді зміну вагових коефіцієнтів можна записати у вигляді

$$\Delta w(t) = -\eta \frac{\partial E}{\partial w} (1 + \mu + \mu^2 + \mu^3 + \dots) \simeq -\frac{\eta}{1 - \mu} \frac{\partial E}{\partial w}. \quad (7.4)$$

У цьому разі ефективний темп навчання збільшується, причому істотно, якщо момент $\mu \rightarrow 1$. Навпаки, поблизу мінімуму поверхні помилки, коли напрям градієнта постійно змінює знак через описані вище осциляції, зміна значень вагових коефіцієнтів $\Delta w(t)$ визначають рівнянням

$$\Delta w(t) = -\eta \frac{\partial E}{\partial w} (1 - \mu + \mu^2 - \mu^3 + \dots) \simeq -\frac{\eta}{1 + \mu} \frac{\partial E}{\partial w}. \quad (7.5)$$

Отже, ефективний темп навчання сповільнюється до значення, близького до η .

Отже, введення інерції в алгоритм навчання дозволяє адаптивно змінювати швидкість навчання на пологих схилах поверхні цільової функції якості – збільшувати довжину кроку навчання, а в крутих місцях – скорочувати їх.

Додаткова перевага від введення моменту – це здатність, що з'являється у алгоритму для подолання дрібних локальних мінімумів. Коефіцієнт згладжує різкі стрибки переміщення поверхнею цільової функції, доповнюючись значенням зміни вагових коефіцієнтів на попередній ітерації. У результаті, «розігнавшись», нейромережа може за інерцією долати невеликі локальні мінімуми поверхні помилки, застрягаючи лише в глибоких значущих мінімумах.

До недоліків цього методу можна віднести введення ще одного глобального налаштовуваного параметра, тоді як під час навчання нейромереж необхідно прагнути до відсутності таких нав'язуваних алгоритму зовні параметрів.

7.1.2. Зменшення вагових коефіцієнтів

Іноді деякі вагові коефіцієнти набувають занадто великих значень і вже самі визначають поведінку мережі. Зазначена модифікація дозволяє дещо уникнути цього

$$\Delta w(t) = -\eta \frac{\partial E}{\partial w} - \gamma \Delta w(t - 1), \quad (7.6)$$

де γ лежить у діапазоні $[0; 1]$. Унаслідок цього середнє значення вагових коефіцієнтів зменшується.

7.1.3. Відкладене навчання

Стандартний алгоритм зворотного поширення помилки є типовим алгоритмом навчання «на льоту». Це означає, що модифікація вагових коефіцієнтів відбувається безпосередньо після пред'явлення кожного об'єкта. У разі відкладеного навчання вагові коефіцієнти оновлюються лише після пред'явлення мережі всієї навчальної вибірки. Тобто всі зразки обробляються мережею, помилки обчислюються, відбувається зворотне поширення, але зміни вагових коефіцієнтів не відбувається, вони накопичуються й модифікація робиться після проходження всієї навчальної вибірки. Хоча цей метод не завжди забезпечує швидку збіжність, він має ще одну перевагу: немає необхідності пред'являти зразки у випадковому порядку.

7.1.4. Метод RProp

Метод RProp (Resilient Propagation – пружне, еластичне поширення) – модифікація алгоритму зворотного поширення помилки, у якій кожний ваговий коефіцієнт має свій адаптивний темп навчання η . Алгоритм RProp використовує знаки частинних похідних для визначення величини корекції кожного вагового коефіцієнта.

Для обчислення значення корекції вагового коефіцієнта використовують таке правило:

$$\Delta w_{ij}^{(t)} = \begin{cases} \eta^+ \delta_{ij}^{(t)}, & \text{якщо } \frac{\partial E^{(t)}}{\partial w_{ij}} \frac{\partial E^{(t-1)}}{\partial w_{ij}} > 0 \\ \eta^- \delta_{ij}^{(t)}, & \text{якщо } \frac{\partial E^{(t)}}{\partial w_{ij}} \frac{\partial E^{(t-1)}}{\partial w_{ij}} < 0 \end{cases}. \quad (7.7)$$

Якщо на поточному кроці частинна похідна за відповідним ваговим коефіцієнтом w_{ij} змінила знак, то із цього випливає, що остання зміна була занадто великою та алгоритм проскочив локальний мінімум. Отже, величину зміни необхідно зменшити в $\eta^- \leq 1$ разів. Якщо знак частинної похідної не змінився, потрібно збільшити величину корекції на $\eta^+ > 1$ задля більш швидкої збіжності.

Потім підлаштовуються вагові коефіцієнти

$$\Delta w_{ij}^{(t+1)} = \Delta w_{ij}^{(t)} + \Delta_{ij}^{(t)}. \quad (7.8)$$

Переваги методу RProp полягають у прискоренні збіжності в 4–5 разів порівняно зі стандартним алгоритмом зворотного поширення помилки.

7.2. Способи навчання багат шарових нейронних мереж

Навчання можна проводити трьома способами:

- стохастичним (stochastic);
- пакетним (batch);
- міні-пакетним (mini-batch).

Існує дуже багато статей та досліджень на тему того, який із методів кращий і ніхто не може дати загальної відповіді. Водночас кожен метод має свої плюси та мінуси.

Стохастичний (його ще іноді називають онлайн) спосіб працює за таким принципом: на вхід нейронної мережі подають по одному прикладу. Для кожного прикладу розраховують різницю вагових коефіцієнтів Δw і відразу оновлюють відповідні вагові коефіцієнти.

Пакетний спосіб працює інакше: на вхід нейронної мережі по черзі подають декілька прикладів (пакет). Для кожного прикладу розраховують відповідне значення Δw . Далі, для всіх прикладів проводять підсумовування усіх вагових коефіцієнтів на поточній ітерації і лише потім оновлюють усі вагові коефіцієнти, використовуючи цю суму. Один із найважливіших плюсів такого підходу – це значна економія часу на обчислення. Недоліком є те, що при такому способі точність роботи мережі може бути нижчою.

Міні-пакетний спосіб є золотою серединою. У ньому поєднані плюси обох методів. Принцип цього способу полягає в такому: у вільному порядку розподіляють об'єкти по групах і змінюються їх вагові коефіцієнти на суму Δw всіх вагових коефіцієнтів у тій чи іншій групі.

7.3. Гіперпараметри та збіжність мережі

Гіперпараметри – це значення, що потрібно підбирати вручну й часто методом спроб і помилок. Серед таких параметрів можна виділити:

- параметр збурення (момент);
- коефіцієнт швидкості навчання;
- кількість прихованих шарів;
- кількість нейронів у кожному шарі;
- наявність або відсутність нейронів зміщення.

В різних типах нейронних мереж наявні додаткові гіперпараметри, але про них тут ми не говоритимемо. Підбір правильних гіперпараметрів дуже важливий і безпосередньо впливатиме на збіжність нейронної мережі. Зрозуміти, чи необхідно використовувати нейрони зміщення чи ні, досить просто. Кількість прихованих шарів і нейронів у них можна обчислити перебором, ґрунтуючись на одному простому правилі – чим більше нейронів, тим точніше результат і тим експоненційно більший час, що буде необхідний для навчання нейронної мережі. Однак потрібно пам'ятати, що не варто робити нейронну мережу з 1 000 нейронів для вирішення простих завдань. А ось із вибором значення збурення (моменту) та швидкості навчання все трохи складніше. Ці гіперпараметри будуть змінюватися, залежно від поставленого завдання та архітектури нейронної мережі. Наприклад, для вирішення «XOR» швидкість навчання може бути в межах 0.3 – –0.7, але в нейронній мережі що аналізує та передбачає ціну акцій, швидкість навчання вище 0.00001 призводить до поганої збіжності. Тому настроювання гіперпараметрів зазвичай проводять експериментальним способом для конкретної архітектури нейронної мережі та специфіки задачі.

Збіжність нейронної мережі говорить про те, чи правильно була обрана архітектура нейронної мережі чи правильно були підбрані гіперпараметри відповідно до поставленого завдання. У процесі навчання нейронної мережі розраховують помилку, яку робить мережа, на кожній епосі навчання. Аналізуючи динаміку зміни помилки мережі можна судити про збіжність нейронної мережі.

У разі коли зі збільшенням епохи навчання помилка поводитьсь немонотонно: то збільшується, то зменшується (див. рис. 7.1а),

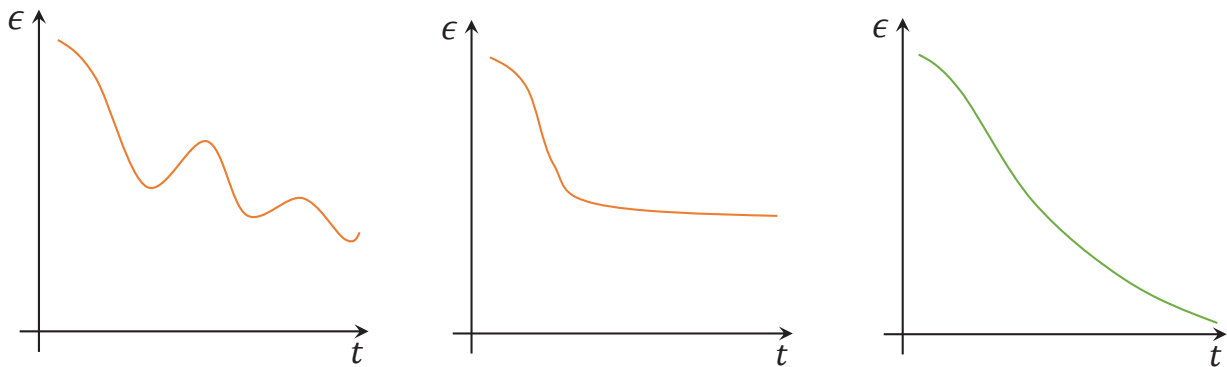


Рисунок 7.1 – Збіжність нейронної мережі

або досягає певного досить високого стаціонарного значення (рис. 7.1б), то нейронна мережа не збігається. У 99 % випадків це вирішують зміною гіперпараметрів; 1 %, що залишився, означатиме, що архітектура нейронної мережі для поставленої задачі обрана неправильно. Якщо ж зі збільшенням номера епохи навчання помилка монотонно спадає (див. рис. 7.1в), то нейронна мережа збігається, а всі гіперпараметри та архітектура мережі підібрані правильно.

7.4. Перенавчання нейронної мережі

Також буває, що на збіжність нейронної мережі впливає її перенавчання. Перенавчання, як впливає з назви, це стан нейронної мережі, коли вона перенасичена даними. Ця проблема виникає, якщо занадто довго навчати мережу на тих самих даних. Іншими словами, мережа почне не вчитися на даних, а запам'ятовувати й зубрити їх. Відповідно, коли до цієї перенавченої мережі на вхід подаватимуться нові дані, то в одержаних даних може з'явитися шум, який впливатиме на точність результату.

Наприклад, якщо нейронній мережі пред'являти різні фотографії яблук (лише червоні) й говорити, що це яблуко, то коли на вхід нейронної мережі подадуть жовте або зелене яблуко, вона не зможе визначити, що це яблуко, оскільки вона запам'ятала, що всі яблука мають бути червоними. І навпаки, коли нейронній мережі подати на вхід зображення чогось червоного й за формою, що збігається з яблуком, наприклад персик, вона скаже, що це яблуко.

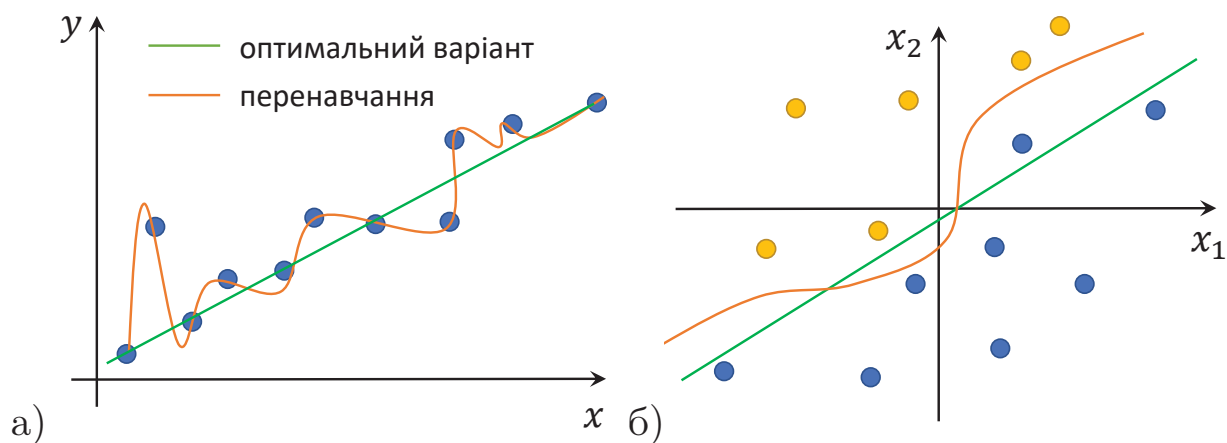


Рисунок 7.2 – Перенавчання нейронної мережі для: а) задачі регресії; б) задачі класифікації

Це і є шум. Схематично приклад перенавчання мережі наведено на рисунку 7.2 для випадку задачі регресії (рис. 7.2а) та задачі класифікації (рис. 7.2б).

Видно, що графік функції (помаранчева крива на рисунку) сильно коливається від точки до точки, які є вихідними даними (результатом) нашої нейронної мережі. В ідеалі цей графік повинен бути менш хвилястий і прямий (зелена пряма на рисунку). Щоб уникнути перенавчання, не потрібно довго тренувати нейронну мережу на одних і тих самих або дуже схожих даних. Також, перенавчання може бути викликане великою кількістю параметрів, які нейронна мережа одержує на вхід, або надто складною архітектурою мережі. Отже, коли виникають помилки (шум) у вихідних даних після етапу навчання, необхідно використовувати один із методів регуляризації, але здебільшого це не знадобиться.

Ефект перенавчання можна контролювати в процесі навчання. Для цього з навчальної вибірки вибирають дані для вибірки валідації. На вхід нейронної мережі подають усі об'єкти з навчальної вибірки й на кожній епосі навчання розраховують критерій якості роботи мережі (загальну помилку) на кожній множині: навчальній та валідаційній із навчальної вибірки. У результаті одержують залежності загальної помилки для кожної множини, як показано на рисунку 7.3.

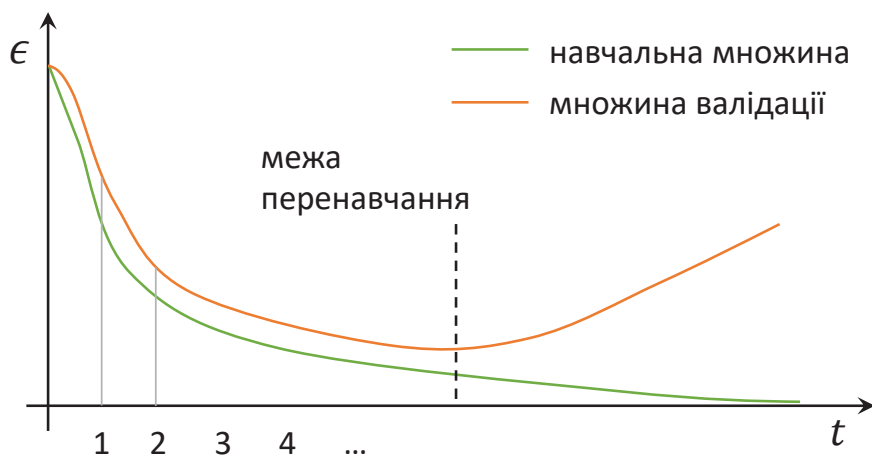


Рисунок 7.3 – Контроль перенавчання на навчанні

Якщо після певної ітерації (епохи навчання) графіки залежності помилки від епохи починають розбігатися, то можна з упевненістю стверджувати, що нейронна мережа починає перенавчатися й процес навчання треба зупинити. У такому разі найкращі значення вагових коефіцієнтів відповідають межі перенавчання. У цьому підході множина валідації, що є частиною загальної навчальної вибірки, також бере участь у навчанні нейронної мережі та налаштуванні вагових коефіцієнтів. Для об'єктивної перевірки якості роботи нейронної мережі використовують тестову вибірку, приклади (об'єкти) якої не брали участі в процесі навчання.

Отже, для контролювання процесу навчання на рахунок перенавчання весь набір даних потрібно розбивати на три множини: навчальну, валідаційну та тестову.

Тема 8. Приклади задач із багат шаровими нейронними мережами прямого поширення

8.1. Апроксимація заданої функції

8.1.1. Постановка задачі

Нехай існують протабульовані значення вимірювання (залежна змінна y) від незалежної змінної x : $y = f(x)$.

Завдання: навчити нейронну мережу прямого поширення апроксимувати табульовані дані.

Для розв'язання цієї задачі спочатку треба згенерувати навчальну вибірку, а саме N значень незалежної змінної x у фіксованому інтервалі $x \in [a, b]$. Далі треба визначити значення цільового вектора – обрати тип нелінійної немонотонної функції $f(x)$, яку будемо апроксимувати.

Для прикладу розглянемо функцію $f(x) = \sin(x)$ на інтервалі $x_{train} \in [0, 10)$ з кроком 0.1, що дає $N = 100$ об'єктів (x_{train}, y_{train}) .

За тестову вибірку можна взяти значення незалежної змінної $x_{test} = x_{train} + 0.05$ та визначити значення цільового вектора для тестової вибірки $y_{test} = f(x_{test})$.

8.1.2. Кроки реалізації

Наступним кроком є ініціалізація нейронної мережі. Для поставленої задачі апроксимації функції на вхід двошарової нейронної мережі подається один сигнал – кожне значення x_{train} . На виході нейронної мережі буде один нейрон, який буде видавати значення апроксимації для значення x_{train} . Кількість n нейронів у прихованому шарі буде визначати розмірність матриць вагових коефіцієнтів w_{1n}^I та w_{n1}^{II} , а також та довжину вектору зміщень b_n^I . На вихідному шарі буде одне значення зміщення b_1^{II} . Для цієї задачі для нейронів прихованого шару можна обрати сигмоїдну функцію активації. Для

нейрона вихідного шару можна лінійну функцію активації. Структура двошарової нейронної мережі матиме вигляд, як показано на рисунку 8.1.

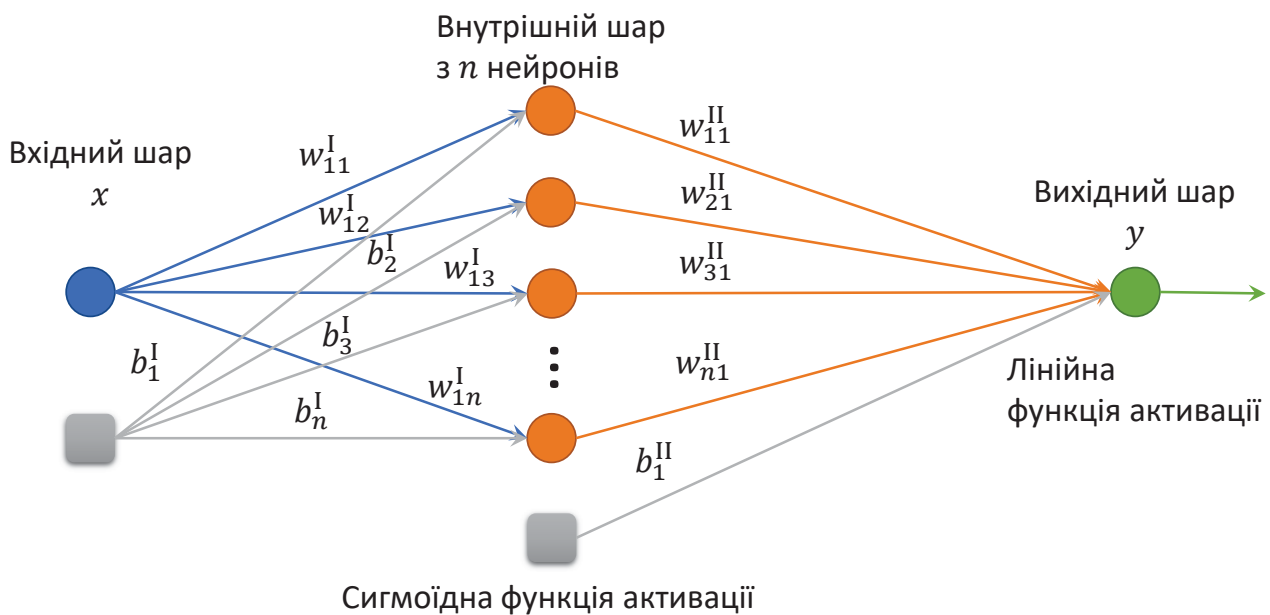


Рисунок 8.1 – Структура двошарової нейронної мережі для задачі апроксимації функції

Далі необхідно зафіксувати кількість епох навчання, крок навчання та допустиму похибку ε . Похибку навчання можна визначити як задане мінімальне значення середньоквадратичної помилки для всіх значень навчальної вибірки. Водночас умовою припинення навчання може бути:

- досягнення зафіксованої кількості епох навчання;
- або умови, коли загальна помилка апроксимації для всіх значень навчальної вибірки стане меншою за фіксоване значення ε .

Тепер, використовуючи метод зворотного поширення помилки навчаємо нейронну мережу апроксимувати задану функцію $f(x)$ використовуючи дані табуляції (x_i, y_i) з (x_{train}, y_{train}) для $i = 1, \dots, N$. Для кожної епохи навчання розраховувати загальну (середню) помилку апроксимації.

Типові результати залежності середньоквадратичної помилки від номера епохи навчання на навчальній вибірці в процесі навчання нейронної мережі подано на рисунку 8.2.

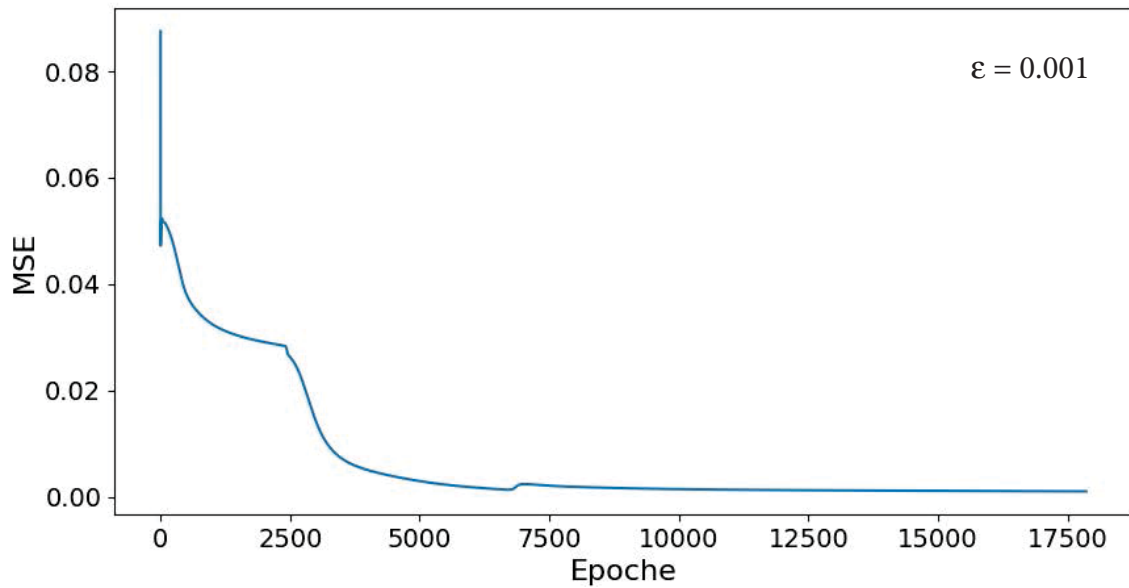


Рисунок 8.2 – Залежності середньоквадратичної помилки від номера епохи навчання

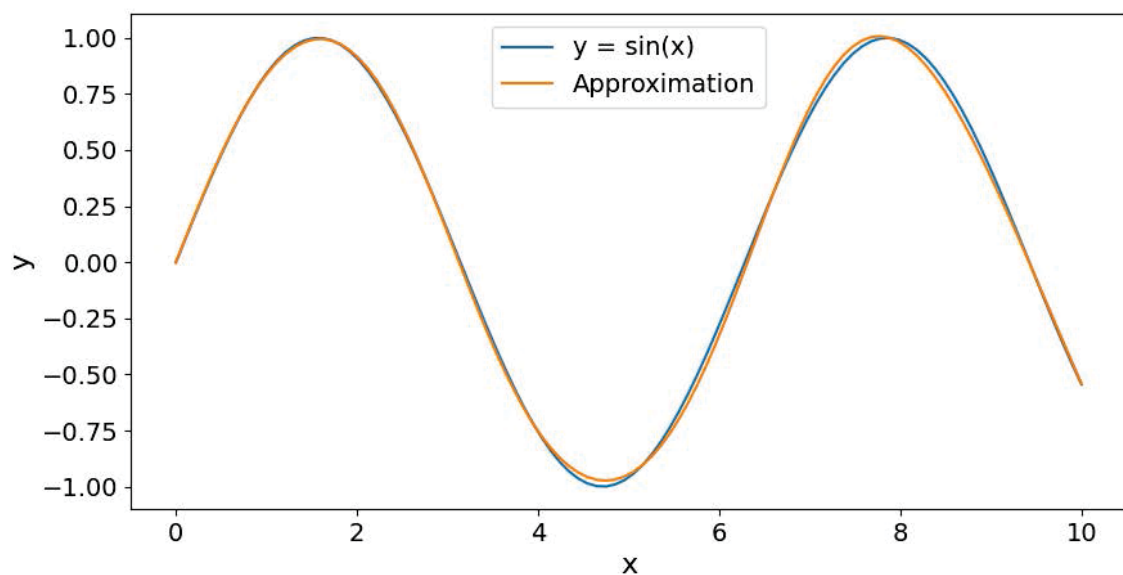


Рисунок 8.3 – Залежність $f(x)$ та результат роботи мережі

Результатом навчання нейронної мережі є матриці вагових коефіцієнтів w_{1n}^I та w_{n1}^{II} , а також вектор зміщень b_n^I та значення b_1^{II} . Після завершення навчання їх можна використати для апроксимації заданої функції, а потім проілюструвати дані навчальної вибірки та результат роботи нейронної мережі.

Бачимо, що в межах зафіксованого значення середньоквадратичної помилки ϵ маємо досить гарну відповідність результату роботи мережі та даних навчальної вибірки. Наприкінці, проводимо тестування результату навчання на тестовій вибірці (x_{test}, y_{test}) , розраховуючи середньоквадратичну помилку на цій вибірці, яка для цього прикладу має значення $MSE_{test} \simeq 0.0025$.

8.1.3. Програмна реалізація навчання нейронної мережі апроксимувати функцію

Ініціалізація параметрів нейронної мережі

```
# Завантаження бібліотек
import numpy as np
import math
from matplotlib import pyplot as plt

# Визначення структури мережі
NumOfFeatures = 1
NumOfExamples = 100
NumOfLayers = 3
NumOfNeurs = 10
NumOfExits = 1
neurons = np.array([NumOfFeatures, NumOfNeurs, NumOfExits])

# Ініціалізація виходів нейронів
a1 = np.zeros((NumOfNeurs, 1))
y1 = np.zeros((NumOfNeurs, 1))
a2 = np.zeros((NumOfExits, 1))
y2 = np.zeros((NumOfExits, 1))
```

Генерація навчальної та тестової вибірок та ініціалізація вагових коефіцієнтів і зміщень

```
def generator():
    # Навчальна вибірка
    x = np.linspace(0, 10, NumOfExamples)
    y = np.sin(x)
    data=np.ones((NumOfExamples, NumOfFeatures))
    data[:,0] = x[:]
    data = np.insert(data, NumOfFeatures, values=y, axis=1)

    # Тестова вибірка - точки між даними навчальної вибірки
    _x = x + 0.05
    _y = np.sin(_x)

    # Ініціалізація вагових коефіцієнтів та зміщень
    # прихованого та вихідного шарів
    w1 = 1 - 2*np.random.rand(neurons[0], neurons[1])
    b1 = 1 - 2*np.random.rand(neurons[1])
    w2 = 1 - 2*np.random.rand(neurons[1], neurons[2])
    b2 = 1 - 2*np.random.rand(neurons[2])

    return data, _x, _y, w1, w2, b1, b2
```

Активаційні функції та їх похідні

```
def sigm(z):
    return(1/(1 + np.exp(-z)))

def dif_sigm(z):
    return((1 - sigm(z))*sigm(z))

def lin(z,C):
    return(z*C)

def dif_lin(z,C):
    return(C)
```

Імпорт даних та навчання нейронної мережі

```
# alpha - крок навчання
# eps - допустима помилка апроксимації
num_epoch, alpha, eps, C = 20000, 0.01, 0.001, 1
data_train, x_test, y_test, w1, w2, b1, b2 = generator()

err = []
time = []
for _i in range(num_epoch):
    er = 0
    for x in data_train:
        # Прямий хід
        a1 = np.dot(x[:NumOfFeatures], w1) + b1
        y1 = sigm(a1)
        a2 = np.dot(y1, w2) + b2
        y2 = lin(a2, C)
        # Розрахунок помилки апроксимації
        delta = (y2 - x[NumOfFeatures])
        er += delta*delta
        # Розрахунок градієнтів
        dEdb_L2 = delta*dif_lin(y2, C)
        dEdw_L2 = np.dot(y1.reshape(-1, 1), dEdb_L2.reshape(-1, 1).transpose())
        dEdb_L1 = np.dot(dEdb_L2, w2.transpose()*dif_sigm(a1))
        dEdw_L1 = np.dot(x[:NumOfFeatures].reshape(-1, 1), dEdb_L1.reshape(-1, 1).transpose())
        # Оновлення вагових коефіцієнтів та зміщень методом градієнтного спуску
        w2 = w2 - alpha*dEdw_L2
        b2 = b2 - alpha*dEdb_L2
        w1 = w1 - alpha*dEdw_L1
        b1 = b1 - alpha*dEdb_L1
    # Умова припинення навчання
    if(math.sqrt(er)/NumOfExamples < eps): break
    # Розрахунок помили
    err.append(math.sqrt(er)/NumOfExamples)
    time.append(_i + 1)
```

Аналіз результату роботи мережі на навчальних даних

```
ans = []
for x in data_train:
    a1 = np.dot(x[:NumOfFeatures], w1) + b1
    y1 = sigm(a1)
    a2 = np.dot(y1, w2) + b2
    y2 = lin(a2, C)
    ans.append(y2)
```

Тестування роботи мережі на тестовій вибірці

```
er = 0
for i in range(len(x_test)):
    a1 = np.dot(x_test[i], w1) + b1
    y1 = sigm(a1)
    a2 = np.dot(y1, w2) + b2
    y2 = lin(a2, C)
    test.append(y2)
    delta = (y2 - x[NumOfFeatures])
    er += delta*delta
er = math.sqrt(er)/len(x_test)
print('Error (MSE) in test = ', er)
```


8.2. Навчання агента грати в «пінг-понг»

8.2.1. Постановка задачі

Нейронну мережу прямого поширення можна використати для розв'язання завдань подібних до задач навчання з підкріпленням. Навчання з підкріпленням (Reinforcement Learning) – це метод машинного навчання, у якому наша система (агент) навчається методом спроб і помилок. Ідея полягає в тому, що агент взаємодіє із середовищем, паралельно навчаючись, і одержує винагороду за виконання дій. У навчанні з підкріпленням використовують спосіб позитивної нагороди за правильну дію, щоб спонукати агента, та негативну – за неправильну. Це програмує нашого агента на пошук довгострокової та максимальної загальної винагороди для досягнення оптимального рішення. Ці довгострокові цілі не дають агенту можливості зупинитися на досягнутому. Згодом система вчиться уникати негативних дій і робить лише позитивні.

На відміну від підходу навчання з підкріпленням, використання нейронної мережі для навчання агента робити правильні «кроки» не використовуються нагороди й покарання. У разі, коли агент робить помилку запускається цикл донавчання нейронної мережі – оптимізація вагових коефіцієнтів мережі, що дозволить йому уникати помилок.

Для приладу використання нейронної мережі прямого поширення в задачах «агент – середовище» розглянемо спрощений варіант гри «пінг-понг». Наша задача навчити агента відбивати ракеткою кульку, що наближається до краю столу. Середовищем тут є обмежений простір прямокутної форми та рухомий об'єкт – кулька; агент це рухома ракетка робота. Спрощену схему «агент–середовище» подано на рисунку 8.4 справа. Відповідно до наведеної схеми агент (синій прямокутник), рухаючись угору або вниз, повинен відбити кульку (помаранчевий круг), яка рухається в напівзамкненому просторі розміром $H \times W$. Водночас потрібно врахувати, що кулька, рухаючись справа наліво, може вдаритися об обмеження простору (зверху або знизу) та відбиватися від них (помаранчеві кола). Завдання агента – відбити кульку серединою ракетки. Цього досягають визначенням напрямку руху y_m відповідно до

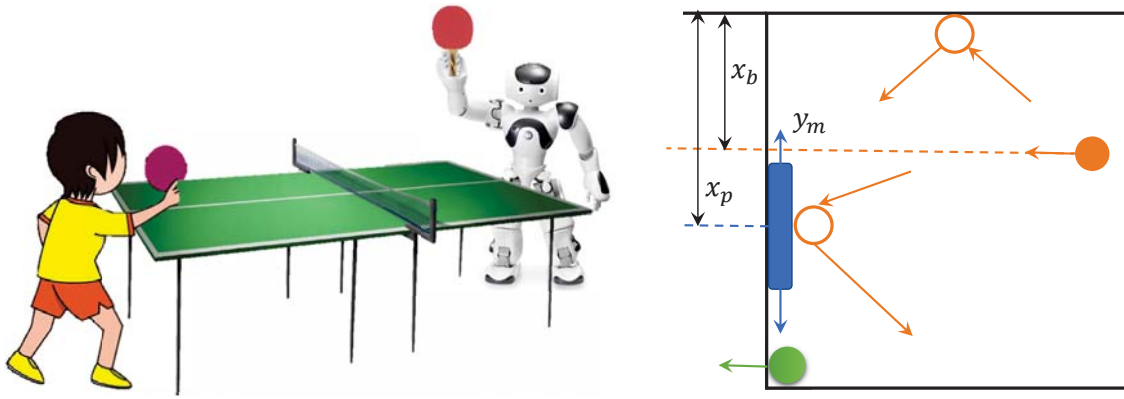


Рисунок 8.4 – Спрощена схема гри в «пінг-понг»

поточних значень положення самого агента x_p та кульки x_b . Помилкою агента вважають ситуацію, коли кулька перетнула ліву межу простору (зелений круг). Після кожної помилки відбувається донавчання нейронної мережі використовуючи метод зворотного поширення помилки: упродовж певної кількості кроків (епох навчання) для одних і тих самих поточних значень положення кульки та агента змушуємо його рухатися в напрямку кульки. Успіхом агента вважають ситуацію, коли він відбив кульку. Отже, для оцінювання ефективності навчання агента можна використовувати залежність кількості успіхів за один ігровий епізод.

8.2.2. Реалізація гри

Спершу встановимо структуру двошарової нейронної мережі. Для цієї задачі на вхід двошарової нейронної мережі подається $M = 2$ сигнали – поточне положення кульки x_b та агента (гравця) x_p . На вихідному шарі буде лише один нейрон $N = 1$, який буде визначати напрямок, куди треба рухатись агенту в поточній конфігурації середовища. Кількість n нейронів у прихованому шарі буде визначати розмірність матриць вагових коефіцієнтів w_{2n}^I та w_{n1}^{II} . Для цієї задачі зміщення b^I та b^{II} для нейронів прихованого шару та вихідного шару можна не розглядати. Для нейронів прихованого шару та нейрона вихідного шару можна обрати сигмоїдну функцію активації. Структура двошарової нейронної мережі матиме вигляд, як показано на рисунку 8.5.

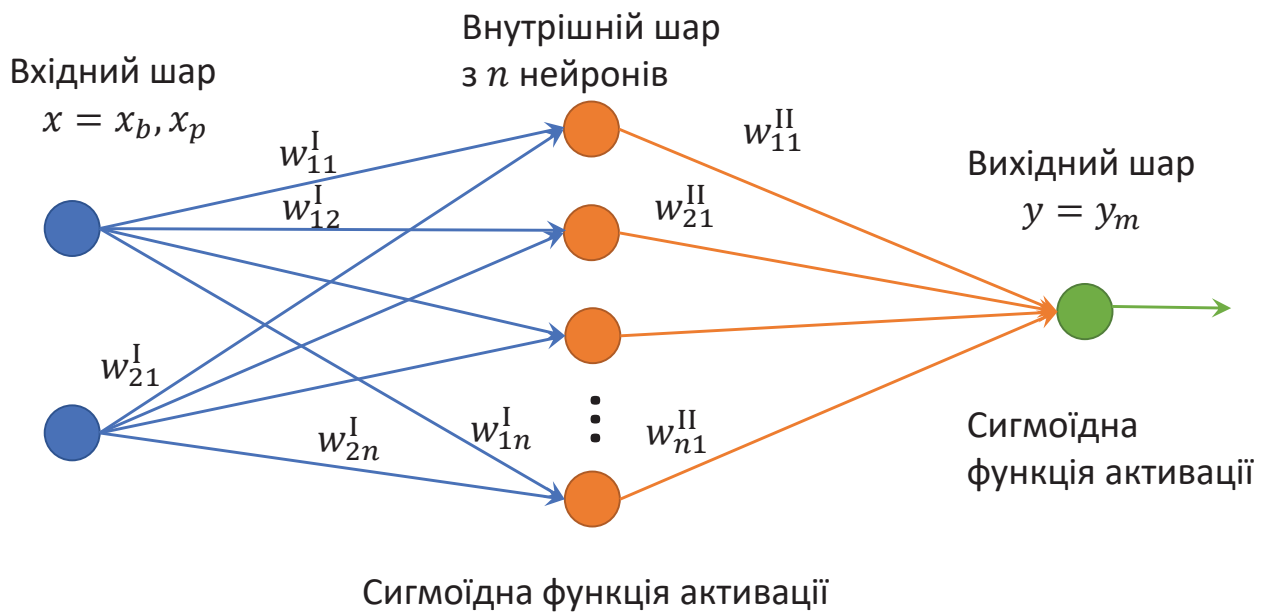


Рисунок 8.5 – Структура двошарової нейронної мережі для навчання агента відбивати кульку

Сигмоїдна функція на вихідному шарі буде визначати ймовірність руху агента

$$\begin{cases} y_m = \text{« } \rightarrow \text{ »}, \text{ якщо результат мережі } \geq 0.5 \\ y_m = \text{« } \leftarrow \text{ »}, \text{ у протилежному випадку.} \end{cases}$$

Далі необхідно зафіксувати кількість епох донавчання мережі за умови помилки агента, крок навчання та умову припинення навчання:

- досягнення зафіксованої кількості епізодів навчання;
- досягнення зафіксованої кількості успіхів агента на одному епізоді.

Результатом навчання нейронної мережі є матриці вагових коефіцієнтів w_{2n}^I та w_{n1}^{II} . Після завершення навчання їх можна зберегти у файл для подальшого їх використання для гри людини з комп'ютером.

Після завершення навчання можна проаналізувати залежність кількості успіхів агента за один ігровий епізод, що показано на рисунку 8.6.

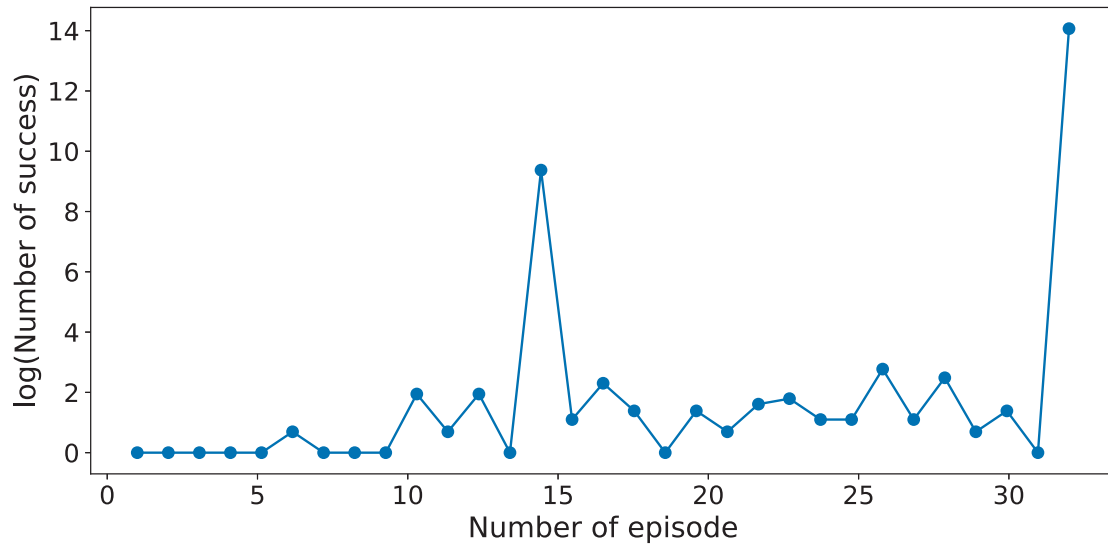
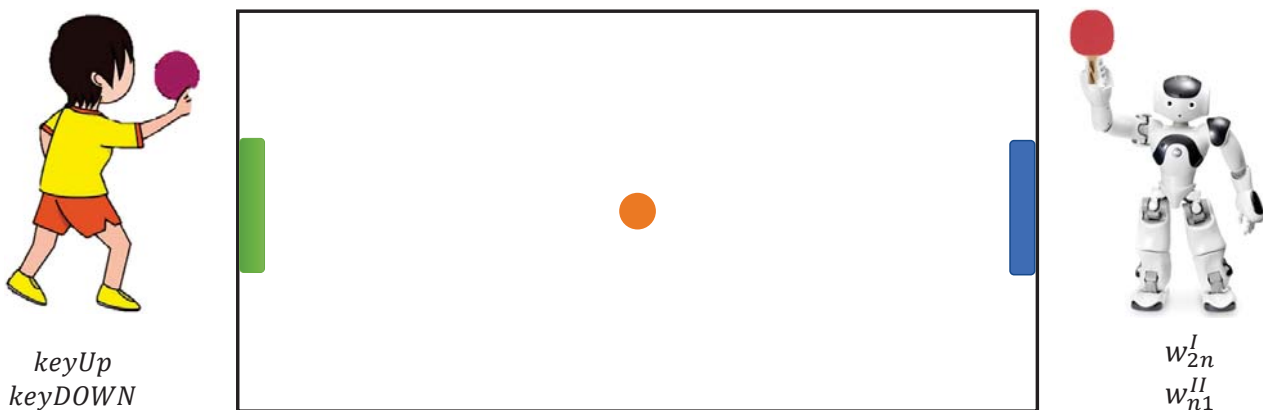


Рисунок 8.6 – Залежність логарифму кількості успіхів агента за один ігровий епізод

Тут для подання результатів використано логарифм, щоб наочно продемонструвати залежність. Видно, що перші 5 епізодів агент не мав жодного успіху. Вже після 30 помилок агент близько $e^{14} > 10^6$ разів підряд безпомилково відбив кульку.

Далі, для агента можна завантажити досвід (матриці вагових коефіцієнтів), створити ще одного гравця (яким керуватиме людина) та спробувати обіграти комп'ютер.



Для візуалізації процесу навчання агента та його гри з людиною можна використати бібліотеку `pygame`.

8.3. Розпізнавання рукописних цифр

8.3.1. Постановка задачі

Нехай існує база даних картинок (зображень) рукописних цифр від 0 до 9, кожна з яких має розмір $t \times t$ пікселів.

Завдання: навчити нейронну мережу прямого поширення розпізнавати рукописні цифри.

Для прикладу розглянемо набір даних `mnist` із бібліотеки `tensorflow` розміром $M = 28 \times 28$ із кількістю цифр у навчальній вибірці $N_{train} = 60\,000$ та в тестовій вибірці $N = 10\,000$. Типові приклади навчальної вибірки подано на рисунку 8.7.

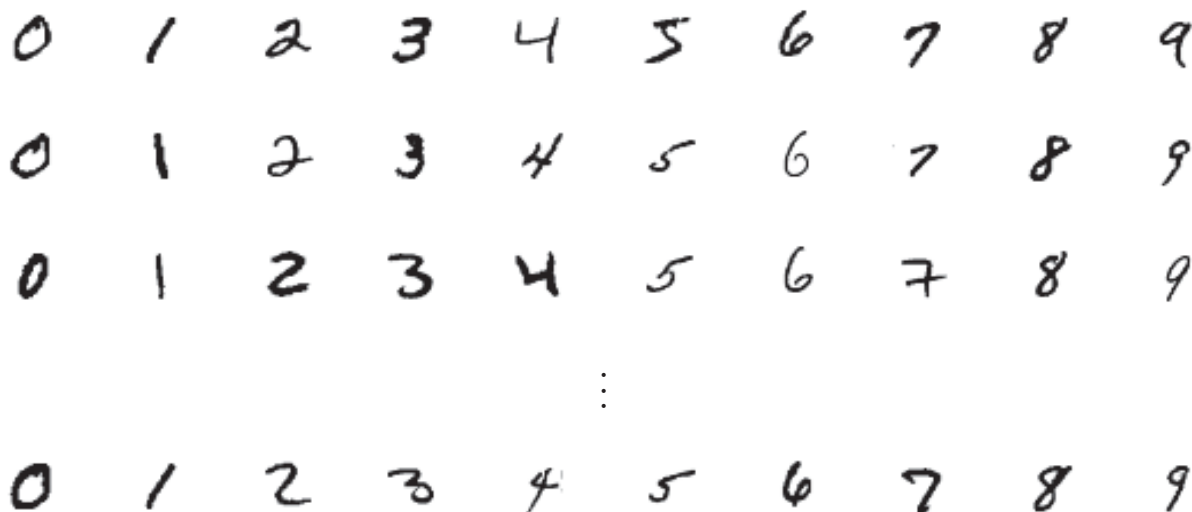


Рисунок 8.7 – Візуалізація даних навчальної вибірки

8.3.2. Етапи навчання та тестування

Спершу проаналізуємо репрезентативність навчальної вибірки, розраховуючи кількість картинок кожної цифри у вибірці у відсотках. Результати подано на рисунку 8.8.

Оскільки цільовий вектор у цій вибірці репрезентовано набором цифр від 0 до 9, то для аналізу вихідних значень нейронної мережі значення цільового вектора потрібно переформатувати в такий спосіб. Для кожного значення цільового вектора треба зробити масив з 10-ти елементів, установивши значення 1 для порядкового

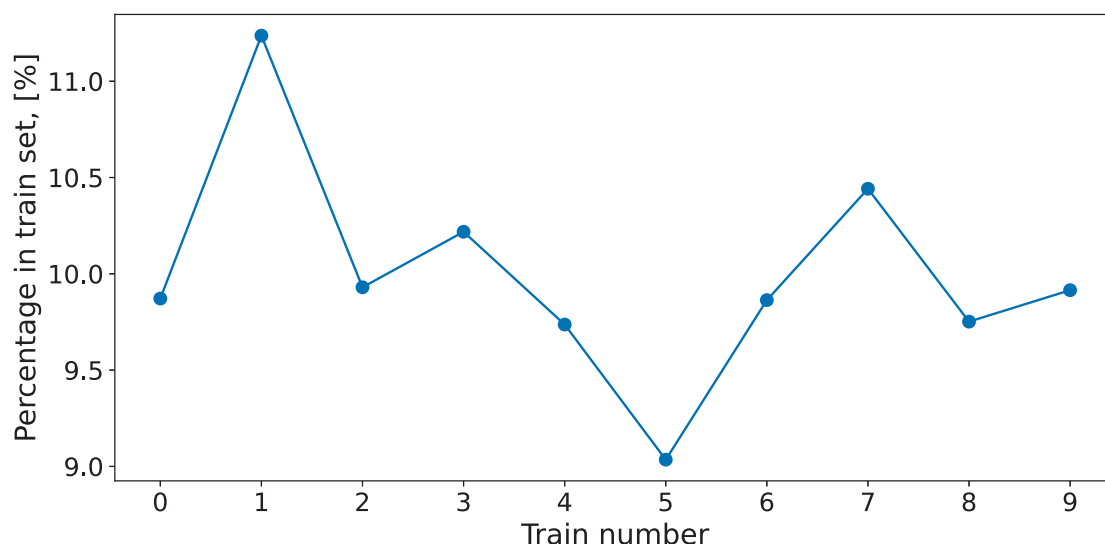


Рисунок 8.8 – Аналіз репрезентативності навчальної вибірки.

номера елемента, який відповідає цифрі, та всі інші нулями. Наприклад,

$$y_i = 5 \quad \Rightarrow \quad y_i = [0, 0, 0, 0, 0, 1, 0, 0, 0, 0].$$

Тепер установимо структуру двошарової нейронної мережі. Для задачі розпізнавання цифр на вхід двошарової нейронної мережі подається $M = 784$ сигналів; на виході буде $N = 10$ нейронів і кожен нейрон буде відповідати кожній цифрі від 0 до 9. Структура двошарової нейронної мережі матиме вигляд, як показано на рисунку 8.9. Кількість n нейронів у прихованому шарі буде визначати розмірність матриць вагових коефіцієнтів w_{Mn}^I та w_{nN}^{II} , а також та довжину вектора зміщень b_n^I ; довжина вектора b_N^{II} визначається кількістю виходів нейронної мережі. Для задачі розпізнавання образів для нейронів прихованого шару можна обрати сигмоїдну функцію активації. Для нейронів вихідного шару можна використати порогову функцію, або сигмоїдну функцію, розраховуючи ймовірність розпізнавання для кожної цифри.

Далі треба зафіксувати кількість епох навчання, крок навчання та допустиму похибку ε – відносну кількість помилок розпізнавання на навчальній вибірці. У такому разі зупиняти процедуру навчання можна коли загальна помилка розпізнавання для всіх зображень стане меншою за встановлене значення ε .

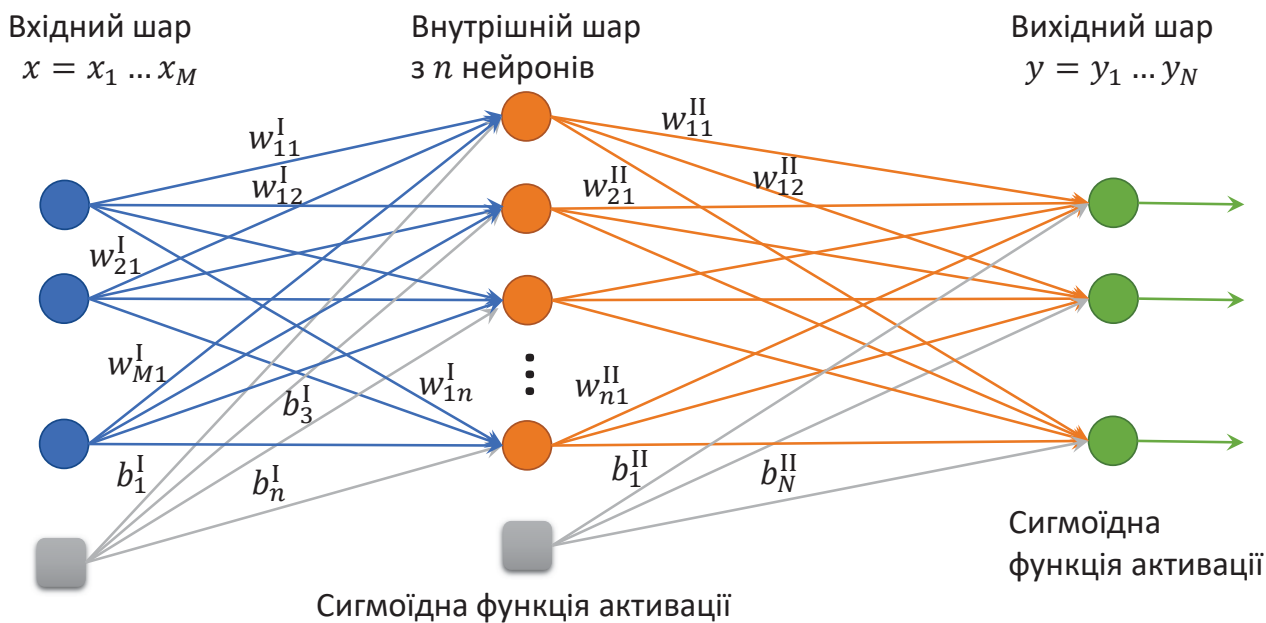


Рисунок 8.9 – Структура двошарової нейронної мережі для розпізнавання рукописних цифр

Тепер, використовуючи метод зворотного поширення помилки, навчаємо нейронну мережу розпізнавати рукописні цифри з навчальної вибірки. Для кожної епохи навчання розраховуємо загальну помилку розпізнавання та одержуємо залежність точності роботи мережі від епохи навчання, як показано на рисунку 8.10.

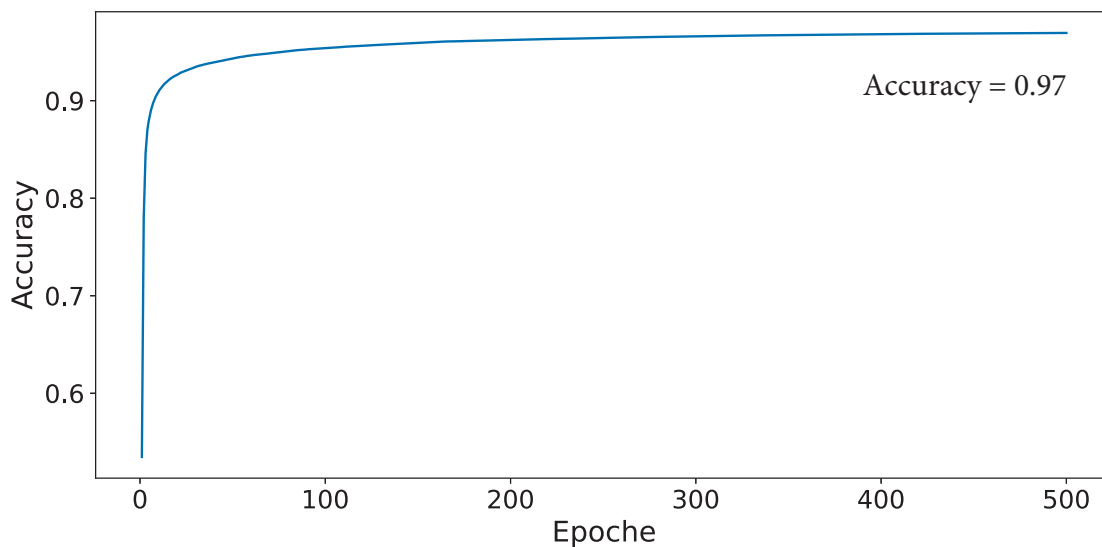


Рисунок 8.10 – Залежність точності мережі від епохи навчання

Результатом навчання нейронної мережі є матриці вагових коефіцієнтів w_{Mn}^I та w_{nN}^{II} , а також вектори зміщень b_n^I та b_N^{II} . Після завершення навчання їх можна зберегти у файл для подальшого використання для розпізнавання цифр тестової вибірки та власних рукописних цифр.

Для того щоб протестувати навчену нейронну мережу виконуємо такі дії:

- завантажуюмо з файлу матриці вагових коефіцієнтів w_{Mn}^I і w_{nN}^{II} та вектори зміщень b_n^I і b_N^{II} ;
- даємо на вхід зображення цифр із тестової вибірки;
- розпізнаємо цифру та порівнюємо відповідь мережі зі значенням цільового вектора;
- розраховуємо загальну помилку мережі на тестовій вибірці;
- аналізуємо частоту помилок на кожній цифрі тестової вибірки, як показано на рисунку 8.11.

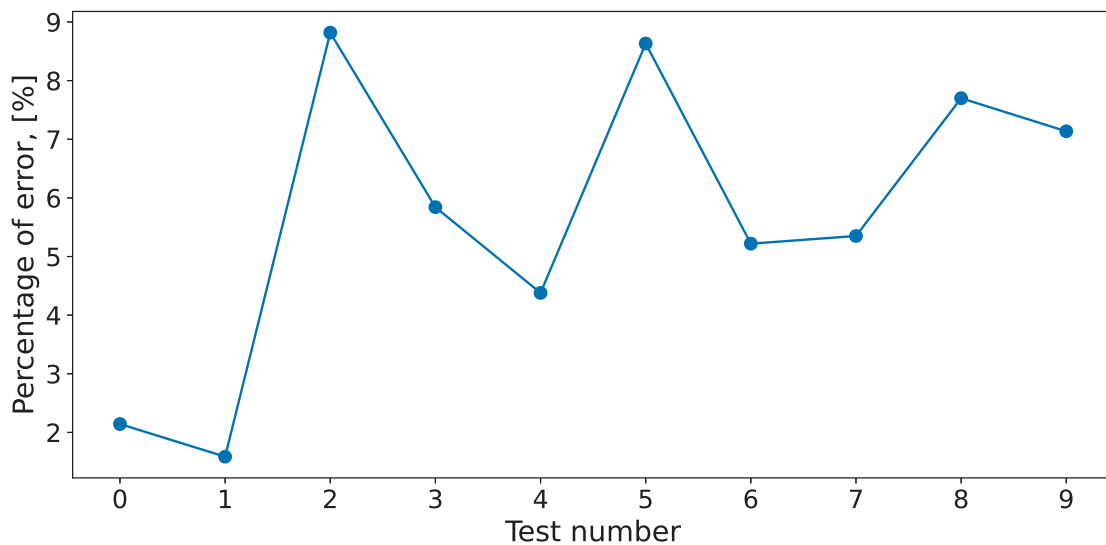


Рисунок 8.11 – Частота помилок на кожній цифрі тестової вибірки.

Тепер можна використати навчену мережу для розпізнавання власних рукописних цифр. Відповідно до типу нейронної мережі цифра повинна вміщатися у квадрат розміром 28×28 пікселів і розміщуватися по центру. Для того, щоб відцентрувати та переформатувати будь-яке зображення рукописної цифри можна використати алгоритм, наведений в кінці розділу.

Результат використання алгоритму масштабування зображення та тестування на навченій нейронній мережі показано на рисунках 8.12 та 8.13 для різних цифр.

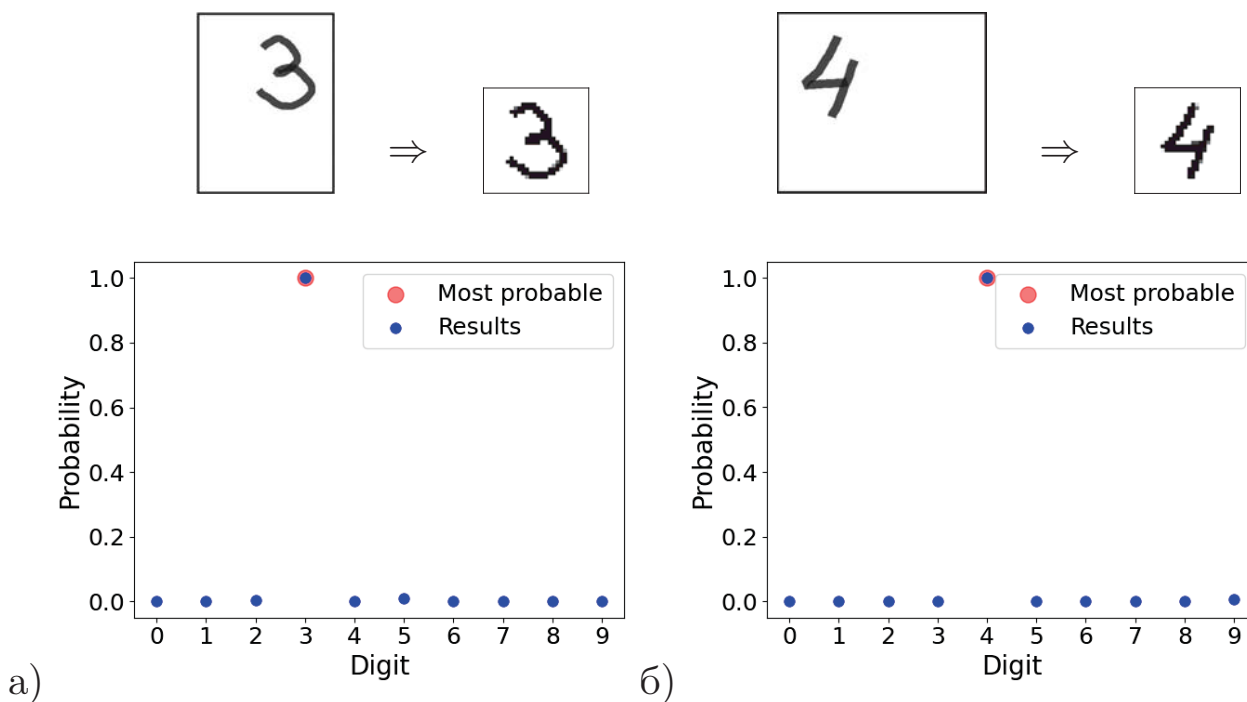
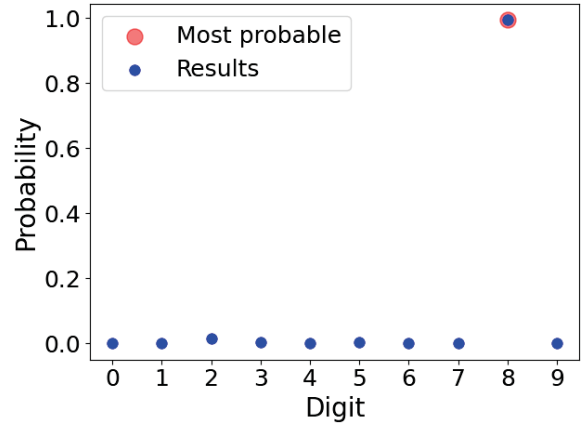
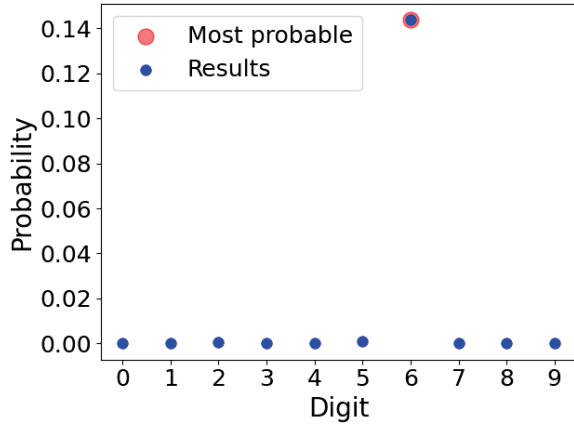
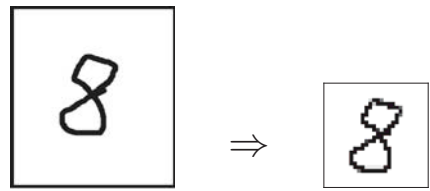
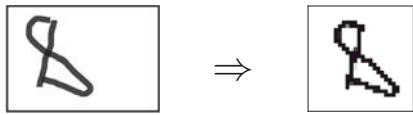


Рисунок 8.12 – Переформатування зображення прописної цифри та результат роботи мережі

Із рисунка 8.12 видно, що алгоритм переформатування зображення впорався з поставленою задачею, а навчена мережа з успіхом упоралась із задачею розпізнавання власних рукописних цифр «3» та «4». На рисунку 8.13 наведено результати для двох різних варіантів написання цифри «8».

Бачимо, що для вісімки на рисунку 8.13а мережа зробила помилку, класифікувавши цифру як «6». Водночас варто зазначити, що ймовірність правильного розпізнавання становить близько 14 %. Такий результат роботи мережі пов'язаний лише з тим, що в більшості прикладів навчальної вибірки для цифри «8» вісімка є або вертикальною, або має діагональний напрям із правого верхнього кута до нижнього лівого, як це видно з рисунка 8.7. Для перевірки цього факту на вхід нейронної мережі подавалось зображення вісімки як на рисунку 8.13б, із яким мережа з успіхом упоралася давши близько 100 % упевненості.



a)

б)

Рисунок 8.13 – Переформатування зображення прописної цифри та результат роботи мережі

8.3.3. Код алгоритму для відцентрування та переформатування будь-якого зображення

```
import math
import cv2
import numpy as np
from scipy.ndimage.measurements import center_of_mass

def get_best_shift(img):
    cy, cx = center_of_mass(img)
    rows, cols = img.shape
    shiftx = np.round(cols / 2.0 - cx).astype(int)
    shifty = np.round(rows / 2.0 - cy).astype(int)
    return shiftx, shifty

def shift(img, sx, sy):
    rows, cols = img.shape
    M = np.float32([[1, 0, sx], [0, 1, sy]])
    shifted = cv2.warpAffine(img, M, (cols, rows))
    return shifted

def rec_digit(img_path):
    img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
    gray = 255 - img
    # застосовуємо порогову обробку
    (thresh, gray) = cv2.threshold(gray, 128, 255, cv2.
                                   THRESH_BINARY | cv2.
                                   THRESH_OTSU)

    # видаляємо нульові рядки та стовбці
    while np.sum(gray[0]) == 0:
        gray = gray[1:]
    while np.sum(gray[:, 0]) == 0:
        gray = np.delete(gray, 0, 1)
    while np.sum(gray[-1]) == 0:
        gray = gray[:-1]
    while np.sum(gray[:, -1]) == 0:
        gray = np.delete(gray, -1, 1)
    rows, cols = gray.shape
```

```

# змінюємо розмір, щоб зображення вміщувалось у квадрат
# розміром 20x20 пікселів
if rows > cols:
    factor = 20.0 / rows
    rows = 20
    cols = int(round(cols * factor))
    gray = cv2.resize(gray, (cols, rows))
else:
    factor = 20.0 / cols
    cols = 20
    rows = int(round(rows * factor))
    gray = cv2.resize(gray, (cols, rows))

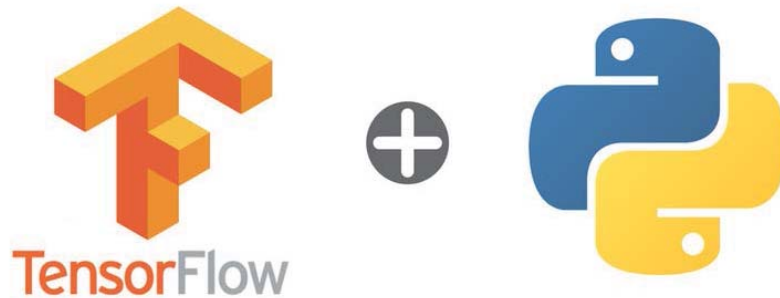
# розширюємо зображення до розміру 28x28
colspadding = (int(math.ceil((28 - cols) / 2.0)), int(
    math.floor((28 - cols) /
                2.0)))
rowspadding = (int(math.ceil((28 - rows) / 2.0)), int(
    math.floor((28 - rows) /
                2.0)))
gray = np.lib.pad(gray, (rowspadding, colspadding), '
    constant')

# міщуємо центр мас
shiftx, shifty = get_best_shift(gray)
shifted = shift(gray, shiftx, shifty)
gray = shifted

cv2.imwrite('gray' + img_path, gray)
img = gray / 255.0
img = np.array(img).reshape(28, 28)
return img

```

Тема 9. Автоматизація навчання нейромереж з використанням пакета TensorFlow



У цьому розділі розглянемо процес навчання моделі нейронної мережі для класифікації зображень одягу та взуття. Для створення нейронної мережі використовуємо бібліотеку TensorFlow.

9.1. Початок роботи

Для роботи нам знадобляться такі бібліотеки:

- `numpy` (інсталяція: `pip install numpy`);
- `matplotlib` (інсталяція: `pip install matplotlib`);
- `keras` (інсталяція: `pip install keras`);
- `tensorflow` (інсталяція `pip install tensorflow`).

```
# Підключаємо бібліотеки
import tensorflow as tf
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt
```

Для прикладу будемо використовувати набір даних Fashion MNIST, який містить 70 000 зображень у відтінках сірого в 10 категоріях.

Ми будемо використовувати 60 000 зображень для навчання мережі та 10 000 зображень, щоб оцінити, наскільки точно мережа навчилася класифікувати зображення. Доступ до Fashion MNIST можна одержати безпосередньо з TensorFlow, просто імпортувавши та завантаживши дані.

```
# Імпорт та завантаження даних
fashion_mnist = keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) =
    fashion_mnist.load_data()
```

Завантаження набору даних повертає чотири масиви NumPy:

- масиви `train_images` та `train_labels` – це дані, які використовує модель для навчання;
- масиви `test_images` та `test_labels` використовують для тестування моделі.

Зображення являють собою NumPy масиви 28×28 , значення пікселів яких знаходяться в діапазоні від 0 до 255. Мітки (`labels`) є масивом цілих чисел від 0 до 9. Вони відповідають класу одягу.

Мітка	Клас
0	T-shirt (Футболка)
1	Trouser (Штани)
2	Pullover (Светр)
3	Dress (Сукня)
4	Coat (Пальто)
5	Sandal (Сандали)
6	Shirt (Сорочка)
7	Sneaker (Кросівки)
8	Bag (Сумка)
9	Ankle boot (Ботильйони)

Імена класів не включені до набору даних, тому прописуємо самі:

```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress',
               ', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle
               boot']
```

9.2. Дослідження даних

Розглянемо формат набору даних перед моделюванням.

```
train_images.shape # У навчальній вибірці міститься 60 000
                   зображень, кожне зображення
                   представлено як  $28 \times 28$ 
                   пікселів

test_images.shape # У тестовій вибірці міститься 10 000
                  зображень, кожне зображення
                  представлено як  $28 \times 28$ 
                  пікселів

len(train_labels) # У навчальній вибірці 60 000 міток

len(test_labels) # У тестовій вибірці 10 000 міток

train_labels # Кожна мітка представляє собою ціле число від
              0 до 9 (Виводиться на екран
              перші 3 мітки та останні 3
              мітки)
```

Перед підготовкою моделі дані повинні бути попередньо оброблені. Значення пікселів першого зображення в навчальній вибірці знаходяться в діапазоні від 0 до 255, як це видно з рисунка 9.1.

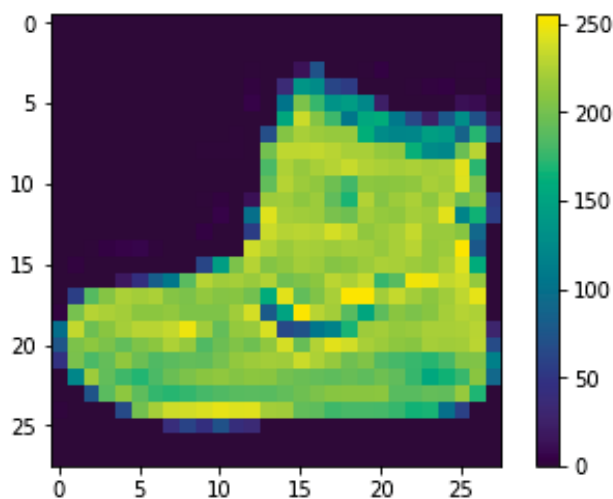


Рисунок 9.1 – Ілюстрація першого зображення в навчальній вибірці

```
plt.figure()
plt.imshow(train_images[0])
plt.colorbar()
plt.grid(False)
```

Масштабуємо ці значення до діапазону від 0 до 1:

```
train_images = train_images / 255.0
test_images = test_images / 255.0
```

Відобразимо перші 25 зображень із тренувального набору та покажемо ім'я класу під кожним зображенням на рисунку 9.2. Переконаймося, що дані знаходяться в правильному форматі.

```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[train_labels[i]])
```

9.3. Побудова моделі

Побудова нейронної мережі потребує налаштування шарів моделі. Основним будівельним блоком нейронної мережі є шар. Більшість глибокого навчання полягає в поєднанні простих верств. Більшість шарів, таких як `tf.keras.layers.Dense` мають параметри, які вивчають під час навчання.

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation=tf.nn.relu),
    keras.layers.Dense(10, activation=tf.nn.softmax)
])
```

Перший шар у мережі `tf.keras.layers.Flatten` перетворює формат зображень з 2d-масиву (28 на 28 пікселів) в 1d-масив із $28 \cdot 28 = 784$ пікселів. Цей шар не має параметрів для навчання, він лише переформатує дані.



Рисунок 9.2 – Зображення окремих предметів одягу з роздільною здатністю (28 × 28 пікселів)

Наступні два шари це `tf.keras.layers.Dense`. Це тісно пов'язані або повністю пов'язані нейронні шари. Перший шар `Dense` містить 128 вузлів (або нейронів). Другий (і останній) рівень – це шар із 10 вузлами `tf.nn.softmax`, який повертає масив із десяти ймовірнісних оцінок, сума яких дорівнює 1. Кожен вузол містить оцінку, яка вказує на ймовірність того, що поточне зображення належить одному з 10 класів.

9.4. Скомпілювання моделі

Перш ніж модель буде готова до навчання, їй знадобиться ще кілька налаштувань. Вони додаються під час етапу компіляції моделі:

- Loss function (функція втрати) – вимірює, наскільки точна модель під час навчання;
- Optimizer (оптимізатор) – це те, як модель оновлюється на основі даних, які вона бачить, та функції втрат;
- Metrics (метрики) – використовують для контролю за етапами навчання та тестування.

```
model.compile(optimizer=tf.train.AdamOptimizer(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

9.5. Навчання моделі

Навчання моделі нейронної мережі потребує кроків, описаних нижче.

1. Подання даних навчання моделі (у цьому прикладі – масиви `train_images` та `train_labels`).
2. Модель учитися асоціювати зображення та мітки.
3. Ми просимо модель зробити прогнози про тестовий набір (у цьому прикладі – масив `test_images`). Ми перевіряємо відповідність прогнозів міток із масиву міток (у цьому прикладі – масив `test_labels`).

Щоб розпочати навчання, потрібно викликати метод `model.fit`.

```
model.fit(train_images, train_labels, epochs=5)
```

```
Epoch 1/5
60000/60000 [=====] - 5s 90us/step - loss: 0.5001 - acc: 0.8253
Epoch 2/5
60000/60000 [=====] - 5s 88us/step - loss: 0.3761 - acc: 0.8644
Epoch 3/5
60000/60000 [=====] - 5s 89us/step - loss: 0.3384 - acc: 0.8759
Epoch 4/5
60000/60000 [=====] - 5s 89us/step - loss: 0.3124 - acc: 0.8857
Epoch 5/5
60000/60000 [=====] - 5s 86us/step - loss: 0.2946 - acc: 0.8922
```

Під час моделювання відображаються показники втрат (loss) та точності (acc). Ця модель досягає точності близько 0.88 % (або 88 %) за даними навчання.

9.6. Оцінка точності

Порівняємо, як модель працює в тестовому наборі даних.

```
test_loss, test_acc = model.evaluate(test_images,
                                     test_labels)
print('Test accuracy:', test_acc)
```

```
10000/10000 [=====] - 1s 53us/step
Test accuracy: 0.8777
```

Виявляється, точність у тестовому наборі даних трохи менша за точність у тренувальному наборі.

9.7. Прогнозування

Використовуємо навчену модель нейронної мережі для прогнозування деяких зображень.

```
predictions = model.predict(test_images)
```

Тут модель передбачила мітку для кожного зображення тестового набору. Давайте подивимося на перший прогноз.

```
predictions[0]
```

```
array([4.0652740e-06, 6.9819279e-08, 2.5388722e-06, 1.3390627e-07,
       1.1847248e-07, 2.9022932e-02, 2.0918555e-06, 6.4492501e-02,
       9.1468155e-06, 9.0646631e-01], dtype=float32)
```

Передбачення є масивом із 10 чисел. Вони описують «упевненість» моделі в тому, що зображення відповідає кожному з 10 різних предметів одягу. Ми можемо бачити, яка мітка має найбільше довірче значення:

```
np.argmax(predictions[0]) # 9
```

Отже, модель найбільш упевнена в тому, що це зображення – Ankle boot (Ботильйони), або `class_names[9]`. І ми можемо перевірити тестову мітку, щоб переконатися, що це правильно.

```
test_labels[0] # 9
```

Напишемо функції для візуалізації цих прогнозів

```
def plot_image(i, predictions_array, true_label, img):
    predictions_array, true_label, img = predictions_array[
        i], true_label[i], img[i]

    plt.grid(False)
    plt.xticks([])
    plt.yticks([])

    plt.imshow(img, cmap=plt.cm.binary)

    predicted_label = np.argmax(predictions_array)
    if predicted_label == true_label:
        color = 'blue'
    else:
        color = 'red'

    plt.xlabel("{} {:2.0f}% ({})" .format(class_names[
        predicted_label], 100*np.
        max(predictions_array),
        class_names[true_label]),
        color=color)
```

```
def plot_value_array(i, predictions_array, true_label):
    predictions_array, true_label = predictions_array[i],
        true_label[i]

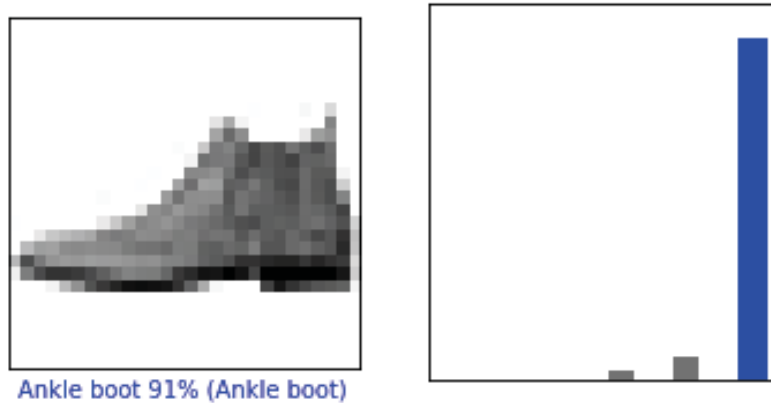
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])
    thisplot = plt.bar(range(10), predictions_array, color=
        "#777777")

    plt.ylim([0, 1])
    predicted_label = np.argmax(predictions_array)

    thisplot[predicted_label].set_color('red')
    thisplot[true_label].set_color('blue')
```

Подивимося на перше (з індексом 0) зображення, передбачення та масив передбачень.

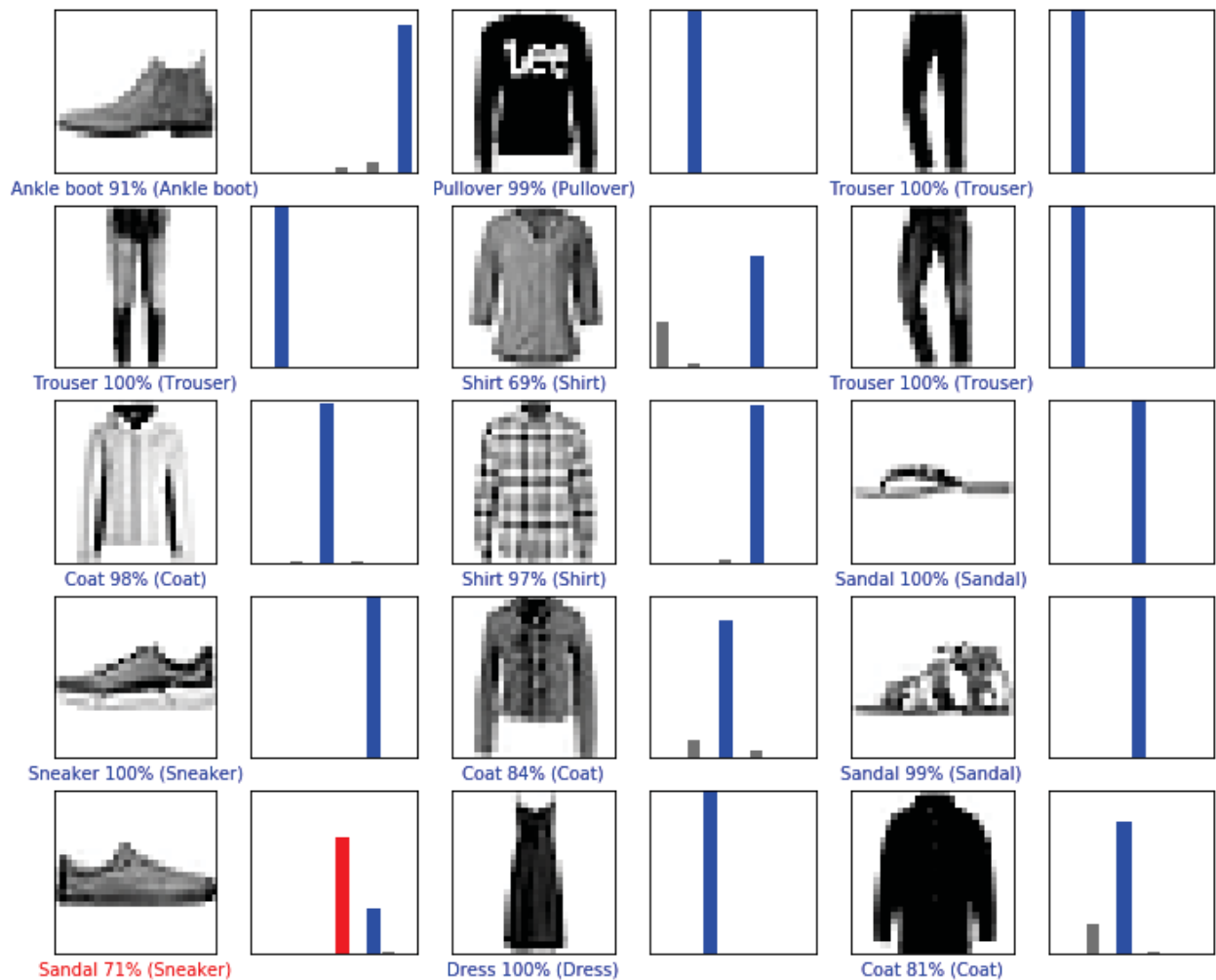
```
i = 0
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions, test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions, test_labels)
```



Побудуємо кілька зображень із їх прогнозами. Правильні мітки прогнозу – сині, а неправильні мітки прогнозу – червоні.

```
num_rows = 5
num_cols = 3
num_images = num_rows*num_cols
plt.figure(figsize=(2*2*num_cols, 2*num_rows))
```

```
for i in range(num_images):
    plt.subplot(num_rows, 2*num_cols, 2*i+1)
    plot_image(i, predictions, test_labels, test_images)
    plt.subplot(num_rows, 2*num_cols, 2*i+2)
    plot_value_array(i, predictions, test_labels)
```



Нарешті, використовуємо навчену модель, щоб зробити прогноз одного зображення.

```
# Візьмемо зображення з тестового набору даних
img = test_images[0]
```

Моделі `tf.keras` оптимізовані для того щоб робити прогнози на пакети (`batch`) або колекції (`collection`). Тому, хоча ми використовуємо одне зображення, нам потрібно додати його до пакету.

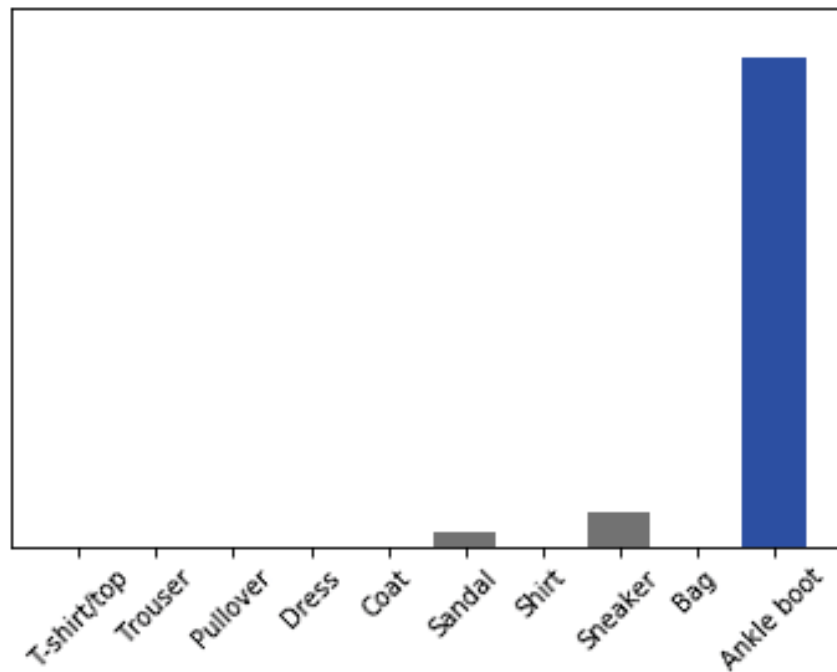
```
# Додамо зображення в пакет, де він є єдиним прикладом
img = (np.expand_dims(img, 0))
```

Робимо прогноз для зображення.

```
predictions_single = model.predict(img)
print(predictions_single)
```

```
[[4.0652740e-06 6.9819272e-08 2.5388672e-06 1.3390652e-07 1.1847247e-07  
2.9022938e-02 2.0918596e-06 6.4492591e-02 9.1468228e-06 9.0646625e-01]]
```

```
plot_value_array(0, predictions_single, test_labels)  
_ = plt.xticks(range(10), class_names, rotation=45)
```



```
np.argmax(predictions_single[0])
```

Як і раніше, модель передбачає мітку 9.

Розділ 3. Асоціативна пам'ять, класифікація без учителя та кластеризація

Тема 10. Організація асоціативної пам'яті за допомогою мережі Хопфілда

Серед різних конфігурацій штучних нейронних мереж трапляються такі, під час класифікації яких за принципом навчання, строго кажучи, не підходять ні навчання з учителем, ні навчання без учителя. У таких мережах вагові коефіцієнти синапсів розраховують лише один раз перед початком функціонування мережі на основі інформації про оброблювані дані, і все навчання мережі зводиться саме до цього розрахунку. З одного боку, пред'явлення апріорної інформації можна розцінювати, як допомогу вчителя, але з іншого – мережа фактично просто запам'ятовує зразки до того, як на її вхід надходять реальні дані, і не може змінювати свою поведінку, тому говорити про ланку зворотного зв'язку зі «світом» (учителем) не доводиться. Серед мереж із подібною логікою роботи найбільш відомі мережа Хопфілда та мережа Хеммінга, які зазвичай використовуються для організації асоціативної пам'яті.

10.1. Задача класифікації мережею Хопфілда

Завдання, яке вирішується цією мережею як асоціативна пам'ять, зазвичай формулюється так: відомий деякий набір двійкових сигналів (зображень, звукових відцифровок, інших даних, що описують об'єкти або характеристики процесів), які вважають

еталонними. Мережа повинна вміти з довільного неідеального сигналу, поданого на її вхід, виділити («згадати» за частковою інформацією) відповідний зразок-еталон (якщо такий є) або «дати висновок» про те, що вхідні дані не відповідають жодному еталонному зразку. Загалом, будь-який сигнал може бути описаний вектором $\mathbf{X} = (x_1, x_2, \dots, x_M)$, M – число нейронів у мережі та розмірність вхідних і вихідних векторів. Кожен елемент x_i дорівнює або $+1$, або -1 . Позначимо вектор, що описує k -ий зразок, через $\mathbf{X}^{(k)}$, а його компоненти, відповідно, x_i^k , $k = 1, \dots, N$, N – кількість зразків. Коли мережа розпізнає (або «згадає») який-небудь зразок на основі пред'явлених їй даних, її виходи будуть містити саме його, тобто $\mathbf{Y} = \mathbf{X}^{(k)}$, де \mathbf{Y} – вектор вихідних значень мережі: $\mathbf{Y} = \{y_i : i = 1, \dots, M\}$. В іншому разі вихідний вектор не збігається з жодним зразковим.

Якщо, наприклад, сигнали являють собою деякі зображення, то, відобразивши у графічному вигляді дані з виходу мережі, можна буде побачити картинку, що повністю збігається з однією з еталонних (у разі успіху) або «вільну імпровізацію» мережі (у разі невдачі). Отже, нейронна мережа Хопфілда, є авто асоціативною мережею, здатною виконувати функцію пам'яті.

Припустимо, ми хочемо зберегти за допомогою мережі Хопфілда цифри від нуля до дев'яти. Розглянемо, наприклад, цифри 2, 4, 6, 8, подані на рисунку 10.1а. Це еталонні зразки, які запам'ятовуються. Нехай мережу вже навчено, тобто цю цифру, як і решту, ми зберегли в пам'яті. Тоді, якщо ми подамо зашумлений варіант збереженої цифри на вхід, мережа повинна видати нам правильне зображення (те, яке ми зберегли під час навчання, без спотворень).

Наприклад, зразок, що подається на вхід, може мати вигляд, як показано на рисунку 10.1б, а в результаті своєї роботи мережа повинна видати нам перший, еталонний варіант як на рисунку 10.1а. У цьому полягає завдання нейронної мережі Хопфілда під час використання як авто асоціативної пам'яті.

Крім того, нейронну мережу Хопфілда використовують для фільтрації даних та вирішення задач оптимізації.

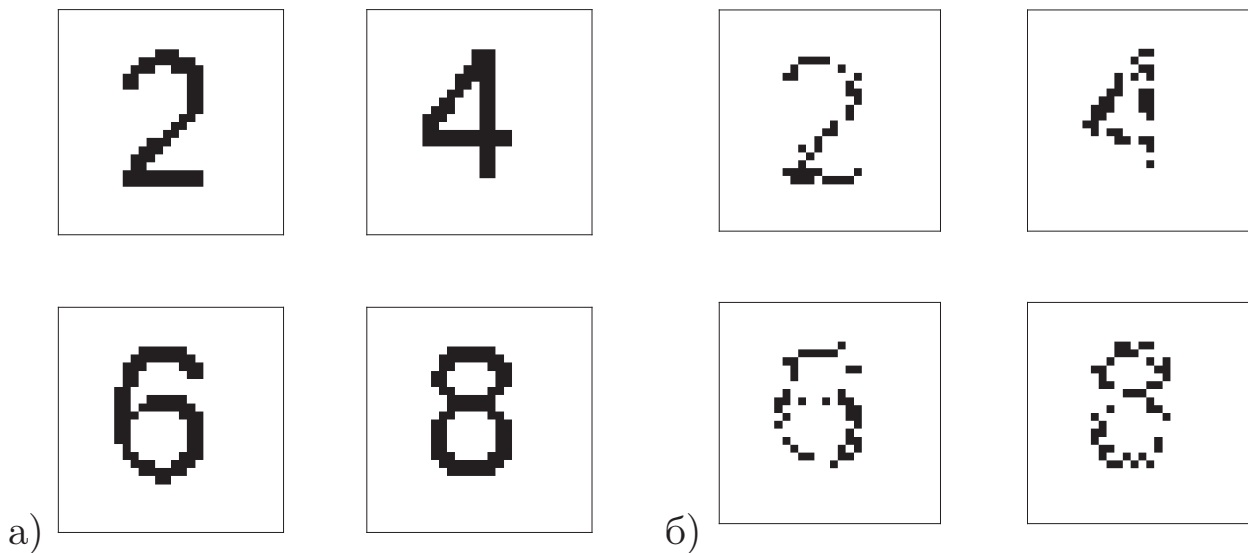


Рисунок 10.1 – Приклади а) еталонних та б) тестових зображень

10.2. Структура мережі Хопфілда

Структурну схему мережі Хопфілда наведено на рисунку 10.2. Вона складається з єдиного шару нейронів, кількість яких є одночасно числом входів та виходів мережі. Кожен нейрон пов'язаний синапсами з іншими нейронами, але не впливає на себе самого, а також має один вхідний синапс, через який здійснюється введення сигналу. Вихідні сигнали зазвичай утворюються на аксонах.

Для активації нейронів мережі Хопфілда використовують жорстку порогову функцію активації, наведену на рисунку 10.3.

Комбінований сигнал на вході кожного з нейронів шару Хопфілда обчислюють із використанням вагових коефіцієнтів зв'язків

$$x_j = \sum_{i=1}^M s_i w_{ij}. \quad (10.1)$$

Тут s_i – стан нейрона з номером i . Якщо комбінований вхід елемента виявляється негативним, стан елемента стає рівним -1 , навпаки, коли комбінований вхід позитивний, то стан елемента відповідає значенню $+1$. Якщо ж $x_j = 0$, то нейрон не змінює свого стану, тобто зберігає попередній.

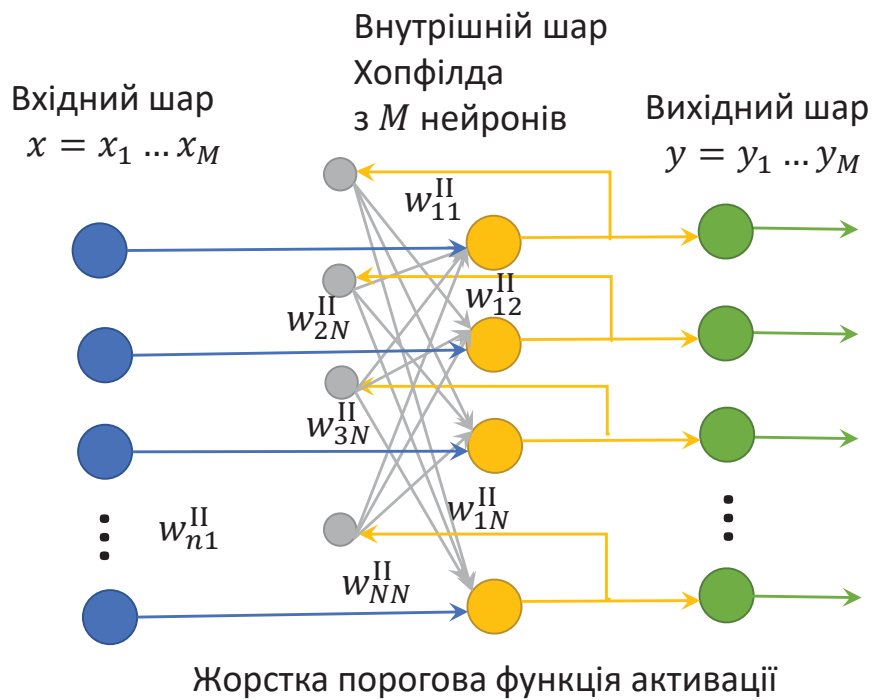


Рисунок 10.2 – Структура нейронної мережі Хопфілда

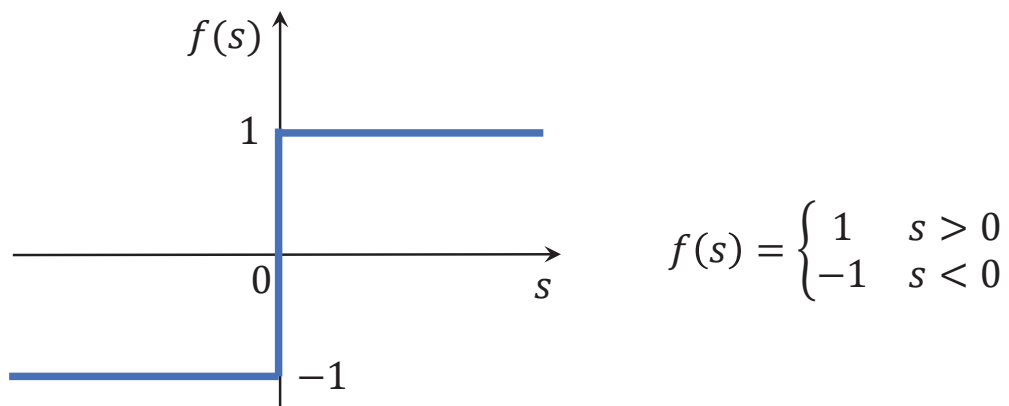


Рисунок 10.3 – Жорстка порогова активаційна функція нейронної мережі Хопфілда

10.3. Навчання мережі Хопфілда

Навчання мережі Хопфілда значно відрізняється від «звичних» алгоритмів навчання (наприклад, навчання за методом зворотного поширення помилки). У випадку мережі Хопфілда процедура навчання для одного вхідного зразка (процедура збереження

зразка) є просто розрахунком вагових коефіцієнтів за цілком конкретною формулою

$$\mathbf{W} = \mathbf{x}^T \mathbf{x}. \quad (10.2)$$

Тут \mathbf{W} – це матриця вагових коефіцієнтів розміром $M \times M$, \mathbf{x} – вхідний зразок, а \mathbf{x}^T – транспонований вектор x . За умови наявності декількох кілька зразків (еталонних векторів) для зберігання (k штук), необхідно обчислити матрицю вагових коефіцієнтів для кожного еталонного вектора, а потім скласти одержані матриці \mathbf{W}_k

$$\mathbf{W} = \mathbf{W}_1 + \mathbf{W}_2 + \dots + \mathbf{W}_k. \quad (10.3)$$

Отже, цей процес, знову ж таки, на відміну від мереж інших типів, відбувається лише в один етап. Потрібно зазначити один важливий аспект. Як зазначалося вище, нейрони в мережі Хопфілда не впливають самі на себе. Тому для забезпечити виконання цієї умови необхідно обнулити діагональні елементи одержаної в результаті збереження зразків матриці. Отже, матриця вагових коефіцієнтів матиме такий вигляд:

$$\mathbf{W} = \begin{bmatrix} 0 & w_{12} & w_{13} & \dots & w_{1M} \\ w_{21} & 0 & w_{23} & \dots & w_{2M} \\ w_{31} & w_{32} & 0 & \dots & w_{3M} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{M1} & w_{M2} & w_{M3} & \dots & 0 \end{bmatrix}. \quad (10.4)$$

Ми одержали навчену мережу Хопфілда. Кроки функціонування мережі Хопфілда такі:

- 1) на вхід подається дещо змінений приклад – зашумлене зображення (наприклад цифри, як на рисунку 10.16);
- 2) у результаті ітеративної процедури стани нейронів змінюються відповідно до формули

$$s_j(p+1) = \sum_{i=1}^M w_{ij} x_j(p), \quad j = 1, \dots, M; \quad (10.5)$$

- 3) за одержаними значеннями станів нейронів із використанням активаційної функції розраховують новий сигнал

$$x_j(p+1) = f(s_j(p+1)). \quad (10.6)$$

Ітераційна процедура повторюється, поки мережа перейде в стійкий стан, тобто в результаті подальших оновлень жоден із нейронів не змінюватиме свого стану. У результаті всіх цих процесів ми одержимо збережений (еталонний) варіант зразка (зображення), що подається на вхід.

Іноді мережа не може провести розпізнавання і видає на виході неіснуючий образ. Це пов'язано з проблемою обмеженості можливостей мережі. Для мережі Хопфілда число образів k , що запам'ятовуються, не повинно перевищувати величини $k_c \simeq 0.15M$. Крім того, якщо два образи А і Б дуже схожі, вони, можливо, викликать у мережі перехресні асоціації, тобто пред'явлення входу мережі вектора А призведе до появи на її виходах вектора Б і навпаки.

10.4. Стійкість мережі

Для успішної роботи мережі Хопфілда необхідно, щоб нейрони перейшли в якийсь стійкий стан, власне, звідси й витікає питання – чи стійка мережа взагалі. Розглянемо два можливі режими роботи.

1. **Синхронний режим.** Ідея полягає в тому, що розрахунки для всіх нейронів проводять послідовно, тобто по черзі. Але їхні стани не змінюються, а запам'ятовуються. І лише після того, як були пройдені всі нейрони мережі, їх стани одночасно змінюються на нові.

2. **Асинхронний режим.** Тут відмінність полягає в тому, що стани нейронів змінюються одночасно (не синхронно). Тобто після проведення розрахунків для першого нейрона встановлюють його новий стан. Далі проводять обчислення для другого нейрона, але вже з урахуванням нового стану першого. І також слідом за цим змінюють стан другого нейрона. І так далі.

Насправді в переважній кількості випадків використовується другий варіант, саме асинхронний режим. І, повертаючись до питання стійкості, нейронна мережа Хопфілда, що працює в асинхронному режимі, завжди буде стійкою.

Розберемося, із чого випливає ця властивість. Уведемо поняття функції енергії системи

$$E = -\frac{1}{2} \sum_{j=1}^M \sum_{i=1}^M s_i s_j w_{ij}. \quad (10.7)$$

Під час зміни стану нейрона j на Δs_j енергія зміниться на величину

$$\Delta E = -\Delta s_j \sum_{i=1}^M s_i w_{ij}. \quad (10.8)$$

Під знаком суми тут знаходиться вхідний сигнал елемента елемента j . Отже, якщо сигнал на вході позитивний, то зміна стану нейрона j позитивне ($\Delta s_j > 0$). Аналогічно, якщо $\sum_{i=1}^M s_i w_{ij} < 0$, то $\Delta s_j < 0$. Тому зміна енергії завжди буде негативною (енергія зменшуватиметься в процесі роботи мережі), що й гарантує нам те, що мережа є стійкою.

10.5. Приклади роботи мережі Хопфілда

10.5.1. Один вектор для запам'ятовування

Нехай ϵ зразок (еталонний вектор) для запам'ятовування

$$x = [1 \quad 1 \quad -1 \quad -1]. \quad (10.9)$$

Визначимо матрицю вагових коефіцієнтів

$$\mathbf{W} = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix} \times [1 \quad 1 \quad -1 \quad -1] = \begin{bmatrix} 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \\ -1 & -1 & 1 & 1 \end{bmatrix}. \quad (10.10)$$

Зануляємо елементи на головній діагоналі

$$\mathbf{W} = \begin{bmatrix} 0 & 1 & -1 & -1 \\ 1 & 0 & -1 & -1 \\ -1 & -1 & 0 & 1 \\ -1 & -1 & 1 & 0 \end{bmatrix}. \quad (10.11)$$

Мережа готова до роботи. подаємо на вхід зашумлений сигнал, наприклад

$$x = [1 \quad 1 \quad -1 \quad 1]. \quad (10.12)$$

Послідовно проведемо обчислення

$$\begin{aligned} x_1 &= 1 \cdot 0 + 1 \cdot 1 + (-1) \cdot (-1) + 1 \cdot (-1) = 1 \\ x_1 > 0 &\Rightarrow s_1 = 1 \\ x_2 &= 1 \cdot 1 + 1 \cdot 0 + (-1) \cdot (-1) + 1 \cdot (-1) = 1 \\ x_2 > 0 &\Rightarrow s_2 = 1 \\ x_3 &= 1 \cdot (-1) + 1 \cdot (-1) + (-1) \cdot 0 + 1 \cdot 1 = -1 \\ x_3 < 0 &\Rightarrow s_3 = -1 \\ x_4 &= 1 \cdot (-1) + 1 \cdot (-1) + (-1) \cdot 1 + 1 \cdot 0 = -1 \\ x_4 < 0 &\Rightarrow s_4 = -1. \end{aligned} \quad (10.13)$$

У результаті на виході мережі одержали той самий зразок, який раніше було збережено. Тепер необхідно переконатися, що мережа стійка, а значить нейрони не змінять свого стану в разі повторного проходу

$$\begin{aligned} x_1 &= 1 \cdot 0 + 1 \cdot 1 + (-1) \cdot (-1) + (-1) \cdot (-1) = 1 \\ s_1 &= 1 \\ x_2 &= 1 \cdot 1 + 1 \cdot 0 + (-1) \cdot (-1) + (-1) \cdot (-1) = 1 \\ s_2 &= 1 \\ x_3 &= 1 \cdot (-1) + 1 \cdot (-1) + (-1) \cdot 0 + (-1) \cdot 1 = -1 \\ s_3 &= -1 \\ x_4 &= 1 \cdot (-1) + 1 \cdot (-1) + (-1) \cdot 1 + (-1) \cdot 0 = -1 \\ s_4 &= -1. \end{aligned} \quad (10.14)$$

Бачимо, що нейронна мережа Хопфілда виконала свою функцію пам'яті й дозволила відновити збережений зразок за зашумленим варіантом, що подається на вхід.

10.5.2. Робота мережі з декількома еталонними векторами

Розглянемо більш складний приклад розпізнавання образів. Для цього визначимо три еталонні зображення букв, як показано на рисунку 10.4. Згенеруємо матрицю вагових коефіцієнтів з вико-



Рисунок 10.4 – Еталонні зображення для запам'ятовування

ристанням правила (10.3). Далі згенеруємо тестові зображення – зашумлені варіанти еталонних зображень двома способами:

- 1) з певною ймовірністю затирання інформативного пікселя;
- 2) з певною ймовірністю заміна кольору кожного пікселя.

Типові приклади зашумлених зображень для відтворення наведено на рисунку 10.5.

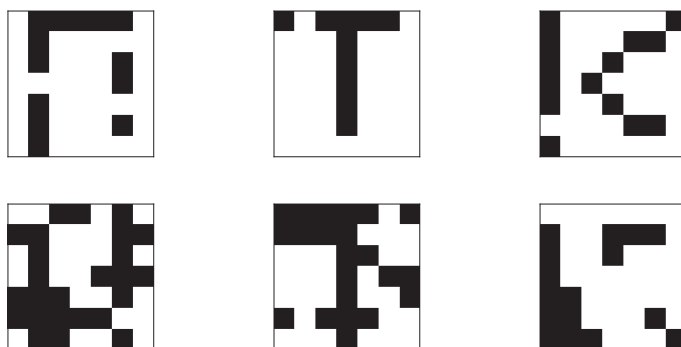


Рисунок 10.5 – Зашумлені зображення для відтворення за першим та другим варіантом з ймовірністю 20 %

Далі проаналізуємо, як справиться навчена мережа під час зміни ймовірності зашумлення зображення. Для цього для кожної літери з еталонних зображень будемо генерувати 1 000 зашумлених екземплярів для кожного з двох варіантів та пропускати їх через

мережу. Водночас будемо рахувати кількість помилок для кожного значення імовірності зашумлення зображень. Результати наведено на рисунку 10.6 для першого варіанта генерації шуму та на рисунку 10.7 для другого варіанта.

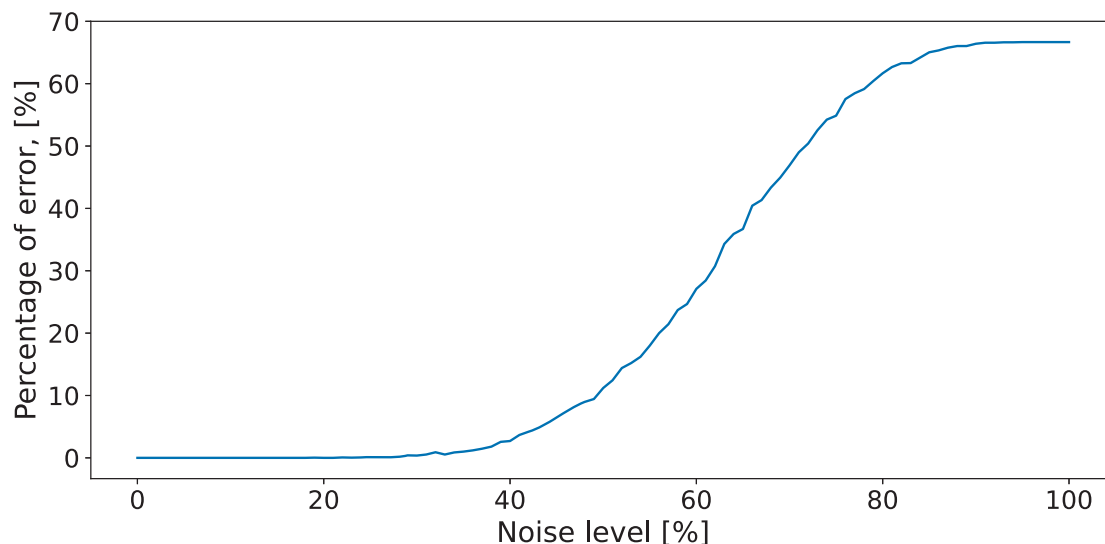


Рисунок 10.6 – Залежність помилки класифікації від імовірності зашумлення зображень для варіанта затирання інформаційних пікселів

Для першого варіанта генерації безладу навіть у разі затирання половини інформаційних пікселів мережа працює досить непогано даючи приблизно 90 % правильних відповідей, як видно з результатів на рисунку 10.6. Водночас, навіть при 100 % безладу точність роботи мережі становить 33 %. Це пов'язано з тим, що в разі сильного безладу (велику кількість чорних пікселів замінено на білі) мережа видає як результат класифікації літеру «Т». Це пояснюють тим, що саме на зображенні цифри «Т» кількість білих пікселів максимальна, порівняно з двома іншими літерами. Оскільки букви подають у вигляді векторів з координатами -1 (білий колір) та $+1$ (чорний колір), то за максимальним співпадінням кількості білих пікселів за умов рівня шуму > 50 % будь-яке зашумлене за першим варіантом зображення найкраще співпадає з літерою «Т». Тому в третині випадків, коли на вхід мережі подається сильно зашумлене зображення будь-якої літери, то мережа класифікує зображення як

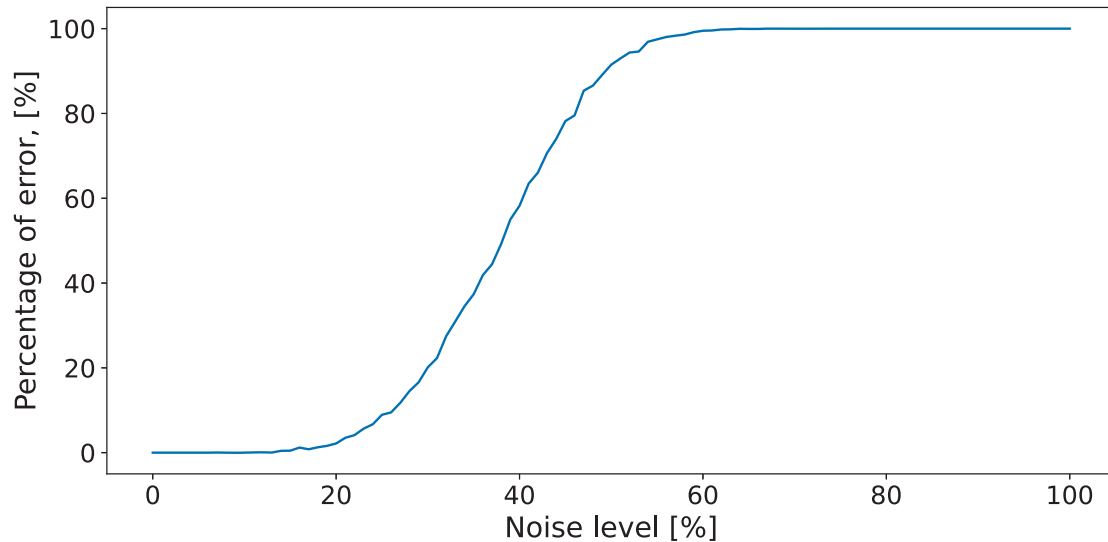


Рисунок 10.7 – Залежність помилки класифікації від імовірності зашумлення зображень для варіанта генерації безладу всього зображення

літеру «Г» і зашумлене зображення літери «Г» мережа розпізнає правильно.

Для варіанта генерації безладу всього зображення результати дещо відрізняються. Тут помилка класифікації стрімко зростає в разі ймовірності генерації безладу більше за 0.2, як видно з рисунка 10.7. Крім того, у разі значень рівня шуму $> 60\%$ мережа дає 100 % неправильних відповідей.

Тема 11. Використання мережі Хемінга для класифікації

Нейронна мережа Хеммінга – це тип нейронної мережі, яку використовують для класифікації бінарних векторів і вона ґрунтується на коді Хеммінга. Код Хеммінга є методом виявлення та виправлення помилок у передаванні даних. У нейронній мережі Хеммінга використовують спеціальні нейрони, які можуть опрацьовувати інформацію з урахуванням можливих помилок у даних. Це дозволяє досягти більш надійної роботи системи під час передачі та оброблення інформації. Нейронна мережа Хеммінга була запропонована Річардом Ліппманном у 1987 році та позиціонувалася як спеціалізований гетероасоціативний пристрій. Основним критерієм цієї нейронної мережі є відстань Хеммінга. Мережу використовують для того, щоб співвіднести тестовий образ, який подається бінарним вектором $\mathbf{x} = (x_1, x_2, x_3, \dots, x_m)$, де $x_i = \{-1, 1\}$, з одним з еталонних образів (бінарним вектором) $\mathbf{e} = (e_1, e_2, e_3, \dots, e_m)$, водночас кожному класу відповідає свій образ. Завдання мережі Хеммінга – вирішити якому саме еталонному образу найімовірніше відповідає тестовий образ. Отже, нейронну мережу Хеммінга можна використовувати для оброблення інформації, вирішення завдань, пов'язаних із виявленням та виправленням помилок у переданих даних, перевірки цілісності даних та контролю помилок у системах передавання інформації.

11.1. Архітектура нейронної мережі Хемінга

Мережа Хемінга – це тришарова нейронна мережа зі зворотним зв'язком. Вхідний шар складається з M входів, де M визначає розмір сигналу. Кількість нейронів у другому та третьому шарах дорівнює кількості класів класифікації. Синапси нейронів другого шару пов'язані з кожним входом мережі, нейрони третього шару пов'язані між собою негативними зв'язками, крім синапсу,

що з власним аксоном кожного нейрона має позитивний зворотний зв'язок. Структуру нейронної мережі Хемінга схематично подано на рисунку 11.1.

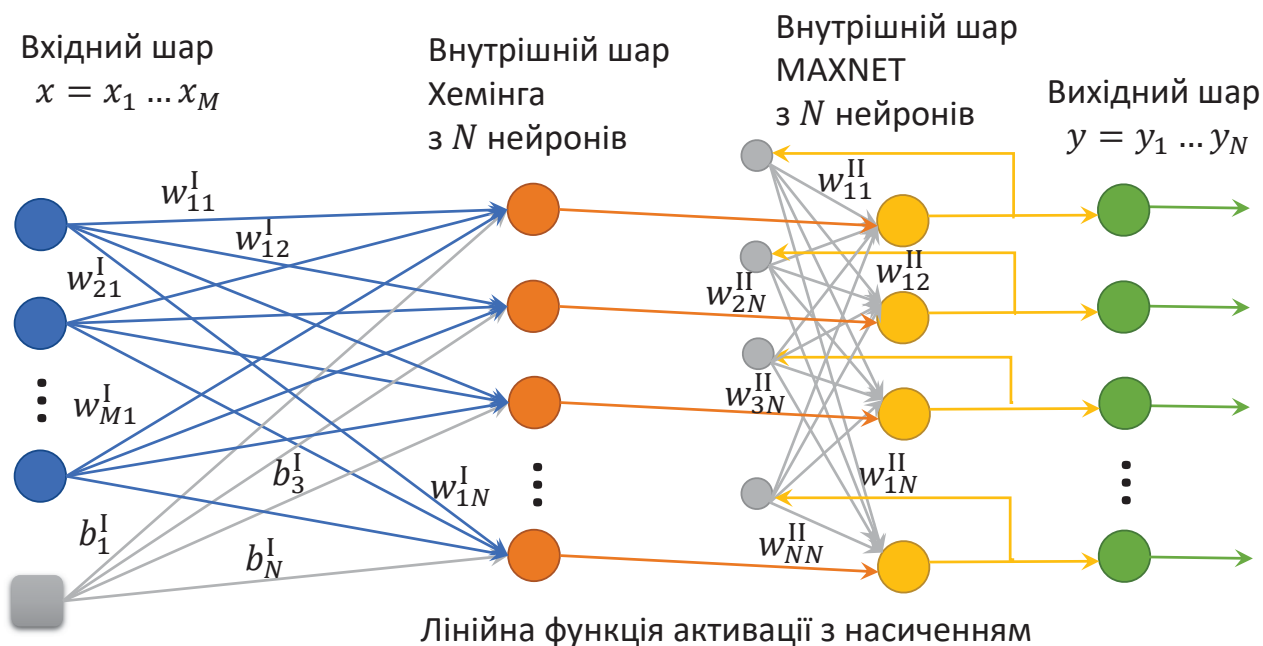


Рисунок 11.1 – Схематичне подання тришарової структури нейронної мережі Хемінга

Перший прихований шар – шар Хемінга, що виконує порівняння вхідного сигналу з еталонними векторами. Другий прихований шар – шар *MAXNET* зі зворотним зв'язком, який визначає найбільш імовірний клас для вхідного тестового сигналу. Для нейронів другого та третього шару використовують лінійну функцію активації з насиченням, подану на рисунку 11.2.

Відповідно до принципу роботи нейронної мережі Хемінга, кожний тестовий сигнал порівнюють із набором тестових сигналів. Для оцінювання міри близькості до кожного класу використовують критерій, що враховує відстань Хемінга – кількість змінних, що різняться, у тестового й еталонного вхідних образів.

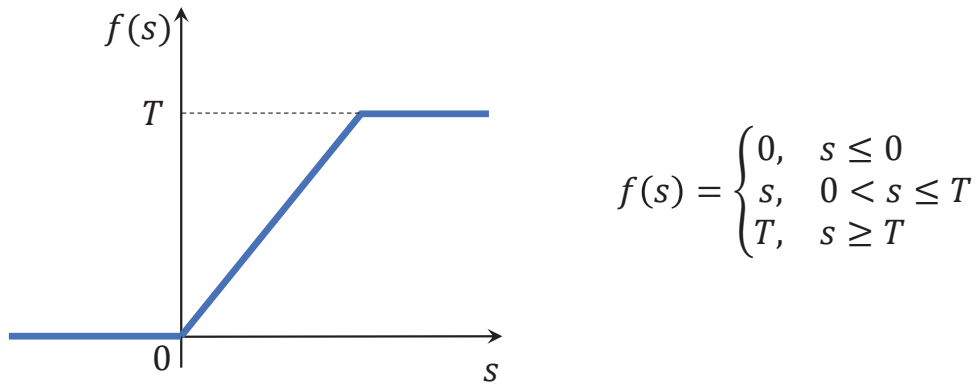


Рисунок 11.2 – Лінійна функція активації з насиченням для нейронної мережі Хемінга

11.2. Прихований шар Хемінга

Із математичного погляду порівняти вектори \mathbf{a} та \mathbf{b} – знайти скалярний добуток $\mathbf{a} \cdot \mathbf{b} = \sum_i^N a_i b_i$ цих векторів. Водночас урахувавши бінарну структуру еталонних та тестових векторів, для скалярного добутку векторів \mathbf{a} та \mathbf{b} маємо такі властивості:

- якщо певні координати a_i та b_i обох векторів є однаковими, тобто: $a_i = b_i = +1$ або $a_i = b_i = -1$, то добуток цих координат $a_i b_i$ дасть $(+1)$;
- якщо певні координати a_i та b_i обох векторів є різними, тобто: $a_i = +1, b_i = -1$ або $a_i = -1, b_i = +1$, то добуток цих координат $a_i b_i$ буде давати (-1) .

Відстань Хемінга для двох векторів $H(\mathbf{a}, \mathbf{b})$ – це кількість «бітів» (координат векторів) \mathbf{a} та \mathbf{b} , які мають протилежні значення, добуток яких дає (-1) . Беручи до уваги, що довжина векторів (тестового й еталонного) M є однаковою, то кількість координат, які для двох векторів є однаковими за значенням (коли $a_i b_i = (+1)$) можна подати у вигляді

$$A(\mathbf{a}, \mathbf{b}) = M - H(\mathbf{a}, \mathbf{b}). \quad (11.1)$$

Ураховуючи наведені властивості можна подати скалярний добуток двох біполярних векторів \mathbf{a} та \mathbf{b} у такому вигляді:

$$\mathbf{a} \cdot \mathbf{b} = A(\mathbf{a}, \mathbf{b}) - H(\mathbf{a}, \mathbf{b}). \quad (11.2)$$

Далі, виражаючи відстань Хемінга $H(\mathbf{a}, \mathbf{b})$ з рівняння (11.2) та підставляючи його в рівняння (11.1) після тривіальних перетворень одержуємо вираз для кількості $A(\mathbf{a}, \mathbf{b})$ координат двох векторів, значення яких співпадають у такому вигляді:

$$A(\mathbf{a}, \mathbf{b}) = \frac{1}{2} \mathbf{a} \cdot \mathbf{b} + \frac{M}{2}. \quad (11.3)$$

Отже, кількість координат двох біполярних векторів \mathbf{a} та \mathbf{b} буде тим більшою, чим більшим буде скалярний добуток цих векторів. Оскільки один із векторів \mathbf{a} , \mathbf{b} є еталонним, а інший – тестовим, то вагові коефіцієнти першого прихованого шару w_{ij}^I можна визначити через координати еталонних векторів так:

$$w_{ij}^I = \frac{1}{2} e_{ij}, \quad i = 1, \dots, M; \quad j = 1, \dots, N. \quad (11.4)$$

Значення параметра зсуву для нейронів першого прихованого шару b_j^I будуть однаковими для всіх нейронів

$$b_j^I = \frac{1}{2} M, \quad j = 1, \dots, N. \quad (11.5)$$

У результаті шар Хемінга буде генерувати вектор відповідей $\mathbf{y}^H = (y_1^H, \dots, y_N^H)$ використовуючи матрицю вагових коефіцієнтів

$$\mathbf{w}^I = \begin{bmatrix} w_{11}^I & w_{12}^I & \cdots & w_{1N}^I \\ w_{21}^I & w_{22}^I & \cdots & w_{2N}^I \\ \vdots & \vdots & \vdots & \vdots \\ w_{M1}^I & w_{M2}^I & \cdots & w_{MN}^I \end{bmatrix} \quad (11.6)$$

та вектор коефіцієнтів зміщень

$$\mathbf{b}^I = [b_1^I \quad b_2^I \quad \cdots \quad b_N^I] \quad (11.7)$$

у такому вигляді:

$$\mathbf{y}^H = \mathbf{xw}^I + \mathbf{b}^I. \quad (11.8)$$

Розмірність вихідного вектора \mathbf{y}^H буде визначатися кількістю нейронів Хемінга, а отже – кількістю нейронів вихідного шару

всієї мережі – класів (еталонних векторів). Формально можна вважати, що номер координати вектора $\mathbf{y}^{\mathbf{H}}$ з найбільшим значенням буде визначати клас, до якого найбільш імовірно належить тестовий вектор \mathbf{x} .

Розглянемо приклад роботи прихованого шару Хемінга. Для цього будемо використовувати два еталонних вектори \mathbf{e}_1 та \mathbf{e}_2 у такому вигляді:

$$\mathbf{e}_1 = [1 \quad -1 \quad -1 \quad -1], \quad \mathbf{e}_2 = [-1 \quad -1 \quad -1 \quad 1]. \quad (11.9)$$

За тестові вектори візьмемо такі чотири вектори:

$$\begin{aligned} \mathbf{x}_1 &= [1 \quad 1 \quad -1 \quad -1], \\ \mathbf{x}_2 &= [1 \quad -1 \quad -1 \quad -1], \\ \mathbf{x}_3 &= [-1 \quad -1 \quad -1 \quad 1], \\ \mathbf{x}_4 &= [-1 \quad -1 \quad 1 \quad 1]. \end{aligned} \quad (11.10)$$

Відповідно до еталонних векторів матриця вагових коефіцієнтів $\mathbf{w}^{\mathbf{I}}$ та вектор коефіцієнтів зміщень $\mathbf{b}^{\mathbf{I}}$ матимуть такий вигляд:

$$\mathbf{w}^{\mathbf{I}} = \begin{bmatrix} \frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} \end{bmatrix}, \quad \mathbf{b}^{\mathbf{I}} = [2 \quad 2]. \quad (11.11)$$

Розрахунок виходів шару Хемінга за формулою (11.8) для кожного тестового вектора \mathbf{x}_i (11.10) дає

$$\mathbf{y}_1^{\mathbf{H}} = [3 \quad 1], \quad \mathbf{y}_2^{\mathbf{H}} = [4 \quad 0], \quad \mathbf{y}_3^{\mathbf{H}} = [0 \quad 4], \quad \mathbf{y}_4^{\mathbf{H}} = [1 \quad 3].$$

Отже, аналізуючи одержані результати можна зробити висновки, що тестові вектори \mathbf{x}_1 та \mathbf{x}_2 більше схожі на еталонний вектор \mathbf{e}_1 , оскільки перша координата векторів $\mathbf{y}_1^{\mathbf{H}}$ та $\mathbf{y}_2^{\mathbf{H}}$ більша за другу координату. Вектори \mathbf{x}_3 та \mathbf{x}_4 більше схожі на еталонний вектор \mathbf{e}_2 . Для встановлення належності того чи іншого тестового сигналу до певного класу в мережі Хемінга використовують другий прихований шар – шар *MAXNET*.

11.3. Прихований шар MAXNET

Нейронна мережа *MAXNET* це конкурентна нейронна мережа. Її можуть застосовувати для знаходження нейрона мережі, вихід якого виявляється максимальним (знаходить нейрон переможець). Кожен нейрон у шарі *MAXNET* з'єднується сам із собою та з іншими нейронами за допомогою двонаправлених зважених зв'язків за принципом кожен із кожним.

Значення всіх вагових коефіцієнтів w_{ij}^{II} встановлюють однаковими, крім вагових коефіцієнтів самозв'язку, що дорівнює 1, тобто

$$w_{ij}^{II} = \begin{cases} 1, & \text{якщо } i = j \\ -\epsilon, & \text{якщо } i \neq j. \end{cases} \quad (11.12)$$

При цьому на параметр ϵ накладається обмеження $0 < \epsilon < N$. Матриця вагових коефіцієнтів для шару *MAXNET* набуває вигляду

$$\mathbf{w}_{ij}^{II} = \begin{bmatrix} 1 & -\epsilon & \cdots & -\epsilon \\ -\epsilon & 1 & \cdots & -\epsilon \\ \vdots & \vdots & \ddots & \vdots \\ -\epsilon & -\epsilon & \cdots & 1 \end{bmatrix}. \quad (11.13)$$

Вихідні значення з нейронного шару Хемінга спочатку пропускаються через лінійну активаційну функцію з насиченням (див. рис. 11.2): значення активності нейрона визначають рівним нулю, якщо вхідний сигнал менший від нуля; дорівнює вхідному сигналу за умови що його величина не перевищує певного порогового значення T ; та визначають рівним T , якщо величина сигналу занадто висока. Верхнє порогове значення T використовують із метою уникнення пересичення.

Принцип роботи шару *MAXNET* є таким: кожен нейрон одержує зважений сигнал від нейрона шару Хемінга, який попередньо пропущений через активаційну функцію. Потім починається ітеративна процедура:

- кожен нейрон шару одержує зважений сигнал від усіх інших нейронів і від самого себе;

- нейрон змінює свою активність з урахуванням своєї активності та активності всіх інших нейронів за такою формулою:

$$y_i^{II}(k+1) = y_i^{II}(k) - \epsilon \sum_{j \neq i}^N y_j^{II}(k); \quad (11.14)$$

- одержані результати від усіх нейронів пропускають через активаційну функцію;

- перевіряють умову припинення ітеративної процедури: коли залишиться не більше одного нейрона з ненульовою активністю.

Отже, нейрони шару *MAXNET* функціонують у режимі «переможець одержує все» (WTA – Winner Takes All). У цьому режимі в кожній фіксованій ситуації, активізується лише один нейрон, інші нейрони перебувають у стані спокою.

Для ілюстрації роботи мережі *MAXNET* розглянемо простий приклад, коли на вхід шару приходять чотири нейрони з такими значеннями активацій:

$$y_1^{II}(0) = 0.6, \quad y_2^{II}(0) = 0.2, \quad y_3^{II}(0) = 0.8, \quad y_4^{II}(0) = 0.4.$$

Фіксуємо значення $\epsilon = 0.2$ для матриці вагових коефіцієнтів одержуємо послідовність результатів роботи нейронної мережі *MAXNET*, подану в таблиці 11.1.

З одержаних результатів бачимо, що під час ітеративної процедури всі активаційні значення y_i^{II} зменшуються з ітеративним кроком k . При цьому «виживає» найбільше значення, y_3^{II} , яке зрештою

Крок, k	$y_1^{II}(k)$	$y_2^{II}(k)$	$y_3^{II}(k)$	$y_4^{II}(k)$
0	0.6	0.2	0.8	0.4
1	0.32	0	0.56	0.08
2	0.192	0	0.48	0
3	0.096	0	0.442	0
4	0.008	0	0.422	0
5	0	0	0.421	0

Таблиця 11.1 – Ітеративна процедура роботи мережі *MAXNET*

і визначає клас до якого належить тестовий вектор. Усі інші активаційні значення зменшуються до нуля й завдяки використанню порогової активаційної функції далі залишаються незмінними.

Потрібно зазначити, що під час написання програмного алгоритму для функціонування шару *MAXNET* можна замість рівняння (11.14) для ітеративної зміни активаційних значень можна використовувати матрицю вагових коефіцієнтів (11.13) «напрямую» у вигляді

$$\mathbf{y}^{\text{II}}(k + 1) = f(\mathbf{y}^{\text{II}}(k) \cdot \mathbf{w}^{\text{II}}), \quad (11.15)$$

пропускаючи результат скалярного добутку векторів \mathbf{y}^{II} та матриці \mathbf{w}^{II} через активаційну функцію.

11.4. Принцип роботи тришарової нейронної мережі Хемінга

Мережу Хемінга можуть використовувати для розпізнавання зображень, що складаються лише з чорних та білих пікселів, наприклад індекс, написаний на кодовому штампі конверта. Кодовий штамп – форма, трафарет із реперними мітками для заповнення числових даних та спрощення їх подальшого розпізнавання автоматизованими засобами. Найбільш широко цю форму застосовують у поштовому зв'язку, де використовують у вигляді індексної сітки на поштових конвертах та картках (див. рис. 11.3).



Рисунок 11.3 – Кодовий штамп у вигляді індексної сітки на поштових конвертах та картках

Схема нейронної мережі Хеммінга завдяки наявності зворотних зв'язків реалізує технологію асоціативної пам'яті. Основна ідея асоціативної пам'яті полягає в тому, щоб за окремими фрагментами даних установити відповідність вхідного образу з одним із відомих шаблонів. Технологію асоціативної пам'яті використовують для розпізнавання тексту, голосу, у системах телеметрії. Приклад роботи нейронної мережі Хеммінга наведено на рисунку 11.4.

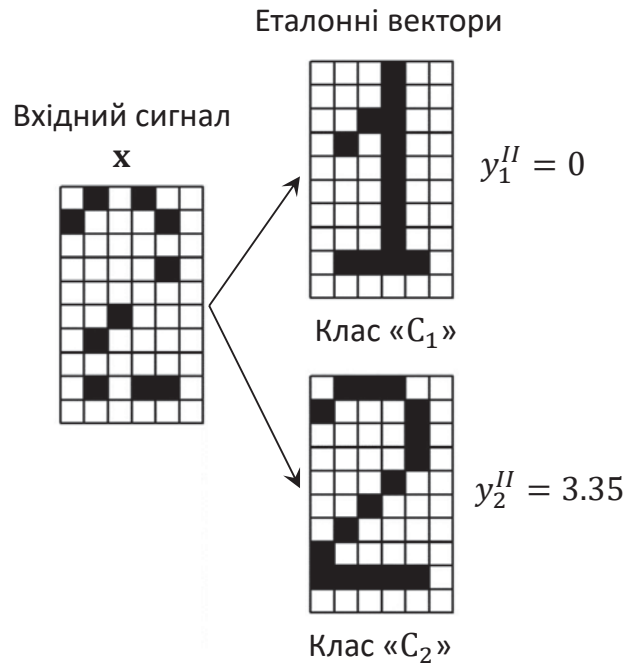


Рисунок 11.4 – Асоціативна пам'ять нейронної мережі Хеммінга в разі розпізнавання зображень цифр (клас « C_1 » – зображення цифри 1, клас « C_2 » – зображення цифри 2, y_1^{II} та y_2^{II} – вихідні сигнали нейронної мережі Хеммінга)

11.5. Застосування мережі Хемінга для класифікації цифр

За приклад роботи тришарової нейронної мережі Хемінга розглянемо задачу класифікації без учителя цифр від 0 до 9. Кроки розв'язання цієї задачі описані нижче.

1. Для цього спершу треба згенерувати еталонні вектори – відцифровані бінарні зображення кожної цифри. Приклад еталонних зображень цифр наведено на рисунку 11.5.

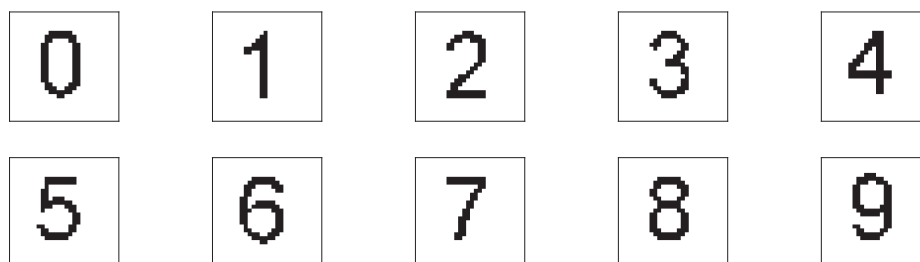


Рисунок 11.5 – Приклади еталонних зображень цифр

2. Сформувані тестові приклади цифр. Тут можна використати два підходи:

1) випадковим чином «затирати» певні чорні пікселі з певною ймовірністю;

2) генерувати шум (безлад) для кожної цифри: випадково «затирати» чорні пікселі та «розкидати» додаткові чорні пікселі.

Приклади тестових цифр для класифікації для варіанта 1) для двох різних рівнів шуму на прикладі цифр 2, 4, 6 та 8 подано на рисунку 11.6.

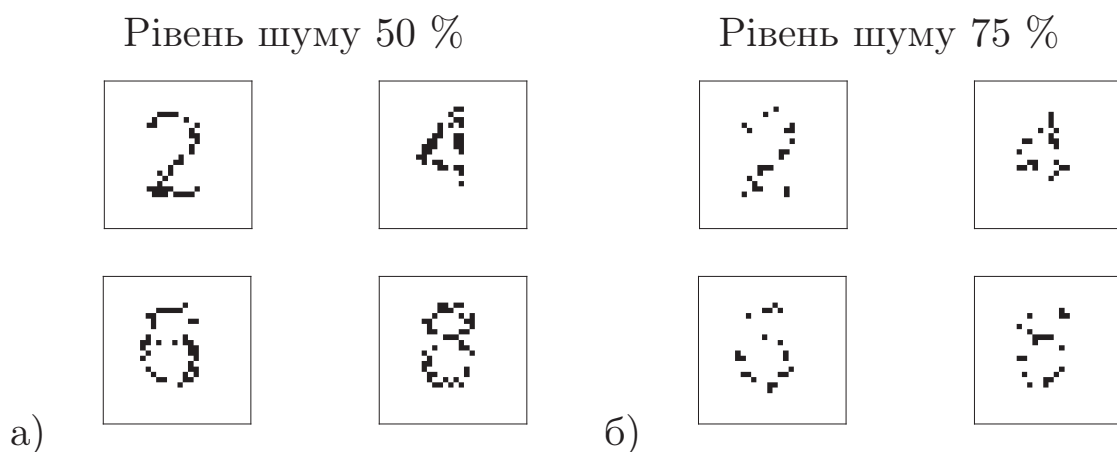


Рисунок 11.6 – Приклади тестових цифр для класифікації для варіанту «затирання» певних чорних пікселів із різним рівнем шуму на прикладі цифр «2», «4», «6», «8»

Приклади тестових цифр для класифікації для варіанта 2) для двох різних рівнів шуму на прикладі цифр 2, 4, 6 та 8 подано на рисунку 11.7.

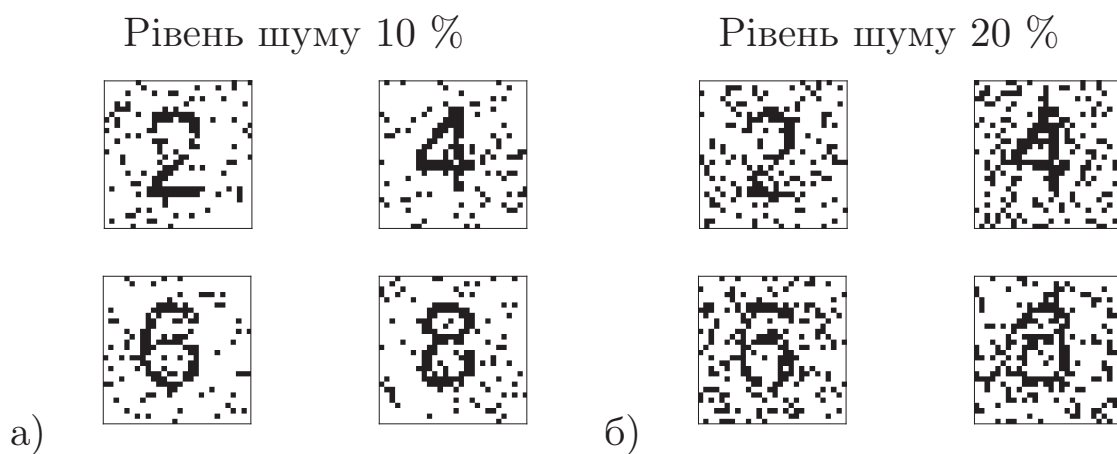


Рисунок 11.7 – Приклади тестових цифр для класифікації для генерування шуму на всьому зображенні цифри з різним рівнем шуму на прикладі цифр «2», «4», «6», «8»

3. Згенерувати цільовий вектор для тестових зображень, щоб розраховувати кількість помилок класифікації; матрицю вагових коефіцієнтів та вектор зсуву для шару Хемінга; матрицю вагових коефіцієнтів для шару *MAXNET*.

4. Змінюючи значення рівня шуму для тестових наборів порахувати залежність загальної помилки класифікації від рівня зашумленості зображення для двох випадків генерації безладу зображення. Результати розрахунків для випадку зашумлення зображення як на рисунку 11.6 подано на рисунку 11.8.

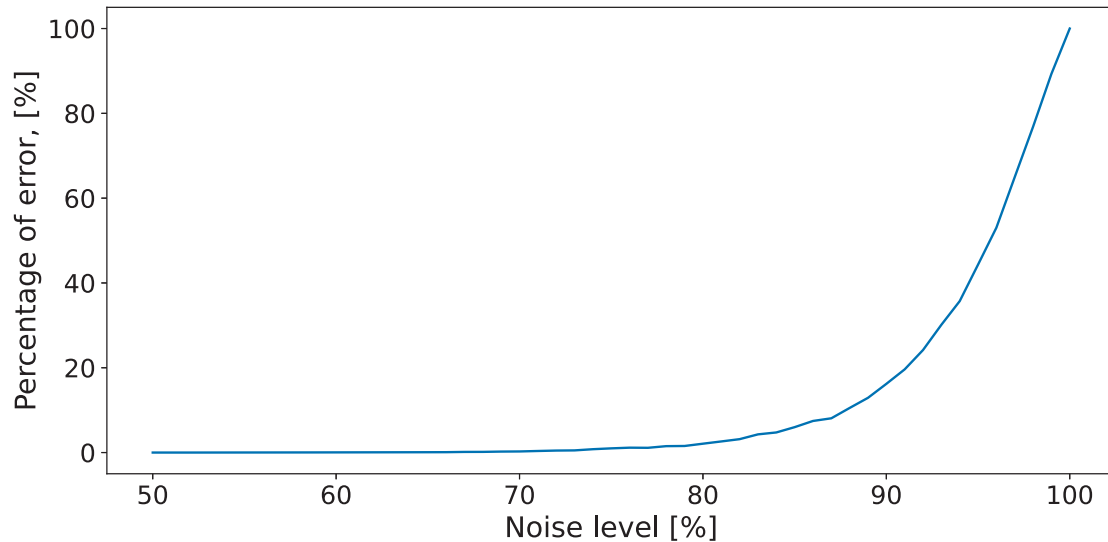


Рисунок 11.8 – Залежність загальної помилки класифікації від рівня зашумленості зображення для випадку зашумлення зображення як на рисунку 11.6

Результати розрахунків для випадку зашумлення зображення як на рисунку 11.7 подано на рисунку 11.9.

5. Для фіксованих значень рівня шуму порахувати відсоток помилок нейронної мережі на кожній цифрі.

Результати розрахунків для випадку зашумлення зображення як на рисунку 11.6 подано на рисунку 11.10.

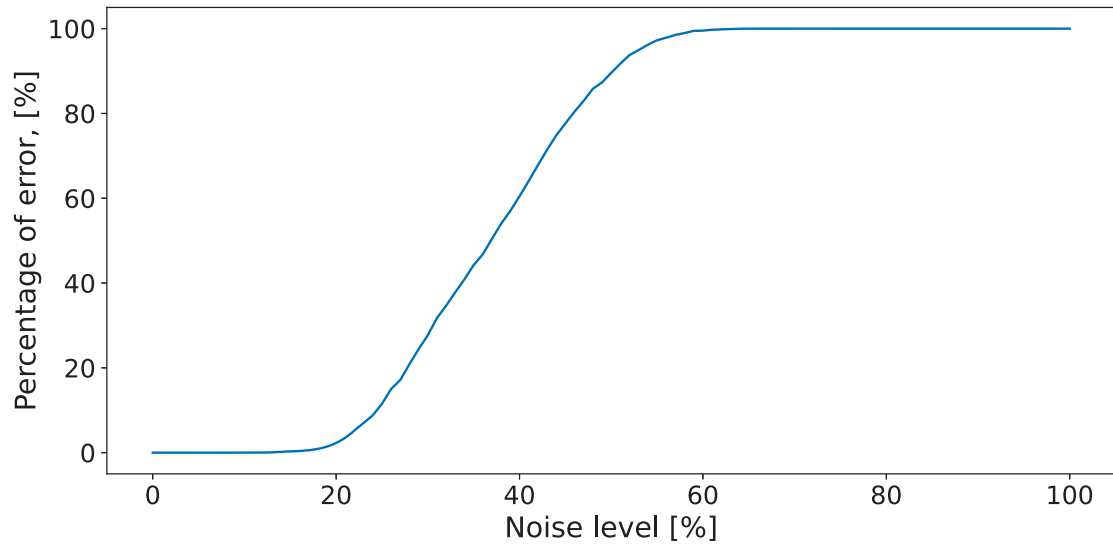


Рисунок 11.9 – Залежність загальної помилки класифікації від рівня зашумленості зображення для випадку зашумлення зображення як на рисунку 11.7

Рівень шуму 80 %

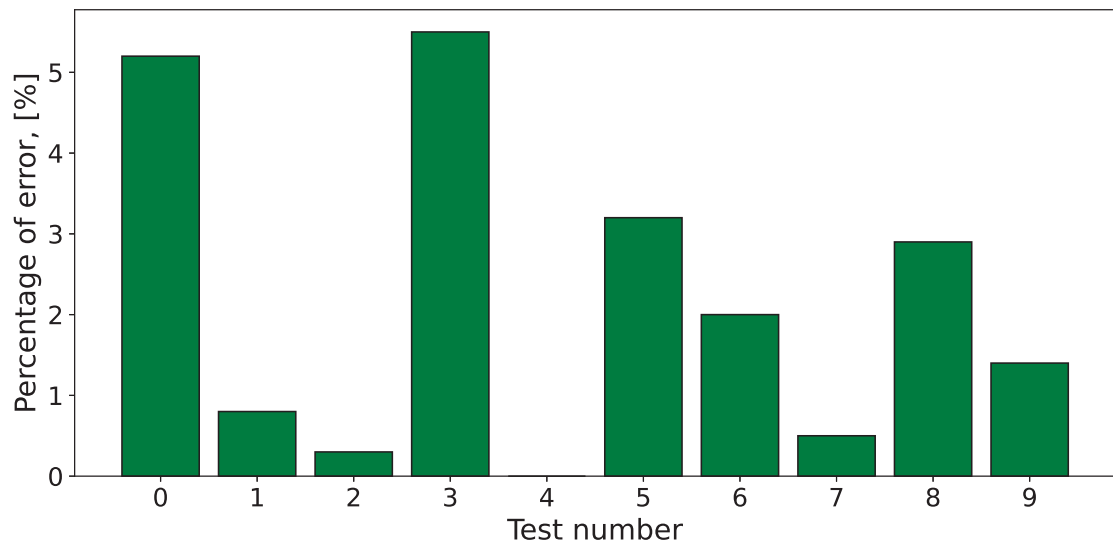


Рисунок 11.10 – Відсоток помилок нейронної мережі на кожній цифрі для випадку зашумлення зображення як на рисунку 11.6

Результати розрахунків для випадку зашумлення зображення як на рисунку 11.7 подано на рисунку 11.11.

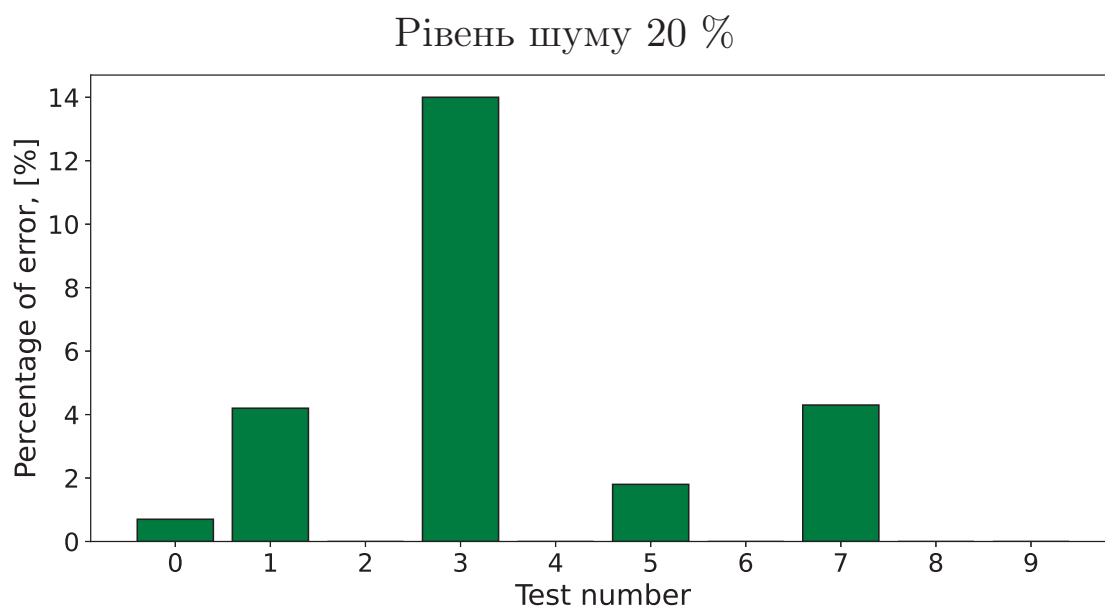


Рисунок 11.11 – Відсоток помилок нейронної мережі на кожній цифрі для випадку зашумлення зображення як на рисунку 11.7

Протестовану нейронну мережу можна перевірити на спроможність класифікувати рукописні цифри з бібліотеки `mnist`. Для цього:

- імпортуємо дані з бібліотеки `mnist`;
- переформатовуємо дані в одновимірні масиви за аналогією до еталонних зображень;
- рахуємо загальну помилку класифікації на вибірці `mnist`, яка становить $\sim 62\%$;
- розраховуємо кількість помилок для кожної цифри тестової вибірки `mnist`.

На рисунку 11.12 наведено відсоток помилок нейронної мережі на кожній цифрі для вибірки зображень рукописних цифр із бібліотеки `mnist`.

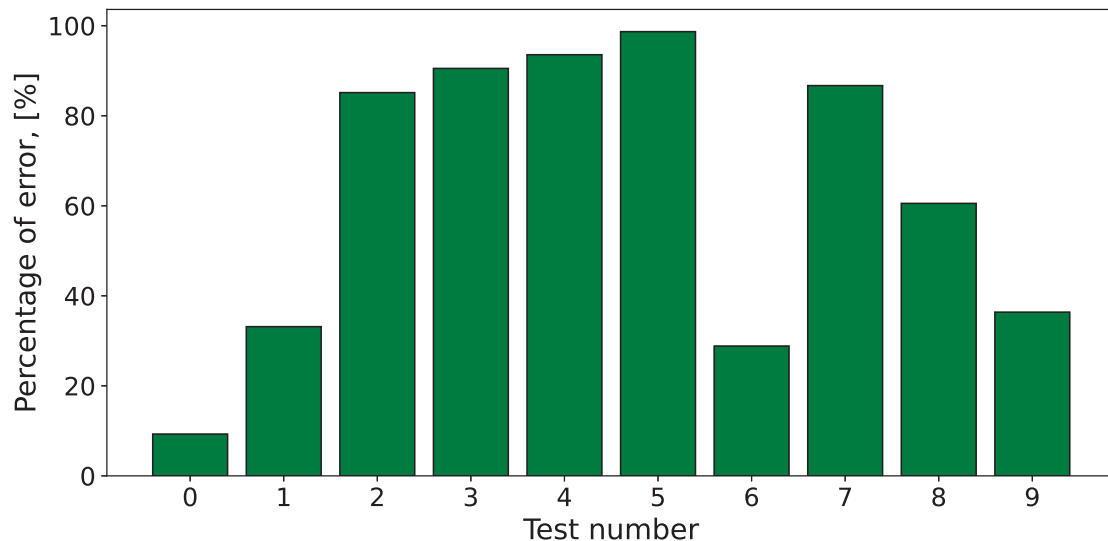
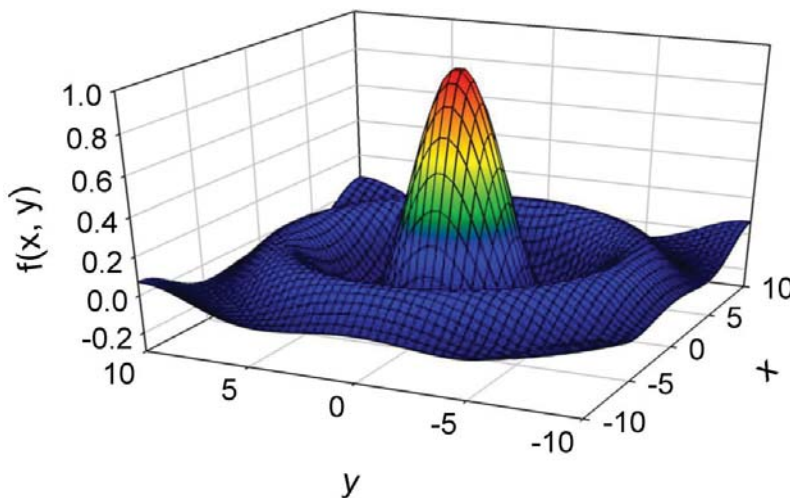


Рисунок 11.12 – Відсоток помилок нейронної мережі на кожній цифрі для вибірки зображень рукописних цифр із бібліотеки *mnist*

З одержаних результатів робимо висновок, що нейронна мережа Хемінга є непридатною для класифікації рукописних цифр, оскільки досить важко співвіднести кожну рукописну цифру з еталонним зображенням. Водночас використання різних еталонних зображень та нейронного шару *Mexican Hat* може привести до покращення результатів класифікації.

Тема 12. Нейронна мережа *Mexican Hat* для збільшення контрастності зображень

Для форматування зображень із подальшим їх використанням для класифікації за допомогою нейронної мережі Хемінга використовують додатковий шар нейронів – нейронну мережу *Mexican Hat*. Вона є першим прихованим шаром перед шаром Хемінга для переформатування вхідного сигналу у бінарний, що складається з нулів та одиниць. Мережа *Mexican Hat* дозволяє підсилити контрастність зображення тим самим виділити області, які можна визначити як інформативні (значення ознаки 1) та не інформативні (значення ознаки 0). Типовий вигляд функції *Mexican hat* наведено на рисунку 12.1.



$$z = \sqrt{x^2 + y^2}$$
$$f(x, y) = \frac{\sin(z)}{z}$$

Рисунок 12.1 – Вигляд функції *Mexican hat*

12.1. Архітектура нейронної мережі *Mexican Hat*

У нейронній мережі (шарі нейронів) *Mexican Hat* для кожного виділеного нейрона всі інші прикріплюються зі збуджувальними позитивними ваговими коефіцієнтами до нейронів, ближчих до виділеного. Подібно, нейрони також пов'язані за допомогою негативних гальмівних вагових коефіцієнтів із нейронами, що знаходяться на більшій відстані від виділеного. У деяких архітектурах певні нейрони, розміщені дуже далеко від еталонного нейрона не пов'язані з ним у сенсі вагових коефіцієнтів. Усі ці зв'язки знаходяться в межах певного рівня нейронної мережі. Нейронна мережа одержує зовнішні входні дані. Ці дані множаться з відповідними ваговими коефіцієнтами залежно від близькості та в результаті одержують значення на виході шару.

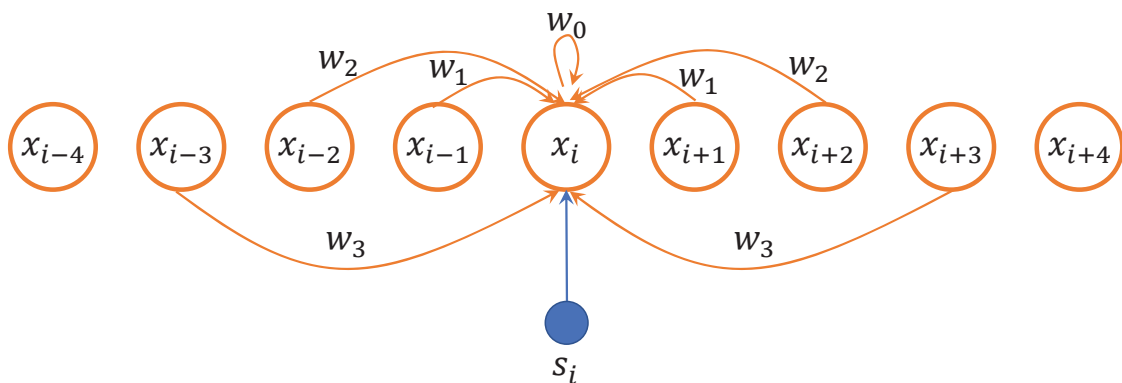


Рисунок 12.2 – Архітектура нейронної мережі *Mexican Hat*

12.2. Алгоритм функціонування мережі *Mexican Hat*

Основною ідеєю алгоритму є умовне виділення двох сфер навколо певного виділеного нейрона. Нехай R_1 буде радіусом посилення входного сигналу. Цей радіус визначає сферу найближчих сусідів навколо виділеного нейрона. R_2 – радіус сфери навколо виділеного нейрона такий, що виконується співвідношення: $R_1 < R_2$.

У такому разі:

- нейрони, що знаходяться в безпосередній близькості до виділеного нейрона (на відстані, що не перевищує радіуса R_1), з'єднані збудливими (позитивними) зв'язками й називаються «кооперативними сусідами»;

- нейрони, що знаходяться на більш віддалених позиціях (ті, що знаходять всередині сфери радіуса R_2 , але не увійшли до сфери радіуса R_1), з'єднані гальмівними (негативними) зв'язками й називаються «конкурентними сусідами»;

- нейрони, що знаходяться за межами конкурентних сусідів, не пов'язані з виділеним нейроном.

Для кожного нейрона вводять вагові коефіцієнти w_k , що змінюються залежно від близькості k -того нейрона до виділеного. Вагові коефіцієнти визначаються так:

$$\begin{cases} w_k > 0, & \text{якщо } 0 < k \leq R_1; \\ w_k < 0, & \text{якщо } R_1 < k \leq R_2; \\ w_k = 0, & \text{якщо } R_2 < k. \end{cases} \quad (12.1)$$

У нейронній мережі *Mexican Hat* використовують ітераційну процедуру й кожну ітерацію називають епохою. Відповідно, у розгляд уведено максимальну кількість ітерацій «підсилення контрасту зображення», t_{max} .

Далі будемо називати $\mathbf{X}^{(1)}$ та $\mathbf{X}^{(0)}$ векторами активацій на поточній та попередній епосі.

Визначимо кроки ітеративної процедури мережі *Mexican Hat*.

1. Ініціація параметрів нейронної мережі:

- установлення значень радіусів R_1 , R_2 і t_{max} ;
- установлення початкових значень вагових коефіцієнтів відповідно до таких правил:

$$w_k = C_1 \text{ для } k = 0, 1, 2, \dots, R_1, \text{ де } C_1 > 0;$$

$$w_k = C_2 \text{ для } k = R_1 + 1, R_1 + 2, \dots, R_2, \text{ де } C_2 < 0.$$

2. Для першої ітерації ініціювати вектор активації $\mathbf{X}^{(0)}$. Оскільки для першого й останнього нейронів також необхідно визначити

сфери радіусами R_1 та R_2 , то вектори активацій $\mathbf{X}^{(1)}$ та $\mathbf{X}^{(0)}$ матимуть розмірність $\mathbf{S} + 2R_2$. Ініціацію вектора $\mathbf{X}^{(0)}$ проводять за такою формулою:

$$X_i^{(0)} = \begin{cases} S_i, & \text{при } i = 1, \dots, M \\ 0, & \text{при } i = -R_2, \dots, 0 \text{ та } i = M + 1, \dots, R_2 \end{cases}.$$

3. Знаходження загального виходу для кожного i -того нейрона мережі за такою формулою:

$$\mathbf{X}_i^{(1)} = C_1 \sum_{k=-R_1}^{R_1} \mathbf{X}_{i+k}^{(0)} + C_2 \left(\sum_{k=-R_2}^{-R_1-1} \mathbf{X}_{i+k}^{(0)} + \sum_{k=R_1+1}^{R_2} \mathbf{X}_{i+k}^{(0)} \right).$$

4. Застосування до одержаного вихідного сигналу активаційної функції $f(z)$

$$f(z) = \begin{cases} z, & \text{якщо } z > 0 \\ 0, & \text{в іншому разі.} \end{cases}$$

та переприсвоєння векторів активацій $\mathbf{X}^{(0)} = \mathbf{X}^{(1)}$ для наступної ітерації.

5. Збільшити індекс ітерацій t та перевірити умову припинення ітеративної процедури: якщо $t < t_{max}$ – перейти на крок 3.

12.3. Застосування мережі *Mexican Hat*

Наведемо декілька прикладів використання нейронної мережі *Mexican Hat* контрастності зображення. Для цього спочатку розглянемо зображення невеликого розміру – рукописної цифри з бібліотеки `mnist`. Зафіксуємо значення глобальних параметрів: $R_1 = 1$, $R_2 = 3$, $C_1 = 0.3$ та $C_2 = -0.1$; кількість епох поставимо рівною 5, а активаційну функцію (12.2) обмежимо зверху значенням 1. Результати для декількох зображень цифр із бібліотеки `mnist` подано на рисунку 12.3: зліва – оригінальне зображення; справа – змінна зображення зі збільшенням номера епохи. Видно, що кількість



















Оригінал	Епоха 1	Епоха 2	Епоха 3	Епоха 4	Епоха 5
					
					
					

Рисунок 12.3 – Результати для декількох зображень цифр із бібліотеки `mnist`: зліва – оригінальне зображення; справа – зміна зображення зі збільшенням номера епохи

чорних пікселів, які в даних зображеннях мають інформативний характер стає більшою зі збільшенням номера епохи. Крім того, зображення стають більш чіткими, як це добре видно із зображень цифр «4» та «6». Ці зображення далі можна пропускати через певний фільтр і переводити в чорно-білий (бінарний) формат без втрат інформації.

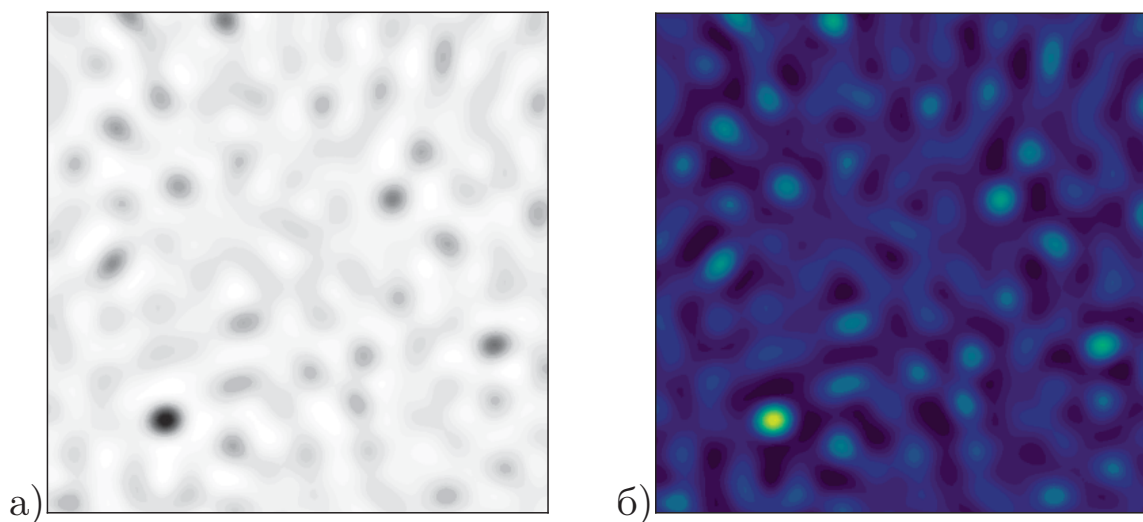


Рисунок 12.4 – Оригінальні зображення: а) у градаціях сірого кольору; б) кольорове

Далі розглянемо зображення більшого розміру, оригінали яких у градаціях сірого кольору та кольоровий наведені на рисунку 12.4. Такі зображення можуть бути одержані під медичного огляді, наприклад, флюорографічні знімки, знімки КТ або МРТ; або у фізичних чи біологічних дослідженнях, зокрема зображення одержані з мікроскопа, чи будь-які інші зображення. Метою аналізу зображень може бути, наприклад, визначення наявності та статистичних характеристик певних об'єктів: пухлин, дефектів, тощо. Саме для цього можна застосувати нейронну мережу *Mexican Hat* для збільшення контрастності цих зображень. Оскільки розмір зображень є більшим, то доцільно збільшити значення радіусів R_1 та R_2 . Покладемо $R_1 = 3$, $R_2 = 5$, $C_1 = 0.05$ та $C_2 = -0.02$ та застосуємо мережу *Mexican Hat*. Зміну зображень зі збільшенням номера епохи подано на рисунку 12.5.

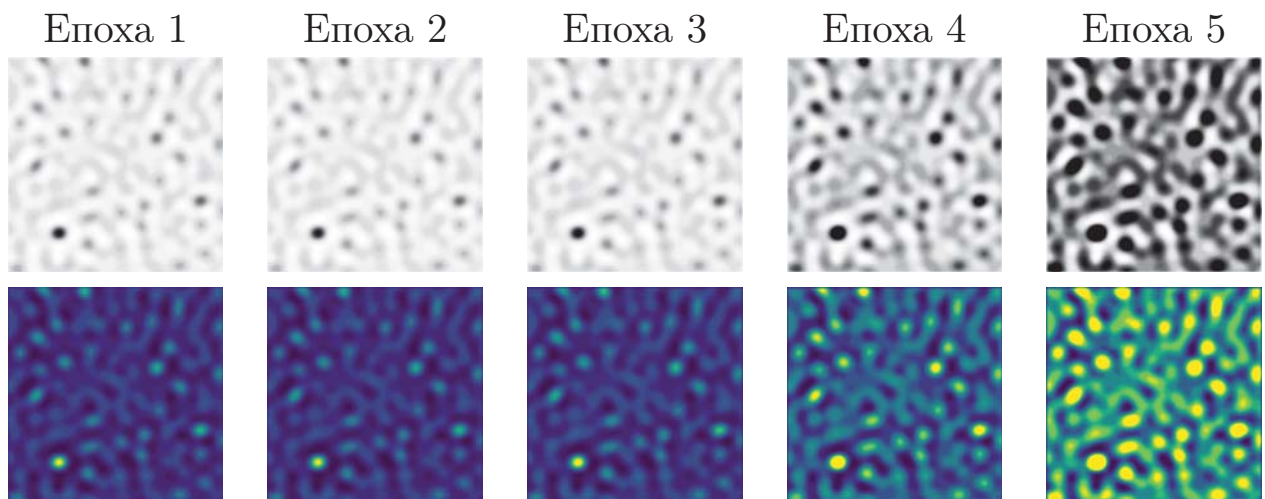


Рисунок 12.5 – Зміна зображень рисунка 12.4 зі збільшенням епохи

Видно, що нейронна мережа *Mexican Hat* успішно упоралась із задачею та виділила більш явно та чітко інформативні області на зображеннях: темні ділянки на зображеннях у градаціях сірого кольору й жовті – у кольорових зображеннях.

Тема 13. Нейронна мережа Кохонена в задачах кластеризації

Нейронні мережі Кохонена це клас нейронних мереж, основним елементом яких є шар Кохонена. Шар Кохонена складається з адаптивних лінійних суматорів (лінійних формальних нейронів). Зазвичай, вихідні сигнали шару Кохонена обробляють за правилом «Переможець отримує все»: найбільший сигнал перетворюється на одиничний, інші перетворюються на нуль.

Нейронні мережі Кохонена – типовий приклад нейромережевої архітектури, що навчається без учителя. Звідси й перелік розв’язуваних ними завдань: кластеризація даних або прогнозування властивостей. Крім того, мережі Кохонена можуть використовувати з метою зменшення розмірності даних із мінімальною втратою інформації.

У нейронних мережах Кохонена вихідні вектори в навчальній вибірці можуть бути, але можуть бути відсутніми, і, в будь-якому випадку, вони не беруть участі в процесі навчання. Тобто виходи не використовуються як орієнтири під час корекції синапсів. Саме тому цей принцип налаштування нейронної мережі називають самонавчанням.

13.1. Архітектура мережі Кохонена

Нейронну мережу Кохонена переважно використовують для вирішення завдань кластеризації, тобто об’єднання деяких об’єктів в окремі групи (кластери). Рішення про потрапляння об’єкта в той чи інший кластер приймають з урахуванням значень його ознак, які є вхідними даними цього класу нейромереж.

Наприклад, може бути завдання класифікації спортсменів за видом спорту, яким вони займаються. Ознаками можуть бути зріст, вага, час, за який спортсмен пробігає стометрівку тощо. Якщо пропустити «параметри» всіх спортсменів через мережу Кохонена,

то на виході ми одержимо певну кількість груп. Водночас повинні бути виконані такі умови:

- зразки, що належать до однієї і тієї самої групи, повинні бути подібні один одному в деякому сенсі;
- групи, подібні одна до одної, зі свого боку повинні розміщуватися близько одна від одної.

У цьому прикладі всі спортсмени, які займаються легкою атлетикою, потраплять до однієї групи, а баскетболісти – до іншої. Під час подальшого навчання мережі від групи легкоатлетів може відокремитися група бігунів. І тоді за другою з перерахованих властивостей група бігунів повинна розміщуватися близько до групи легкоатлетів і далеко від групи баскетболістів. Надалі, подаючи дані ознак спортсмена на вхід мережі, ми одержуємо результат, який полягає у визначенні того, до якої групи (кластеру) кожний спортсмен може бути віднесений.

Практичний зміст полягає в тому, що не завжди можна математичними або аналітичними методами згрупувати наявні об'єкти, нейронна мережа Кохонена дає спосіб це здійснити. Причому механізм буде незмінний, якими б не були дані.

Об'єктами можуть бути, припустимо, організації, а ознаками – показники прибутку, оборотний капітал, якісь ще дані з відкритої фінансової звітності. Нейронна мережа Кохонена може допомогти в розподіленні цих фірм на групи більш-менш благонадійних контрагентів, на основі чого вже можна приймати рішення про доцільність спільної діяльності з тією чи іншою організацією.

За структурою нейронна мережа Кохонена складається з одного вхідного шару та одного вихідного, причому всі нейрони вхідного шару пов'язані з кожним із нейронів вихідного, як показано на рисунку 13.1.

Число вхідних нейронів дорівнює кількості ознак M об'єктів, а кількість вихідних нейронів – кількості кластерів N , на які поділяються об'єкти.

Ця мережа працює за принципом «переможець отримує все», тобто максимальний вихідний сигнал трансформується в одиничний, інші виходи зануляються. Отже, подаючи ознаки деякого об'єкта на вхід, на виході ми одержуємо ряд нульових значень, і лише в

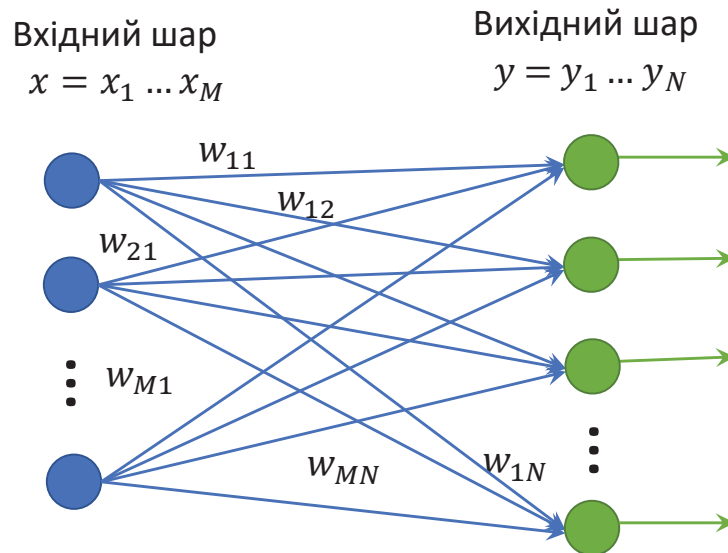


Рисунок 13.1 – Схематичне подання структури нейронної мережі Кохонена

одного нейрона значення дорівнюватиме одиниці. Це й сигналізує про те, що об'єкт віднесений до кластера.

Об'єкти подаються на вхід мережі векторами з координатами, рівними значенням ознак

$$\mathbf{x} = \{x_1, x_2, x_3, \dots, x_M\}. \quad (13.1)$$

Аналогічно вихідні елементи також можна подати у вигляді векторів, координати яких визначаються як вагові коефіцієнти зв'язків, що приходять до них від нейронів вхідного шару

$$\mathbf{y}_k = w_{1k}, w_{2k}, w_{3k}, \dots, w_{Mk} \quad (13.2)$$

Тут \mathbf{x} – об'єкт на вході, \mathbf{y}_k – k -ий вихідний нейрон, w_{ij} – ваговий коефіцієнт зв'язку між i -м вхідним та j -м вихідним нейронами. Отже, матриця вагових коефіцієнтів розміром $M \times N$ матиме такий вигляд:

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{21} & w_{31} & \cdots & w_{M1} \\ w_{12} & w_{22} & w_{32} & \cdots & w_{M2} \\ w_{13} & w_{23} & w_{33} & \cdots & w_{M3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{1N} & w_{2N} & w_{3N} & \cdots & w_{MN} \end{bmatrix}. \quad (13.3)$$

Ініціація матриці вагових коефіцієнтів – обрання координат розташування центрів кластерів відбувається випадковими числами.

13.2. Навчання нейронної мережі

Навчання мережі Кохонена полягає в налаштуванні значень вагових коефіцієнтів w_{ij} . Мережа Кохонена навчається методом послідовних наближень. У процесі навчання таких мереж на входи подаються дані, але мережа при цьому підлаштовується не під еталонне значення виходу, а під закономірності у вхідних даних. Починається навчання ініціації матриці коефіцієнтів w_{ij} .

У процесі послідовної подачі на вхід мережі прикладів із вибірки визначається найбільш схожий (найближчий) нейрон з індексом k – той нейрон, для якого скалярний добуток вектора вагових коефіцієнтів

$$\mathbf{w}_k = \{w_{1k}, w_{2k}, w_{3k}, \dots, w_{Mk}\}$$

і поданого на вхід вектора

$$\mathbf{x} = x_1, x_2, x_3, \dots, x_M$$

є мінімальним. Цей нейрон є переможцем і визначається як центр у разі підлаштовування вагових коефіцієнтів інших нейронів. Таке правило навчання передбачає «змагання» з урахуванням відстані нейронів від «нейрона-переможця».

Навчання при цьому полягає не в мінімізації помилки, а в підлаштовуванні вагових коефіцієнтів (внутрішніх параметрів нейронної мережі) для найбільшого збігу зі вхідними даними.

Основний ітераційний алгоритм Кохонена послідовно проходить ряд епох, у кожній із яких обробляється один приклад з навчальної вибірки. Вхідні сигнали послідовно пред'являються мережі, водночас бажаних вихідних сигналів не визначають. Після пред'явлення достатньої кількості вхідних векторів синаптичні ваги мережі стають здатними визначити кластери. Ваги організуються так, що близькі топологічно вузли чутливі до схожих вхідних сигналів.

Умовно всю процедуру навчання мережі Кохонена можна поділити на чотири етапи.

Етап 1

Ініціалізуємо вагові коефіцієнти випадковими значеннями. Тут необхідно зупинитися на одному нюансі, у цілому звичайному для нейромережових завдань. І полягає він у тому, що вхідні ознаки зазвичай нормалізуються так, щоб їх значення знаходилися або в діапазоні від -1 до 1 або від 0 до 1.

Залежно від цього запроваджують обмеження на початкові значення вагових коефіцієнтів:

- якщо вибірку нормалізовано в діапазоні $[-1; 1]$, то початкові значення вагових коефіцієнтів повинні задовольняти співвідношенню

$$|w_{ij}| < \frac{1}{\sqrt{M}};$$

- якщо вибірку нормалізовано в діапазоні $[0; 1]$, то початкові значення вагових коефіцієнтів повинні задовольняти співвідношенню

$$\frac{1}{2} - \frac{1}{\sqrt{M}} < w_{ij} < \frac{1}{2} + \frac{1}{\sqrt{M}}.$$

Тут M – це кількість ознак, яка й визначає кількість нейронів вхідного шару.

Фіксуємо кількість кластерів – кількість нейронів вихідного шару, N .

Етап 2

На входи мережі подаються значення навчальної вибірки, тобто ознаки одного з досліджуваних об'єктів. Для цих значень проводять розрахунок евклідових відстаней R_n від вхідного вектора \mathbf{x} до центрів кожного з N кластерів окремо за стандартною формулою

$$R_j = \sqrt{\sum_{i=1}^N (x_i - w_{ij})^2}. \quad (13.4)$$

Етап 3

Одержані на другому етапі значення R_1, R_2, \dots, R_N аналізують із метою пошуку мінімального з них. Нехай мінімальною виявилася відстань R_k від вхідного вектора до k -того нейрона. Тоді

вихідний нейрон k оголошують переможцем і починається коригування його вагових коефіцієнтів відповідно до співвідношення

$$w_{ik}(s) = w_{ik}(s-1) + \eta(s)(x_i - w_{ik}), \quad (13.5)$$

де $w_{ik}(s)$ та $w_{ik}(s-1)$ – значення вагових коефіцієнтів на поточній та попередній епохах навчання; $\eta(s)$ – швидкість навчання, яка загалом залежить від номера епохи s . Її визначають як спадну функцію. Отже, вагові коефіцієнти нейрона-переможця будуть змінюватися все менше в процесі навчання. Для швидкості навчання можна використати, наприклад зворотну пропорціональну залежність

$$\eta(s) = \frac{\eta_0}{s}, \quad (13.6)$$

де η_0 – де-яке початкове значення кроку навчання, $s_0 < 1$, або експоненційну залежність

$$\eta(s) = \eta_0 \exp\left(-\frac{s-1}{s_c}\right), \quad (13.7)$$

де s_c визначає швидкість спадання експоненти, а одиницю віднімають для забезпечення того, щоб на першій епосі значення швидкості навчання $\eta(1)$ відповідало зафіксованому початковому значенню η_0 .

Типові залежності швидкості навчання від епохи навчання подано на рисунку 13.2. Аналізуючи наведені залежності можна побачити, що зворотна пропорціональна залежність (синя крива) спадає повільніше, аніж експоненціальна залежність (помаранчева крива) при $s_c = 1$. Збільшення параметра s_c приводить до уповільнення спадання залежності швидкості навчання від епохи (порівняй помаранчеву та зелену криві).

Етап 4

Кроки 2–3 повторюються для всіх вхідних векторів. Коли всі об'єкти вибірки «пропущені» через нейронну мережу – закінчилась перша епоха. Навчання нейронної мережі проводять, поки виконається умова закінчення навчання. Для умови припинення навчання можна використовувати різні варіанти:

- час, витрачений на навчання;
- досягнення фіксованої кількості епох навчання;

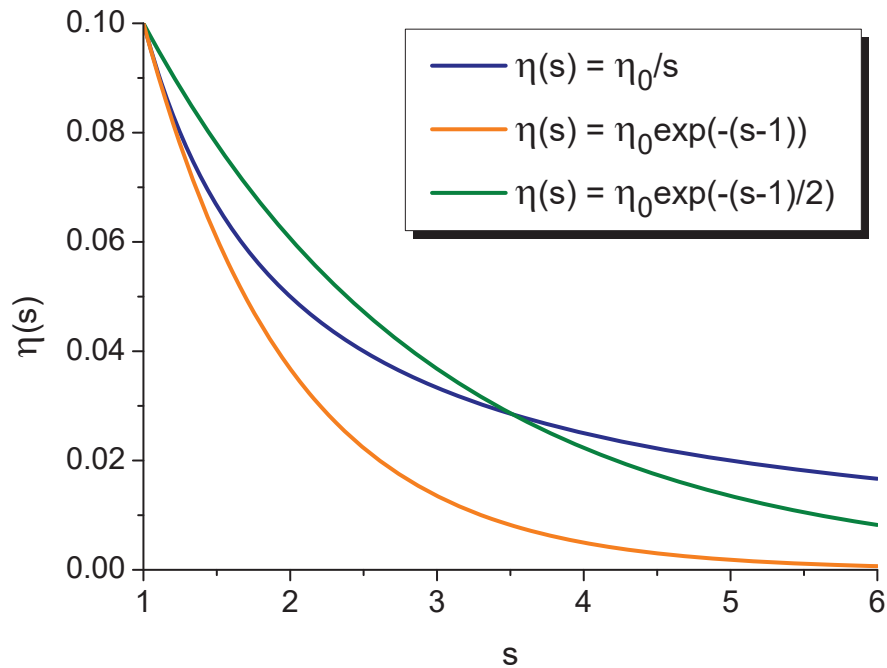


Рисунок 13.2 – Типові залежності швидкості навчання від епохи навчання для $\eta_0 = 0.1$

- значення вагових коефіцієнтів майже не змінюються.

Останній варіант дозволяє частково оптимізувати процес навчання: фіксуючи максимальне значення епох навчання, після завершення поточної можна знаходити відстань між матрицями вагових коефіцієнтів $\mathbf{W}(s)$ та $\mathbf{W}(s-1)$, порівнюючи її із заздалегідь зафіксованим значенням помилки ε . Умовою припинення навчання буде

$$[\mathbf{W}(s) - \mathbf{W}(s-1)] \leq \varepsilon. \quad (13.8)$$

Необхідно відзначити, що нейронна мережа Кохонена може навчатися, навіть якщо початкова структура мережі невідома, яким би дивним це не здавалося. У такому разі на початку навчання вихідні нейрони відсутні, але фіксоване значення максимальної кількості нейронів на вихідному шарі – максимально допустима кількість кластерів.

Під час навчання такої мережі відправною точкою є певне значення евклідової відстані R_0 , яка відповідає максимально допустимій відстані від нейрона-переможця до вхідного вектора.

На першій ітерації навчання матриця вагових коефіцієнтів порожня. Отже, у мережу додається один нейрон, координати якого w_{i1} ініціюються відповідно до процедури першого етапу навчання. На вхід мережі аналогічно подається один з елементів, тобто ознаки-координати вхідного вектора з навчальної вибірки. Також як і в описаному вище прикладі відбувається розрахунок евклідових відстаней від цього вектора до цього вихідного нейрона

$$R_1 = \sqrt{(x_i - w_{i1})^2}. \quad (13.9)$$

Якщо виконується співвідношення $R_1 < R_0$, то проводять коригування вагових коефіцієнтів w_{i1} нейрона за формулою

$$w_{ik}(s) = w_{ik}(s - 1) + \eta(s)(x_i - w_{ik}(s - 1)). \quad (13.10)$$

У протилежному разі, тобто коли $R_1 > R_0$, до нейронної мережі додається новий вихідний нейрон. Його вагові коефіцієнти задають рівними поточному вхідному вектору, який ми подали на вхід мережі. Ця процедура повторюється відповідно до етапів 2–3, поки всі приклади з навчальної вибірки не будуть проаналізовані мережею. Для цього випадку умовою припинення навчання може бути те, що кількість видних нейронів перестане змінюватись і виконується умова (13.8).

Треба зупинитися на одному моменті. Справа в тому, що в разі фіксованого значення максимально допустимої відстані від нейрона-переможця до вхідного вектора R_0 та максимальної кількості нейронів вихідного шару N , може виникнути ситуація, коли залишаться певні об'єкти навчальної вибірки, які не попади в жодний кластер. Тоді можливі два підходи залежно від поставленого завдання:

- 1) фіксувати ці об'єкти як «викиди»;
- 2) збільшити значення максимальної відстані R_0 або кількість кластерів N і провести донавчання мережі за цими об'єктами.

У першому підході ми визначимо найбільш не типові об'єкти вибірки. Такі задачі актуальні, наприклад, під час виявлення банком нетипових (підозрілих) транзакцій. Другий підхід застосовують, коли є необхідність обов'язково всі об'єкти розподілити по кластерах.

13.3. Приклад застосування нейронної мережі Кохонена

У цьому розділі наведемо приклад використання нейронної мережі Кохонена для кластеризації даних. Одержані в результаті навчання мережі кластери можуть інтерпретуватися як найбільш типові представники об'єкти свого класу. Розглянемо окремо два приклади:

- навчання мережі з фіксованою кількістю кластерів;
- мережі з невизначеною кількістю вихідних нейронів.

За приклад візьмемо набір зображень рукописних цифр із бібліотеки `mnist`: розмір навчальної вибірки становить 60 000 зображень; тестова вибірка містить 10 000 зображень. Типові приклади навчальної вибірки подано на рисунку 8.7. Репрезентативність навчальної вибірки подано на рисунку 8.8.

13.3.1. Фіксована кількість нейронів вихідного шару

Для прикладу фіксованої кількості нейронів у вихідному шарі, оберемо їх кількість рівною 10, що відповідає кількості різних цифр: від 0 до 9 у вибірці `mnist`. Кожна цифра в цій вибірці має розмір 28×28 пікселів, отже матриця вагових коефіцієнтів \mathbf{W} матиме розмір 784×10 . Віднормуємо значення ознак кожної цифри від 0 до 1 та оберемо відповідний алгоритм для ініціації вагових коефіцієнтів w_{ij} . Для функції зменшення кроку навчання оберемо експоненційну залежність (13.7) із зафіксованими $\eta_0 = 0.5$ та $s_c = 1$. За критерій зупинки ітераційного процесу оберемо такий: якщо різниця норм матриць вагових коефіцієнтів на попередній та поточній епохах

$$\Delta_{||} = \|\mathbf{W}(s)\| - \|\mathbf{W}(s - 1)\|$$

стає меншою за фіксоване число ε . Для цього розрахуємо норму ініційованої матриці W та покладемо $\varepsilon = 0.001$. Далі по черзі подаємо на вхід мережі об'єкти – зображення цифр у вигляді вектора й робимо визначені в попередньому розділі кроки:

- розраховуємо відстані R_j від поточного об'єкта до кожного нейрона з індексом $j = 0, \dots, 9$ за формулою (13.9);

- визначаємо нейрон-переможець – той нейрон з індексом k , для якого відстань R_k виявилась мінімальною;
- використовуючи правило перерахунку вагових коефіцієнтів (13.10) змінюємо значення вагових коефіцієнтів w_{ik} нейрона-переможця. Після подання всіх об'єктів із навчальної вибірки перевіряємо критерій зупинення алгоритму і, якщо він не виконується, повторюємо зазначені кроки.

Процес навчання, як залежність різниці норм матриць вагових коефіцієнтів на поточній та попередній епохах навчання $\Delta_{||}$ від епохи, подано на рисунку 13.3. Вже після першої епохи навчання значення $\Delta_{||}$ зменшується з 27.8 до 0.233, що пов'язано з великою кількістю об'єктів навчальної вибірки. Для виконання встановленого критерію достатньо було 10-ти епох навчання. У результаті навчання ми одержали 10 центрів кластерів, координати яких визначаються ваговими коефіцієнтами w_{ij} . Ці десять центрів кластерів можна асоціювати як найбільш типові представники свого класу.

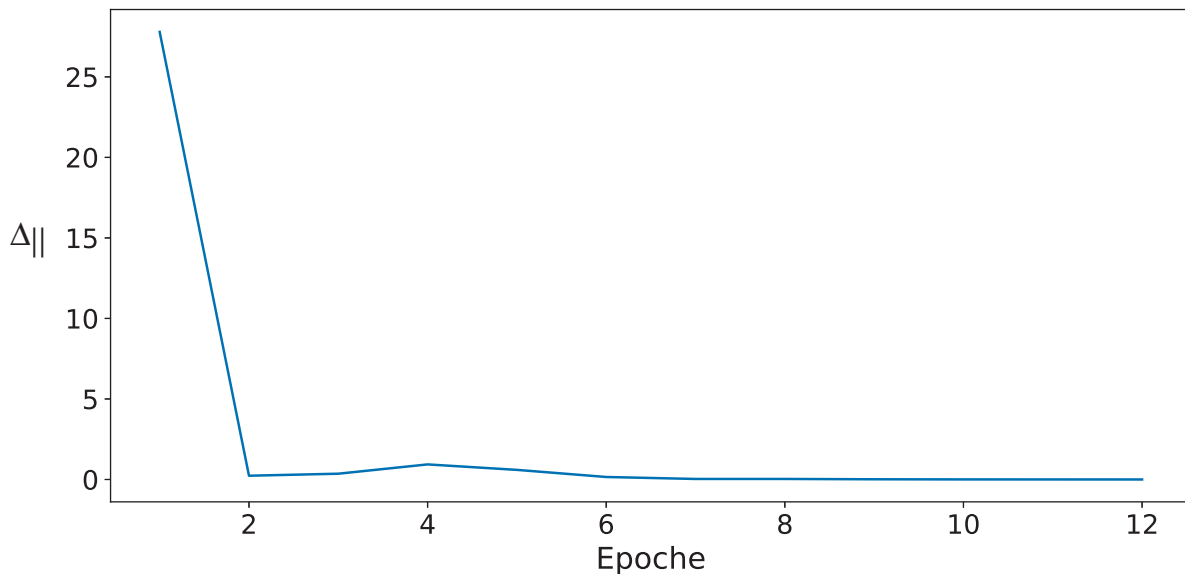


Рисунок 13.3 – Залежність різниці норм матриць вагових коефіцієнтів на поточній і попередній епохах $\Delta_{||}$ навчання від епохи

Зображення десяти центрів кластерів подано на рисунку 13.4. Тут зверху наведено номер j нейрону вихідного шару в позначенні y_j ; під ним відповідне зображення для центру j -го кластеру.

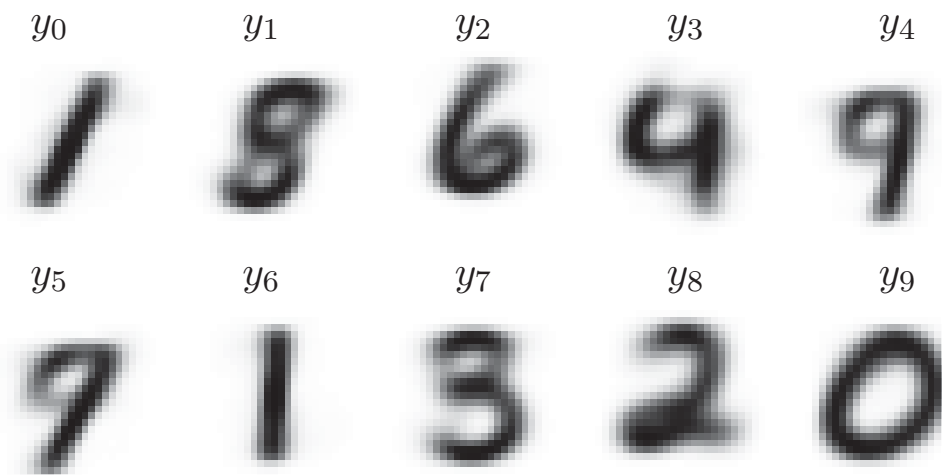


Рисунок 13.4 – Результат роботи мережі Кохонена – зображення кластерів рукописних цифр

З первинного аналізу одержаних результатів можна зробити проміжні висновки.

1. У вибірці `mnist` реалізується два типи цифри «1», для яких нейронна мережа Кохонена виділила окремі кластери: нахилена одиниця – кластер із номером 0 та пряма одиниця – кластер із індексом 6.

2. Досить однозначно виділені кластери з номерами 9, 8 та 2 для цифр «0», «2» та «6».

3. Доволі непогано виділилися кластери із цифрами «3» та «4», що репрезентовано нейронами з номерами 7 та 3. Водночас певні варіанти цифри «3» можна також розгледіти в зображенні нейрону з номером 1, а певні варіанти цифри «4» можуть бути знайдені в зображеннях нейронів із номерами 4 та 5.

4. Зображення нейрона номер 1 містить у собі перетин зображень цифр «5» та «8». Отже, можна стверджувати що зображення цих цифр перетинаються й відокремити їх дуже важко.

5. Для зображень цифри «7» також відсутні явно виражений типовий представник – центр відповідного кластера; нейрони з номерами 4 та 5 можуть містити зображення цифри «7».

Для перевірки наведених висновків перевіримо належність кожного із зображень навчальної та тестової вибірки набору `mnist` до свого кластера. Для цього будемо рахувати частоту попадання

зображень кожної з десяти цифр до різних кластерів для обох вибірок. Одержані результати для зображень цифри «0» наведено на рисунку 13.5. З рисунка 13.5а видно, що в навчальній вибірці 80 % зображень цифри «0» є найближчими в сенсі евклідової відстані між координатами-ознаками до центра свого кластера з номером 9. 10 % цифр потрапили до кластера з номером 1, де перемішані зображення цифр «5» та «8», а отже цифра «0» теж може бути схожа на зображення центру кластера 1. Імовірність приналежності цифри «0» до інших кластерів менша за 5 %. Результати для тестової вибірки наведені на рисунку 13.5б. Вони топологічно повторюють результати для навчальної вибірки з тією відмінністю, що тут імовірність попадання цифри «0» до свого кластера номер 9 становить майже 70 % і відповідно відсоток віднесення цифри «0» до інших кластерів (відсоток помилок кластеризації) є більшим, ніж на навчальній вибірці.

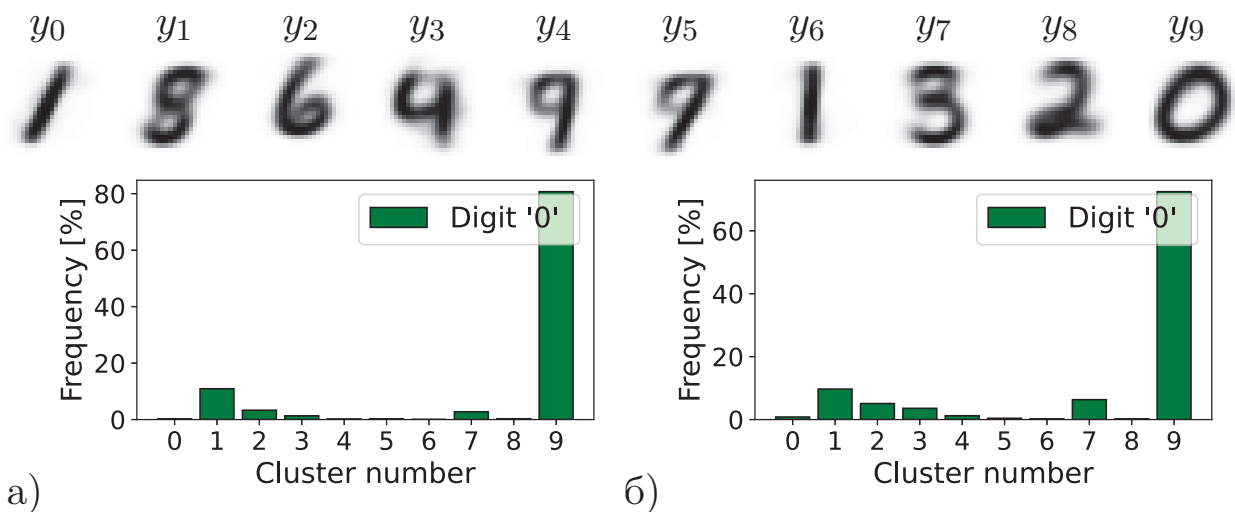


Рисунок 13.5 – Номера й зображення центрів кластерів та розподілення цифр «0» за кластерами для: а) навчальної та б) тестової вибірки

Результати для зображень цифри «1» наведено на рисунку 13.6. Бачимо, що як у навчальній, так і в тестовій вибірках близько 40 % зображень цифри «1» є представниками кластеру з номером 0, який представляє нахилені одиниці, та близько 60 % зображень належать до кластера з номером 6 – прямі одиниці.

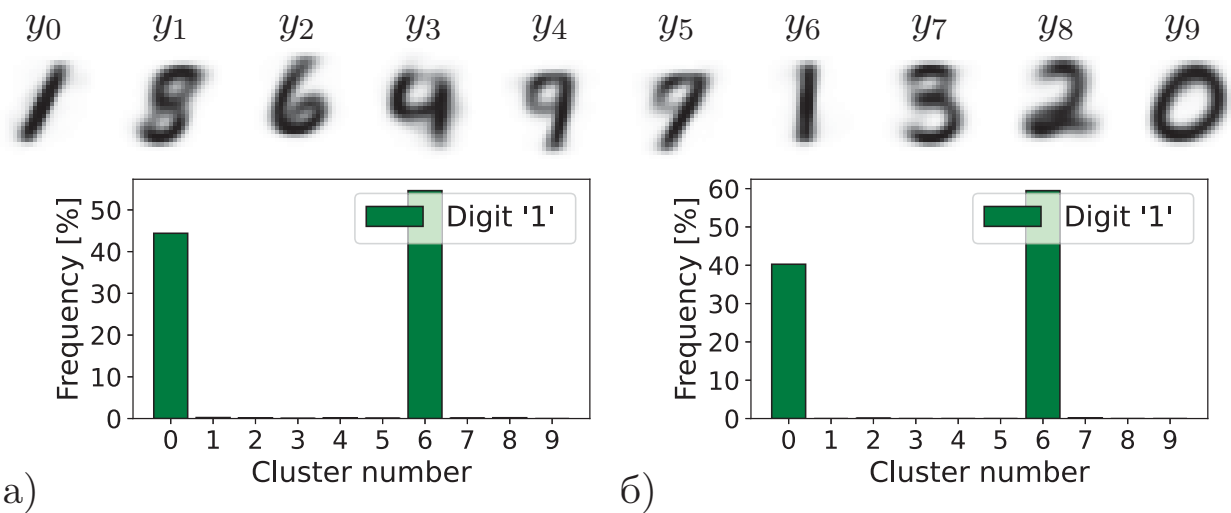


Рисунок 13.6 – Номера й зображення центрів кластерів та розподілення цифр «1» за кластерами для: а) навчальної та б) тестової вибірки

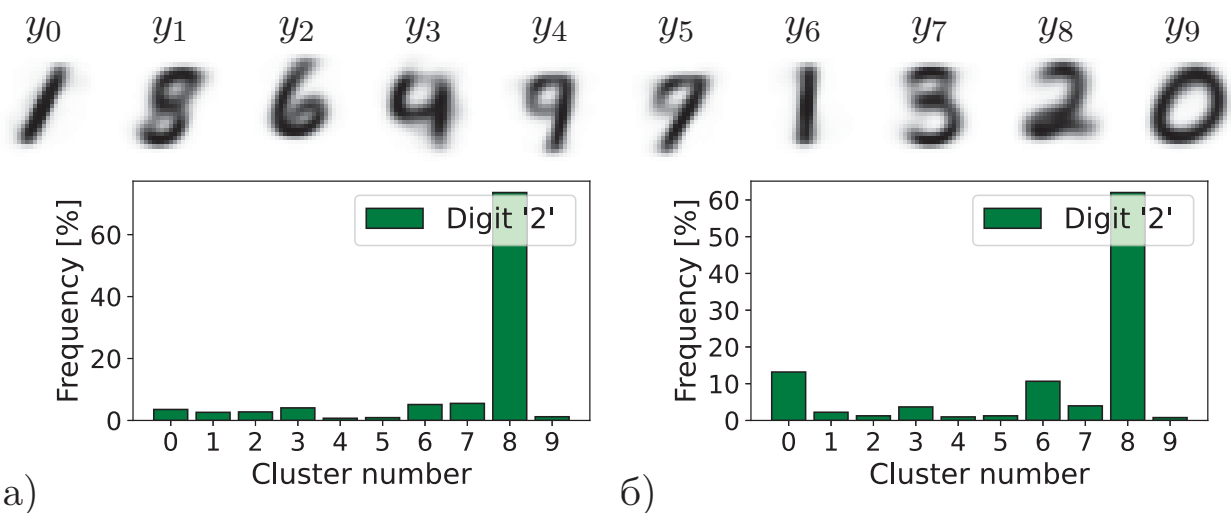


Рисунок 13.7 – Номера й зображення центрів кластерів та розподілення цифр «2» за кластерами для: а) навчальної та б) тестової вибірки

З рисунка 13.7а,б видно, що майже 70 % двійок у навчальній вибірці кластеризувалися в один кластер із номером 8, при тому, що з ненульовою імовірністю можна зустріти двійки в усіх 10 кластерах. Аналогічна ситуація спостерігається і на тестовій вибірці, де трохи більше 69 % двійок кластеризовано в один кластер.

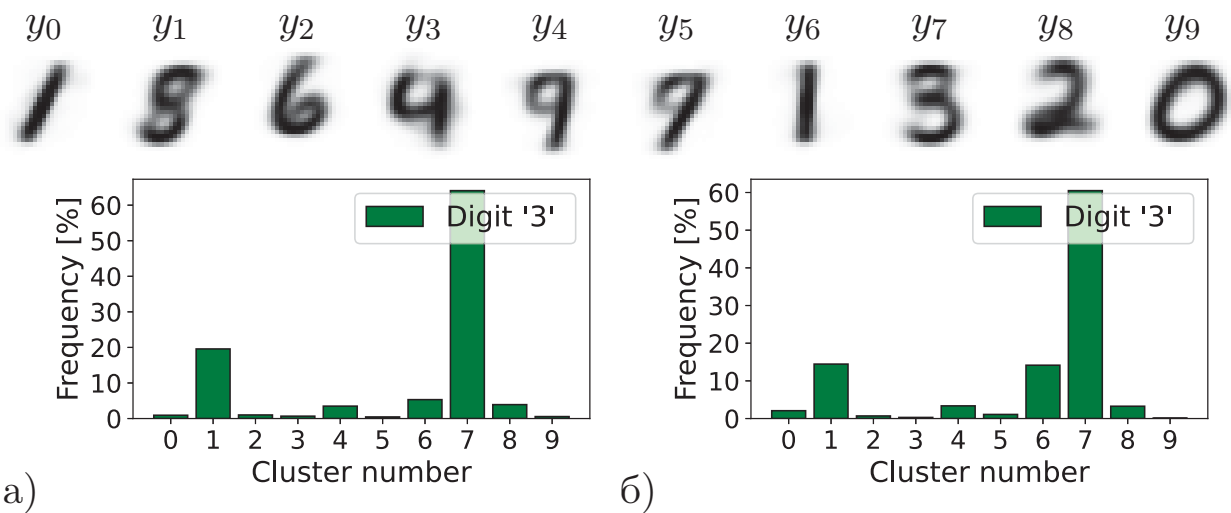


Рисунок 13.8 – Номера й зображення центрів кластерів та розподілення цифр «3» за кластерами для: а) навчальної та б) тестової вибірки

Більшість трійок в обох вибірках потрапили до свого кластеру (близько 60 %) із номером 7, як це видно з рисунка 13.8. Деяко нахилені зображення трійок (близько 20 % усіх трійок) знаходяться ближче до центру кластера з номером 1.

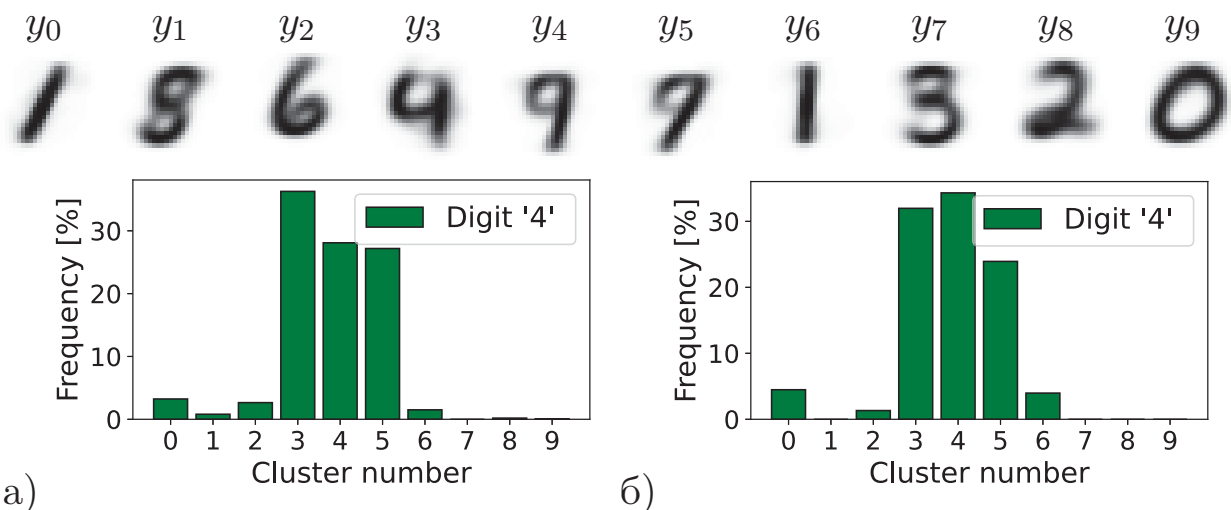


Рисунок 13.9 – Номера й зображення центрів кластерів та розподілення цифр «4» за кластерами для: а) навчальної та б) тестової вибірки

Четвірки здебільшого потрапили в три кластери з номерами 3, 4 та 5, як це видно з рисунка 13.9. Це надає змогу зробити висновок, що загалом усі зображення цифри «4» можна поділити на три різні типи.

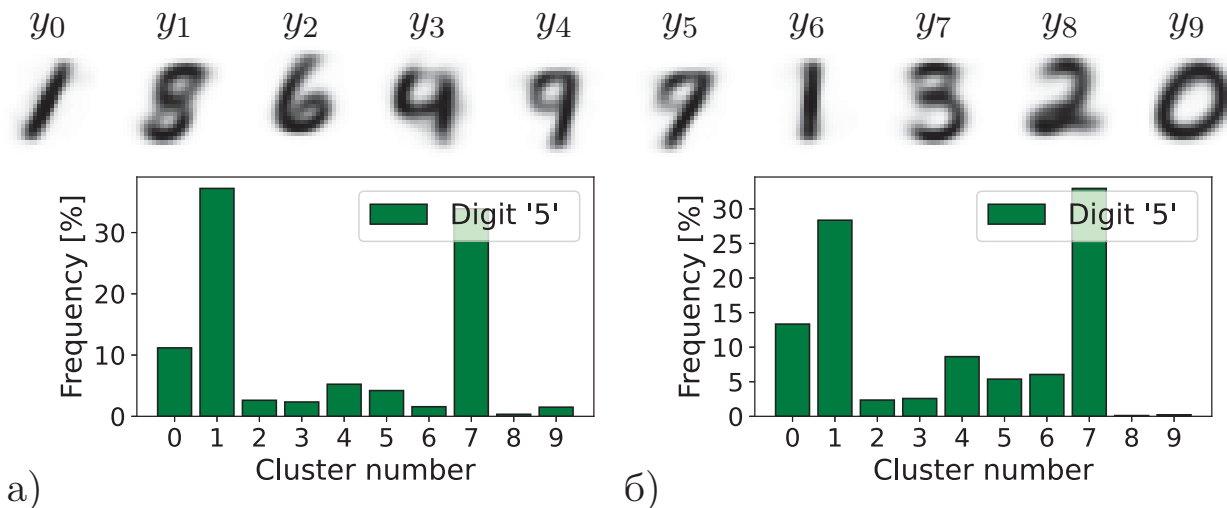


Рисунок 13.10 – Номера й зображення центрів кластерів та розподілення цифр «5» за кластерами для: а) навчальної та б) тестової вибірки

Зображення цифри «5» кластеризовано приблизно так само як і цифри «3» через подібність написання цих цифр 13.10: приблизно третина всіх трійок схожі на зображення нейронів із номерами 1 та 7 на обох вибірках.

Відповідно до результатів на рисунку 13.11 майже 70 відсотків зображення цифри «6» належать до кластера з номером 2. Водночас невелика кількість 10–15 % належать до кластера з номером 3.

Кластери з номерами 4 та 5 містять зображення прямої та нахиленої цифри «7», кількість яких приблизно однакова в навчальній та тестовій вибірках, що видно з рисунка 13.12.

Аналогічно з результатів на рисунку 13.13 можна виділити два типи зображень цифри «8», а саме – нахилені які належать до кластера з номером 1, та прямі – кластер із номером 7. Водночас нахилених зображень приблизно удвічі більше як у навчальній, так і в тестовій вибірках.

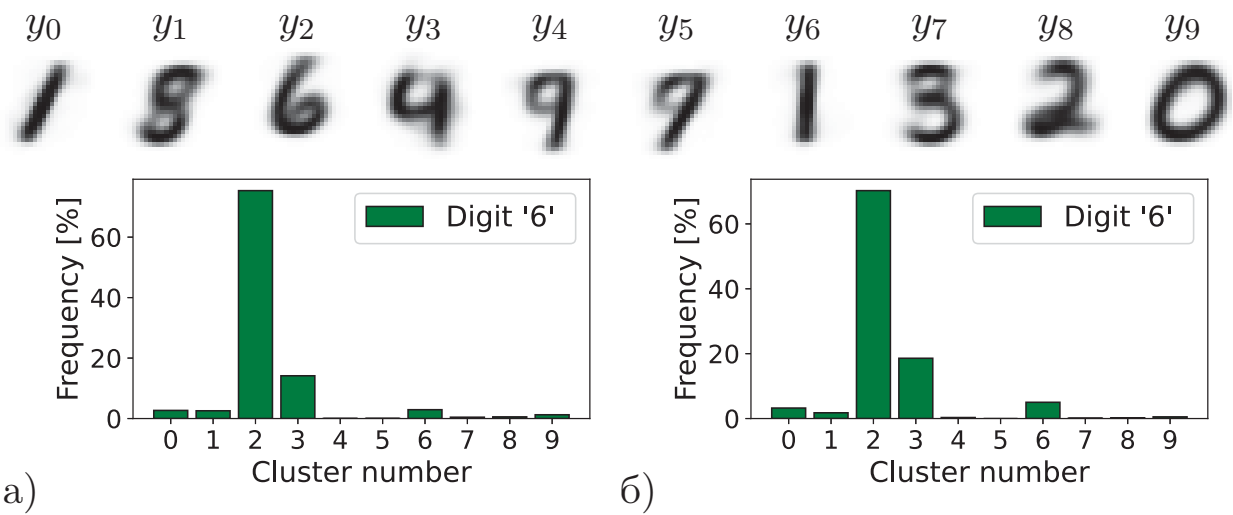


Рисунок 13.11 – Номера й зображення центрів кластерів та розподілення цифр «6» за кластерами для: а) навчальної та б) тестової вибірки

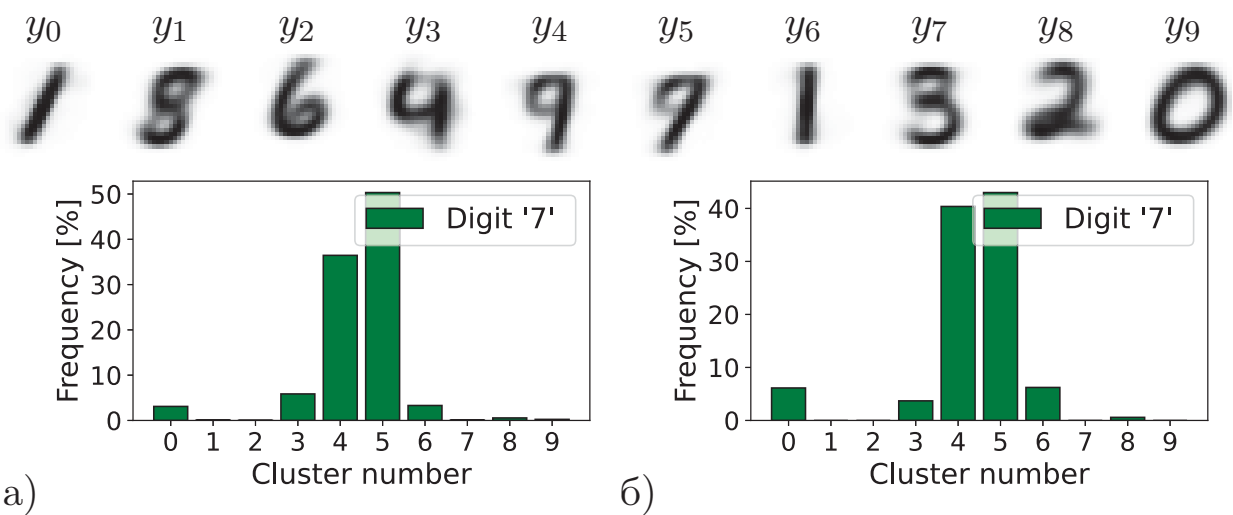


Рисунок 13.12 – Номера й зображення центрів кластерів та розподілення цифр «7» за кластерами для: а) навчальної та б) тестової вибірки

Зображення цифри «9» здебільшого знаходяться в третьому, четвертому та п'ятому кластерах, подібно до розподілення цифри «4», проте найбільше дев'яток у кластері з номером 4, як видно з рисунка 13.14.

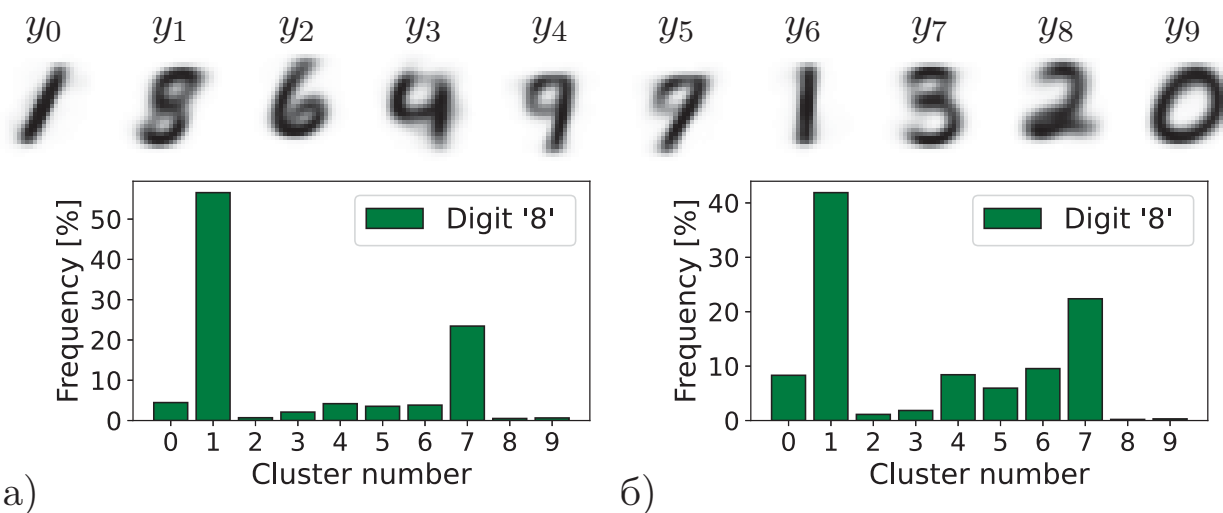


Рисунок 13.13 – Номера й зображення центрів кластерів та розподілення цифр «8» за кластерами для: а) навчальної та б) тестової вибірки

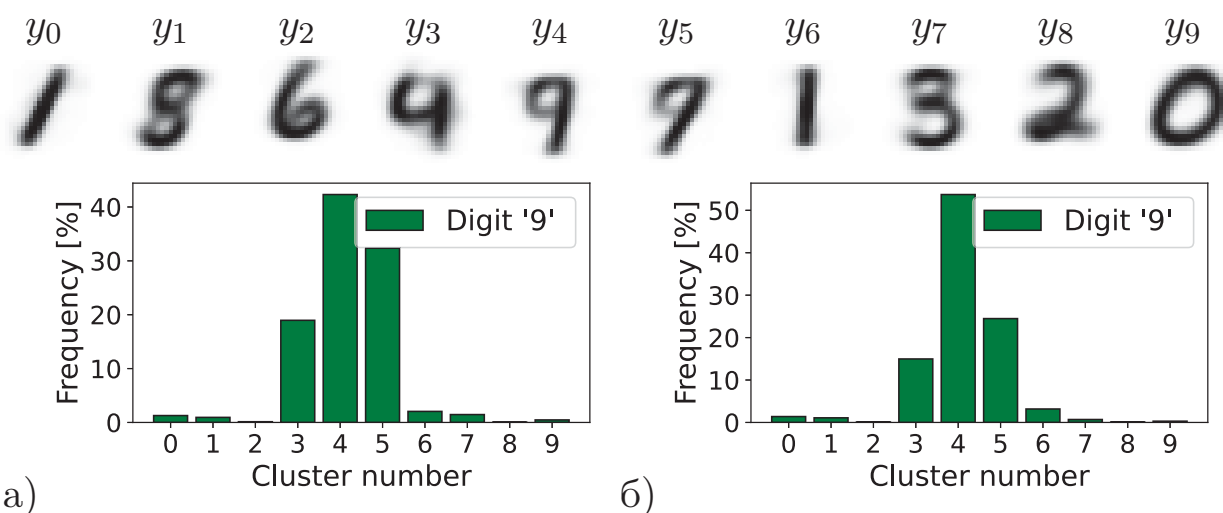


Рисунок 13.14 – Номера й зображення центрів кластерів та розподілення цифр «9» за кластерами для: а) навчальної та б) тестової вибірки

Аналізуючи одержані результати, можна зробити висновок про те, що кількість кластерів, що реалізуються в усьому наборі зображень цифр із бібліотеки *mnist* є більшою, за фіксоване значення $N = 10$. За умови використання певного значення p_c можна встановити найбільш оптимальне значення кількості кластерів, коли ча-

стота зустрічі певної цифри в кластерах є більшою за p_c . Фіксуючи $p_c = 0.2$, тобто якщо частота зустрічі цифри в кластері перевищує 0.2, за одержаними результатами можна виділити:

- один кластер для зображення цифри «0»;
- два кластера для зображення цифри «1»;
- один кластер для зображення цифри «2»;
- один кластер для зображення цифри «3»;
- три кластери для зображення цифри «4»;
- два кластери для зображення цифри «5»;
- два кластери для зображення цифри «6»;
- два кластери для зображення цифри «7»;
- два кластери для зображення цифри «8»;
- два кластери для зображення цифри «9».

Загалом можна виділити 18 кластерів для зображень цифр із бібліотеки `mnist`.

13.3.2. Нейронна мережа Кохонена з невизначеною структурою

Тепер розглянемо нейронну мережу Кохонена з невизначеною структурою, коли початкова кількість вихідних нейронів – кластерів невідома. У такому разі відповідно до алгоритму необхідно зафіксувати максимально допустиме значення відстані від об'єкта до кластерів, R_0 . Водночас кількість кластерів визначають у процесі навчання мережі й вона залежить від значення R_0 . Одержані результати залежності кількості кластерів – нейронів вихідного шару N , що реалізуються для навчальної вибірки з набору `mnist` від фіксованої максимальної відстані R_0 наведено на рисунку 13.15.

Видно, що зі збільшенням відстані R_0 кількість кластерів N монотонно зменшується та вже при $R_0 = 13$ мережа кластеризує всі зображення з навчальної вибірки як один кластер, центр якого характеризується зображенням, показаним на рисунку 13.16.

У разі меншого значення максимальної відстані між об'єктами вибірки та центрами кластерів, $R_0 = 12$ мережа виділяє три кластери зображення центрів яких подано на рисунку 13.17.

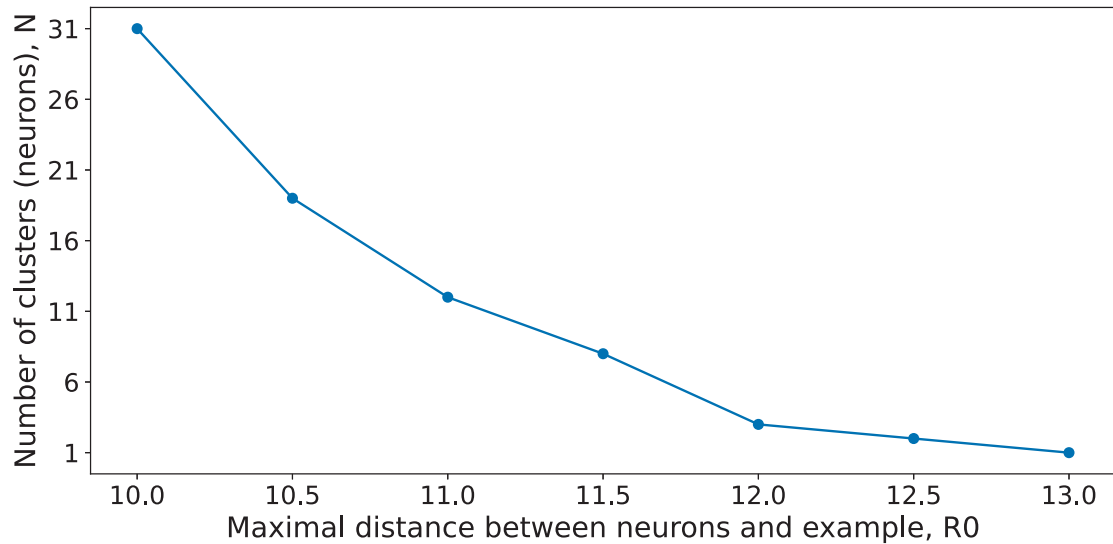


Рисунок 13.15 – Залежність кількості кластерів у нейронній мережі Кохонена від максимальної відстані від об’єкта (зображення цифри) до існуючих кластерів



Рисунок 13.16 – Зображення центрів кластерів для $R_0 = 13$



Рисунок 13.17 – Зображення центрів кластерів для $R_0 = 12$

Зображення, характерні для центрів дванадцяти кластерів при $R_0 = 11$ наведено на рисунку 13.18. Порівнюючи ці зображення із зображеннями десяти центрів, одержаними в попередньому прикладі, можна виявити топологічну схожість певних кластерів.



Рисунок 13.18 – Зображення центрів кластерів для $R_0 = 11$

У попередньому розділі ми виявили, що всі зображення рукописних цифр із бібліотеки `mnist` можна поділити на 18 кластерів. Відповідно до результатів на рисунку 13.15 зменшимо максимальну відстань R_0 до $R_0 = 10.5$ та проаналізуємо кількість кластерів та зображення їх центрів. Мережа згрупувала всі зображення цифр із навчальної вибірки в 19 кластерів, центри яких мають зображення, показані на рисунку 13.19.

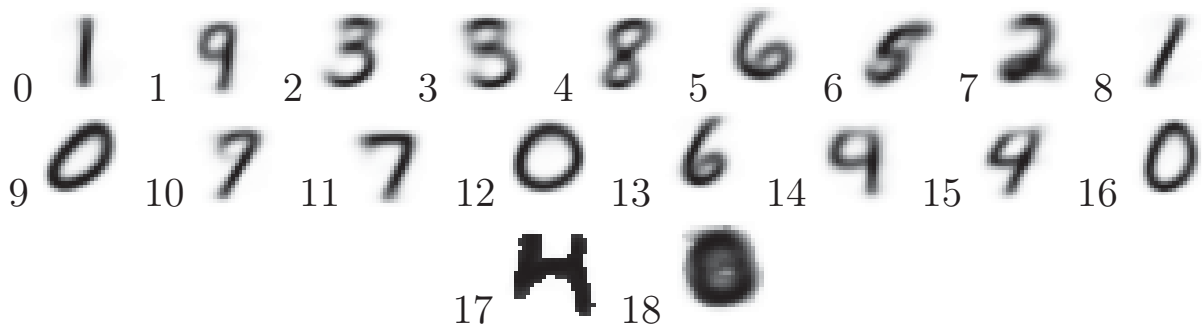


Рисунок 13.19 – Зображення центрів кластерів для $R_0 = 10.5$

Бачимо, що одержані результати щодо розподілення кількості кластерів для кожної цифри дещо різняться зі зробленими висновками в кінці попереднього розділу. Крім того, два останні зображення не містять певної інформації щодо того, зображення яких саме цифр належать до цих кластерів. Отже, проведемо навчання мережі для $R_0 = 10.5$ і зафіксуємо максимальну кількість кластерів $N = 17$. Результати навчання нейронної мережі як залежності різниці норм матриць вагових коефіцієнтів на поточній та попередній епохах навчання від епохи подано на рисунку 13.20.

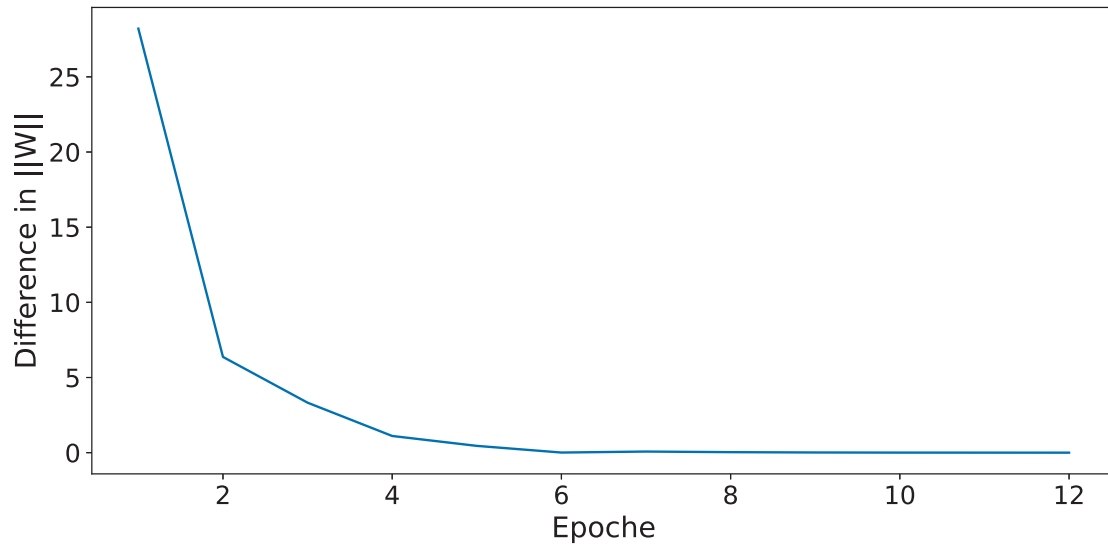


Рисунок 13.20 – Залежність різниці норм матриць вагових коефіцієнтів на поточній та попередній епохах $\Delta_{||}$ навчання від епохи

Бачимо, що в разі фіксованих параметрів навчання, а саме $\eta_0 = 0.5$ та $\varepsilon = 0.001$ та експоненційному закону (13.7) зменшення кроку навчання η достатнім виявилось 12 епох навчання. На рисунках 13.21–13.30 наведено розподілення зображень цифр від «0» до «9» із тестової вибірки.

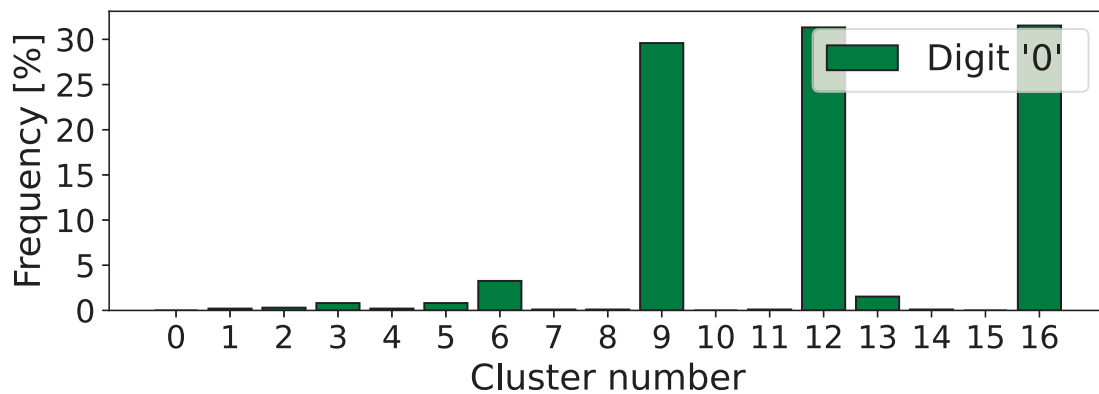


Рисунок 13.21 – Розподілення цифри «0» за кластерами для тестової вибірки

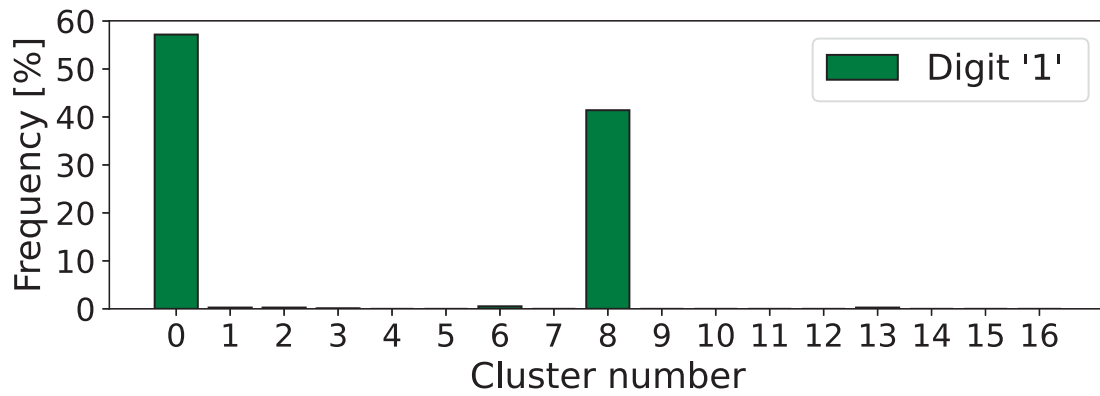


Рисунок 13.22 – Розподілення цифри «1» за кластерами для тестової вибірки

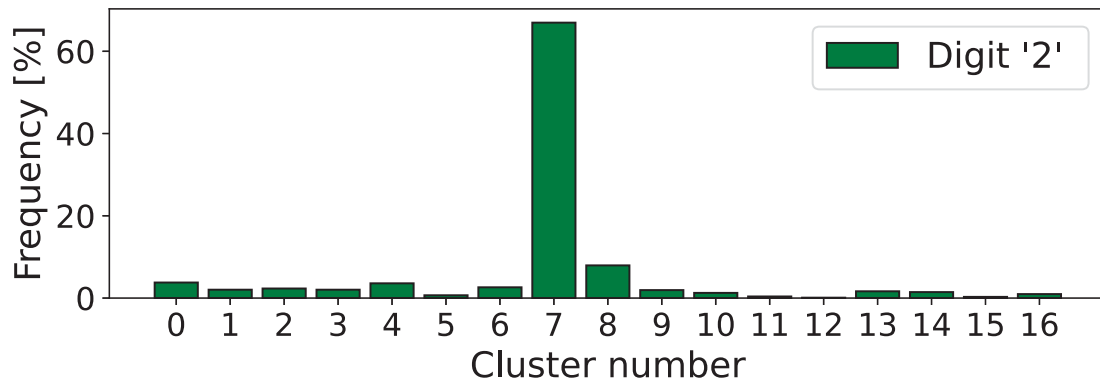


Рисунок 13.23 – Розподілення цифри «2» за кластерами для тестової вибірки

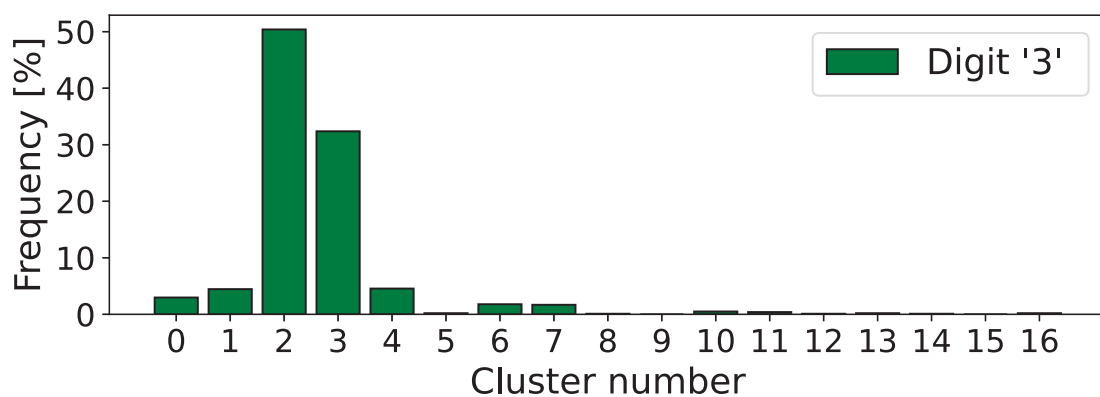


Рисунок 13.24 – Розподілення цифри «3» за кластерами для тестової вибірки

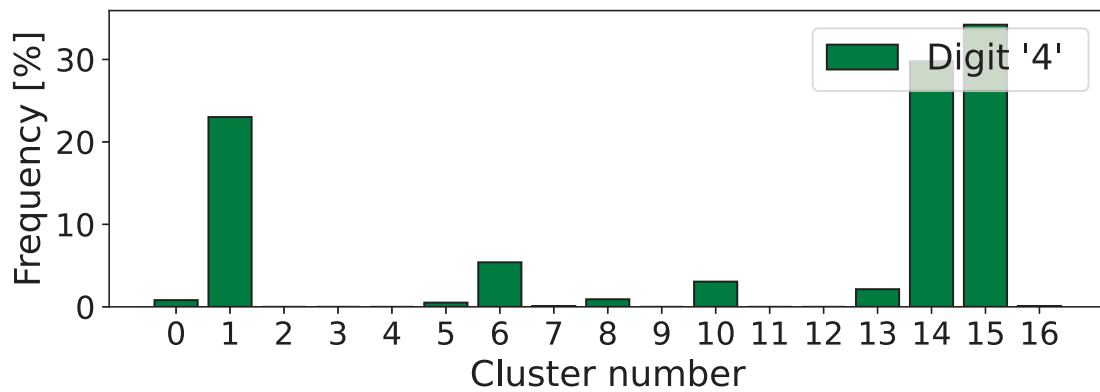


Рисунок 13.25 – Розподілення цифри «4» за кластерами для тестової вибірки

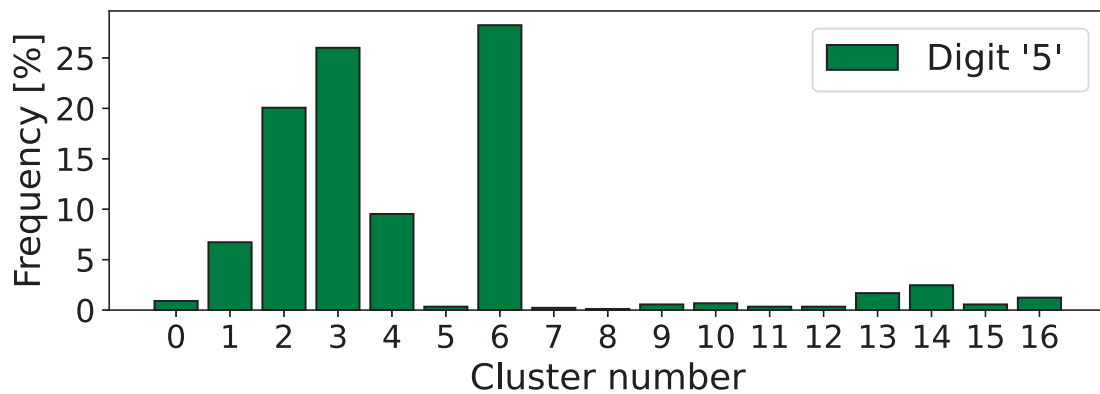


Рисунок 13.26 – Розподілення цифри «5» за кластерами для тестової вибірки

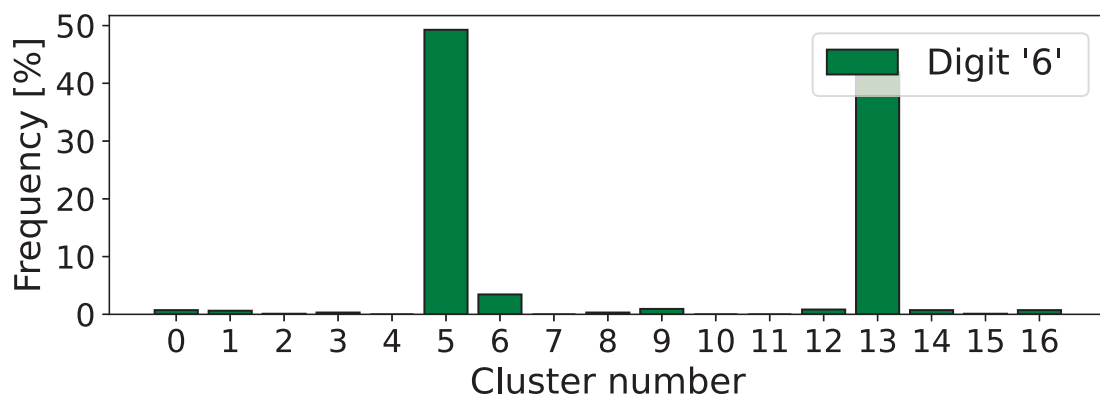


Рисунок 13.27 – Розподілення цифри «6» за кластерами для тестової вибірки

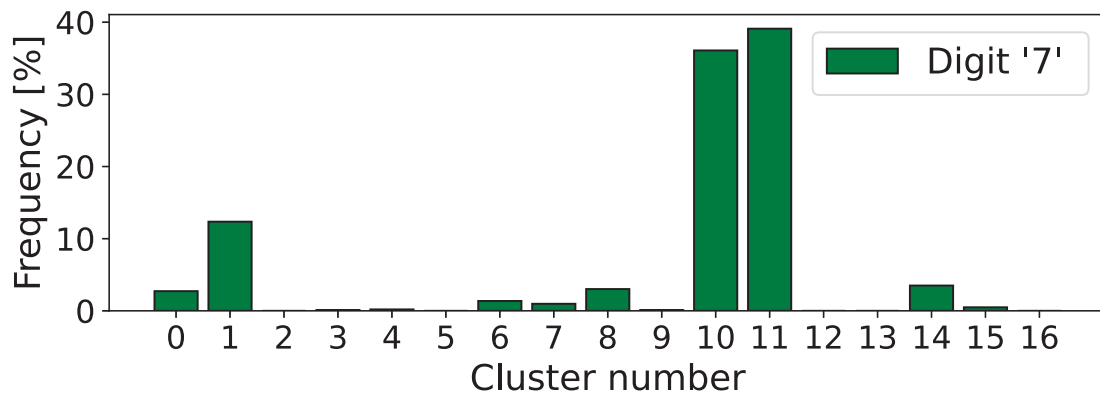


Рисунок 13.28 – Розподілення цифри «7» за кластерами для тестової вибірки

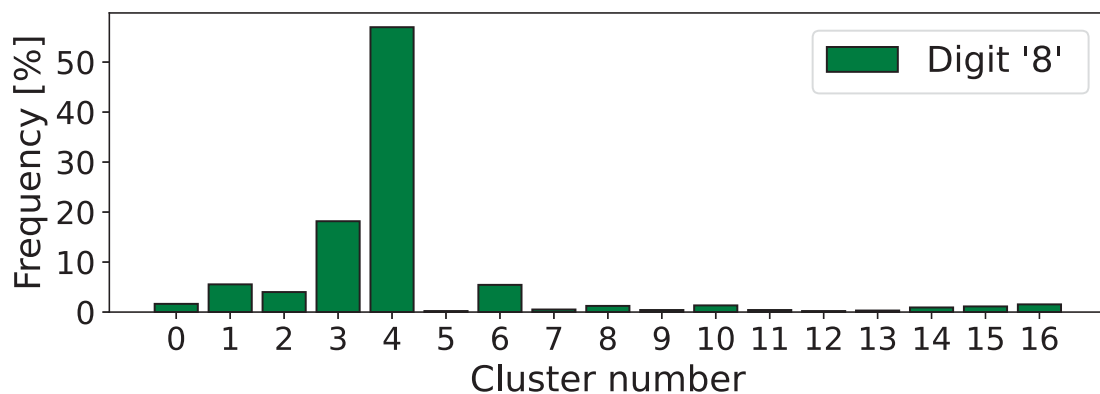


Рисунок 13.29 – Розподілення цифри «8» за кластерами для тестової вибірки

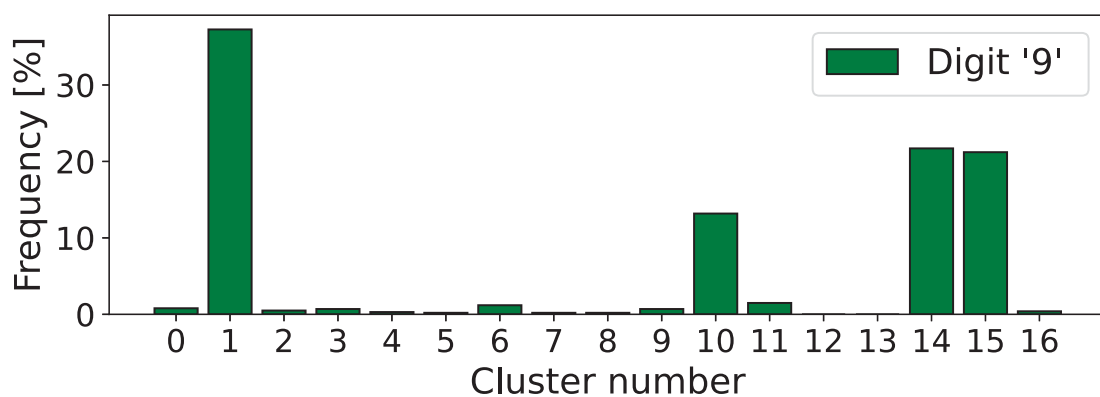


Рисунок 13.30 – Розподілення цифри «9» за кластерами для тестової вибірки

З одержаних результатів можна зробити такі висновки:

- зображення цифри «0» розподілені майже рівномірно по своїх трьох кластерах на відміну від випадку з фіксованою кількістю $N = 10$ кластерів;
- зображення цифри «1» розподілені аналогічно до прикладу з фіксованою кількістю $N = 10$ кластерів;
- більшість зображень двійок, як і в попередньому прикладі, зосереджені в одному кластері;
- трійки, прямі та нахилені, здебільшого розподілені за двома кластерами;
- четвірки, як і раніше, мають три різні типи;
- п'ятірки також розподілено за трьома кластерами;
- усі зображення цифри «6» поділені здебільшого на два типи;
- для зображень цифри «7» реалізується два типи кластерів;
- виділився явний кластер із зображеннями цифри «8»;
- для дев'яток також виділився один найбільш імовірний кластер.

Отже, під час використання мережі Кохонена з невизначеною кількістю кластерів для кластеризації даних спочатку треба, змінюючи значення максимально допустимої відстані між об'єктами та центрами кластерів, проаналізувати центри кластерів і потім зафіксувати максимально допустиму кількість кластерів навчити нейронну мережу.

13.4. Карты Кохонена, що самоорганізуються

Карта Кохонена, що самоорганізується (SOM, Self-Organizing Map) – це одна з версій нейронних мереж Кохонен. Ключовий момент полягає в тому, що використання карт Кохонена дозволяє відобразити результати своєї роботи у вигляді зручних для розуміння карт (найчастіше двовимірних). І вже на основі наочної графічної інтерпретації можна проводити аналіз одержаних даних, пошук залежностей, прогнозування тощо.

Отже, карти Кохонена дозволяють перетворити вихідний N -вимірний простір на простір меншої розмірності, із яким вже набагато простіше працювати.

13.4.1. Структура карти Кохонена

Вихідні нейрони карти Кохонена найчастіше зображують розташованими у вигляді двовимірної сітки, як показано на рисунку 13.31. Водночас кожен з цих нейронів, як і раніше, характеризується значеннями своїх вагових коефіцієнтів.

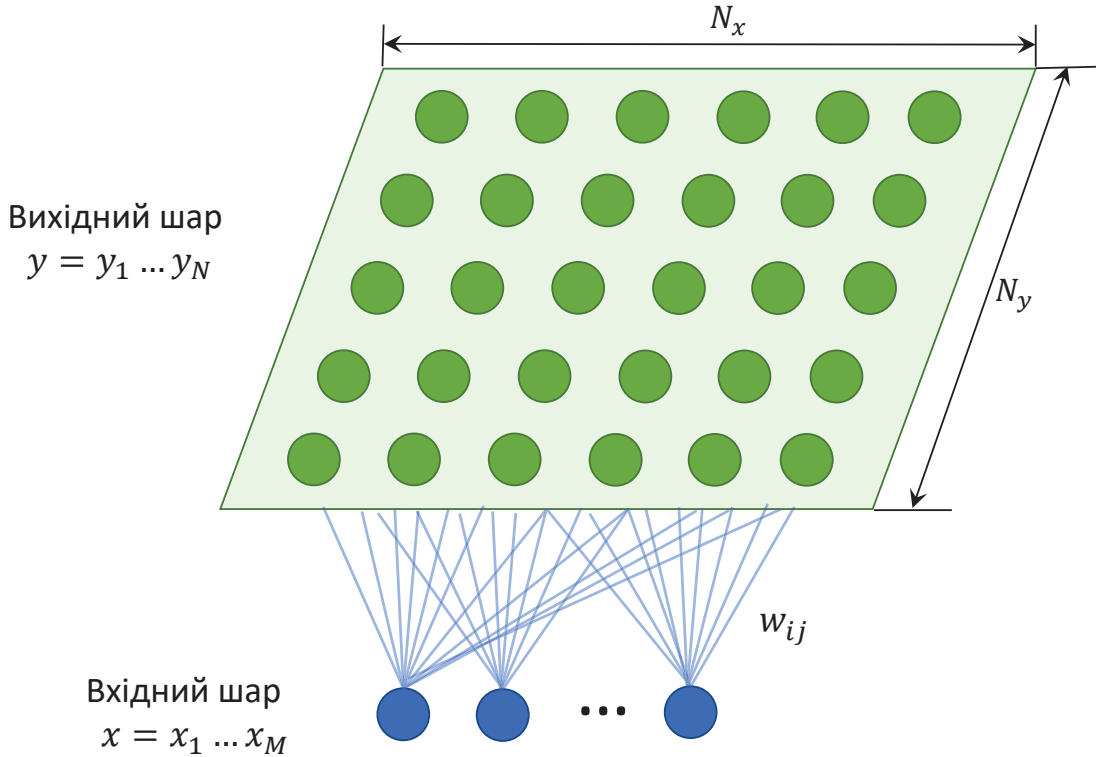


Рисунок 13.31 – Схематичне подання карти Кохонена

Отже суть і логіка залишаються такими самими як і для звичайних мереж Кохонена з тією відмінністю, що вихідні нейрони впорядковано розміщені, що дозволяє наочно побачити результати роботи мережі. Очевидно, що N -вимірний простір, елементами якого, по суті, і є нейрони вихідного шару з координатами

$$\{w_{1k}, w_{2k}, w_{3k}, \dots, w_{MN}\},$$

не відображається на екрані або папері. У тому разі, коли нейрони умовно розташовують у вигляді двовимірної сітки, її легко можна представити графічно.

Процес навчання протікає так само, як і для загального випадку нейронних мереж Кохонена, тобто за тими кроками, які ви-

кладено в попередньому розділі, але є одна важлива відмінність. Полягає вона в тому, що оновлення значень вагових коефіцієнтів здійснюється не лише для нейрона-переможця з індексом k , а й для деяких сусідніх нейронів. Яких саме сусідів – визначається функцією сусідства $h_{kj}(s)$, що визначає міру сусідства вихідних нейронів. Найчастіше застосовують варіант, коли ця функція репрезентує функцію Гауса

$$h_{kj}(s) = \exp\left(-\frac{d_{kj}^2}{2\sigma^2(s)}\right), \quad (13.11)$$

де

- k, j – номери нейрона-переможця та іншого нейрона;
- s – поточна епоха процесу навчання;
- $\sigma(s)$ – навчальний множник, значення якого зменшується зі збільшенням номера епохи навчання s ;
- d_{kj} – евклідова відстань між нейронами k і j .

Типові залежності функції сусідства h_{kj} між нейроном переможцем з індексом k та іншими нейронами з індексами j вихідного шару наведено на рисунку 13.32.

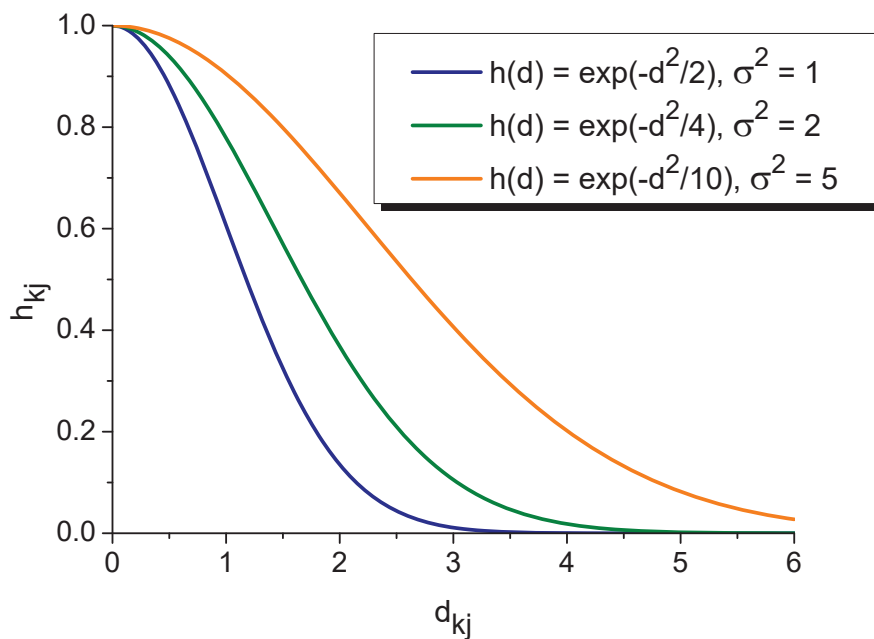


Рисунок 13.32 – Типові залежності функції сусідства від відстані між нейронами для різних значень множника σ

Видно, що функція сусідства спадає зі збільшенням відстані між нейронами d_{kj} . Крім того, збільшення значення множника σ приводить до того, що функція сусідства спадає більш плавно. Оскільки множник $\sigma(s)$ зменшується з номером епохи s , то для подання цього ефекту можна використати експоненціальну функціональну залежність, аналогічну тій, що використовувалася для залежності швидкості навчання (13.7), а саме

$$\sigma(s) = \sigma_0 \exp\left(-\frac{s-1}{s_c}\right), \quad (13.12)$$

де s_c визначає швидкість спадання експоненти, а одиницю віднімають для забезпечення того, щоб на першій епосі значення множника $\sigma(1)$ відповідало зафіксованому початковому значенню σ_0 . У такому разі залежність функції близькості від відстані між нейронами d та номера епохи навчання s набуває вигляду, як показано на рисунку 13.33.

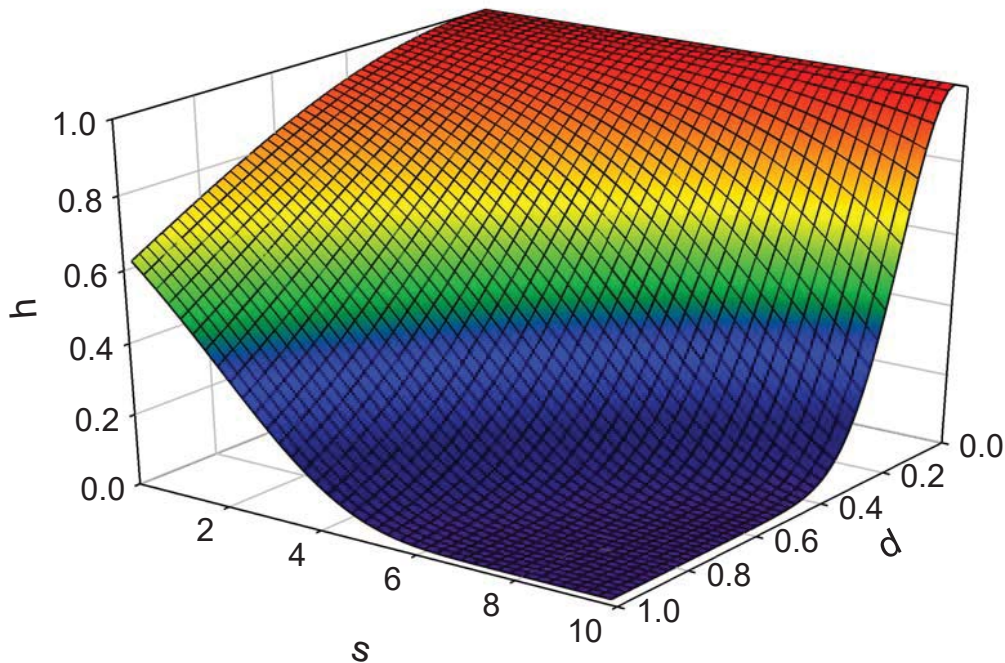


Рисунок 13.33 – Типові залежності функції сусідства від відстані між нейронами та епохи навчання

Видно, що функція сусідства спадає як зі збільшенням відстані між нейронами, так і зі збільшенням номера епохи навчання. З використанням уведеної функції сусідства (13.11) оновлення вагових коефіцієнтів w_{ij} відбувається за формулою

$$w_{ij}(s) = w_{ij}(s - 1) + \eta(s)h_{kj}(s) (x_i - w_{ij}(s - 1)). \quad (13.13)$$

Зауважимо, що оновлюються не лише вагові коефіцієнти w_{ik} для k -го нейрона-переможця, але й вагові коефіцієнти всіх інших вихідних нейронів w_{ij} . Крім того, формула оновлення вагових коефіцієнтів (13.13) відрізняється від попередньої формули для стандартної мережі Кохонена (13.5) тим, що тут перерахунок вагових коефіцієнтів здійснюється з урахуванням функції сусідства, яка й визначає, якою мірою буде скориговано вагові коефіцієнти того чи іншого нейрона. Якщо нейрон з індексом j сильно віддалений від нейрона-переможця з індексом k , то значення $h_{kj}(s)$ теж буде малим.

Розглянемо функцію сусідства безпосередньо для нейрона-переможця, тобто прийmemo j рівним k . Одержуємо

$$h_{kk}(s) = \exp\left(-\frac{d_{kk}^2}{2\sigma^2(s)}\right) = [d_{kk} = 0] = \exp\left(-\frac{d0}{2\sigma^2(s)}\right) = 1, \quad (13.14)$$

де враховано, що відстань від k -го нейрона до нього самого дорівнює 0. Тоді для оновлення вагових коефіцієнтів формула (13.13) перетворюється на формулу (13.5). Отже, відмінність процесу навчання карти Кохонена полягає саме в тому, що оновлюються значення не лише для нейрона-переможця, а й для його сусідів. Механізм по суті незмінний:

1) подаємо на вхід дані одного з об'єктів, визначаємо нейрон-переможець, а потім для нього та його сусідів здійснюємо оновлення вагових коефіцієнтів (див. рис. 13.34);

2) відповідно до функції сусідства в міру протікання процесу навчання, дедалі менше сусідів будуть піддаватися оновленню, оскільки для «далеких» сусідів значення функції сусідства буде близьким до нуля, і ними можна знехтувати (див. рис. 13.35).

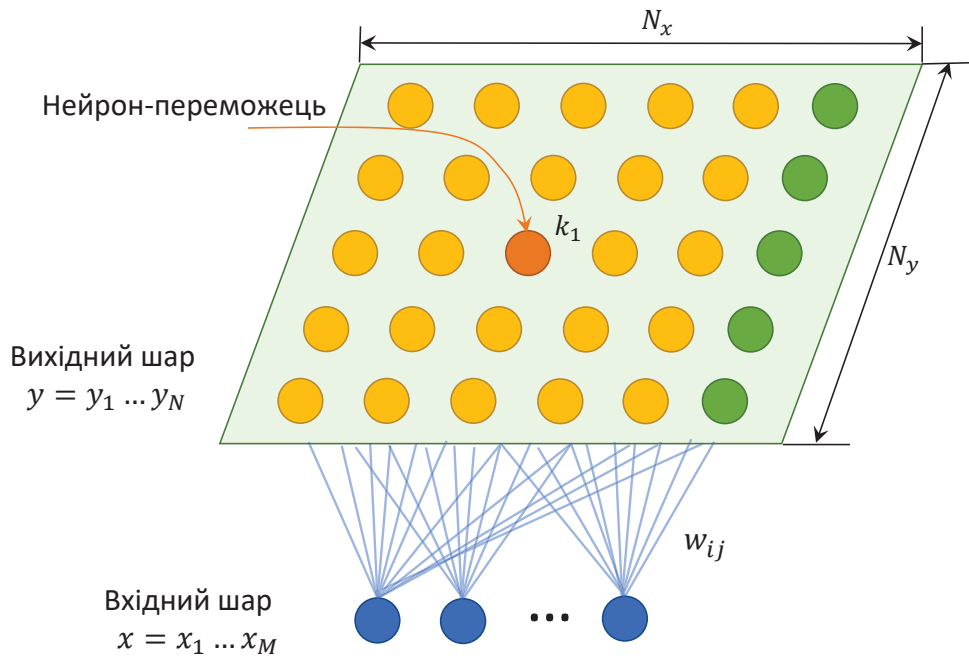


Рисунок 13.34 – Схематичне подання карти Кохонена

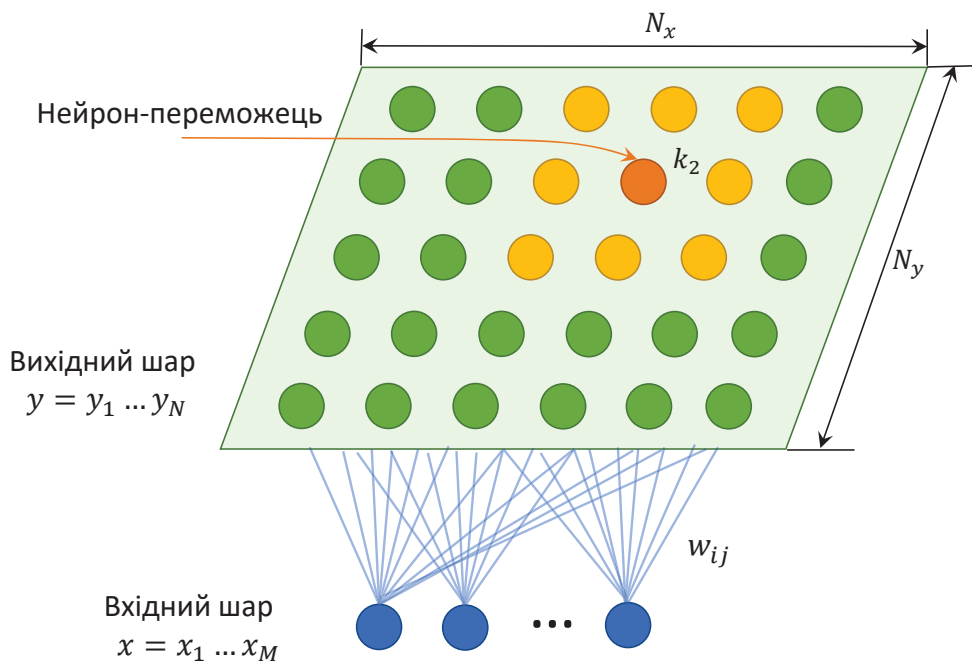


Рисунок 13.35 – Схематичне подання карти Кохонена

Найчастіше під час коригування вагових коефіцієнтів нейрон-переможець та його сусіди переміщуються ближче до того об'єкта, який було подано на вхід на поточній ітерації.

Розберемо детальніше цей процес.

1. На початку навчання вихідний шар має структуру, як показано на рисунку 13.36а.

2. На вхід карти Кохонена подається один з об'єктів вибірки $\mathbf{x}_0 = \{x_{01}, x_{02}, x_{03}, \dots, x_{0M}\}$.

3. У межах подання нейронів у вигляді двовимірної сітки ми маємо справу з двома просторами:

- n -вимірний простір, у якому нейрон з індексом k характеризується координатами $\{w_{1k}(0), w_{2k}(0), w_{3k}(0), \dots, w_{nk}(0)\}$;

- двовимірний простір, у межах якого ми графічно розмістили вихідні нейрони.

4. Визначаємо нейрон-переможця з індексом k (позначений помаранчевим кольором) та його сусідів (позначені жовтим кольором), як показано на рисунку 13.36б.

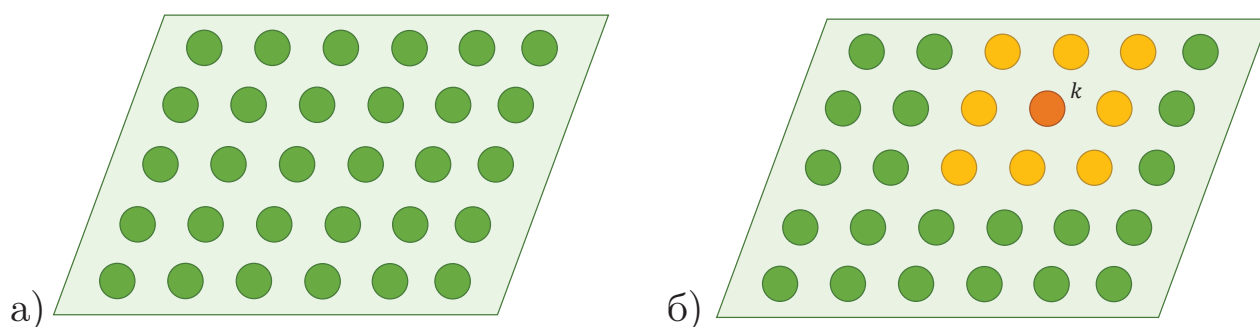


Рисунок 13.36 – Схематичне подання вихідного шару карти Кохонена: а) до навчання та б) після навчання

5. Одержані вектор $\mathbf{y}_k = \{w_{1k}(1), w_{2k}(1), w_{3k}(1), \dots, w_{Mk}(1)\}$ для нейрона-переможця з індексом k та вектори його найближчого оточення $\mathbf{y}_j = \{w_{1j}(1), w_{2j}(1), w_{3j}(1), \dots, w_{Mj}(1)\}$ після перерахунку вагових коефіцієнтів $w_{ij}(0) \rightarrow w_{ij}(1)$ за формулою (13.13) стають ближчими до вхідного вектора \mathbf{x}_0 . Водночас у двовимірному вигляді вихідні нейрони залишаються розміщені так само.

13.4.2. Приклади використання карт Кохонена

Далі розглянемо низку наочних прикладів.

Приклад № 1: кластеризація за ознаками

Припустимо, у нас є набір із 100 000 об'єктів, кожен із яких має свій колір. RGB-складові кольори – це три значення, відповідно об'єкти мають по 3 ознаки. Вихідних нейронів нехай буде 400, їх розмістимо у вигляді сітки розміром 20 на 20 (графічно репрезентуємо кожен із них у вигляді прямокутника).

Значення вагових коефіцієнтів ініціалізуємо випадковими величинами, а для візуалізації кожен із нейронів зобразимо за допомогою кольору, причому цей колір відповідатиме його ваговим коефіцієнтам. Так для нейрона зі значеннями вагових коефіцієнтів $\{255, 0, 0\}$ колір буде червоним. На етапі ініціалізацію маємо конфігурацію карти Кохонена, як показано на рисунку 13.37.

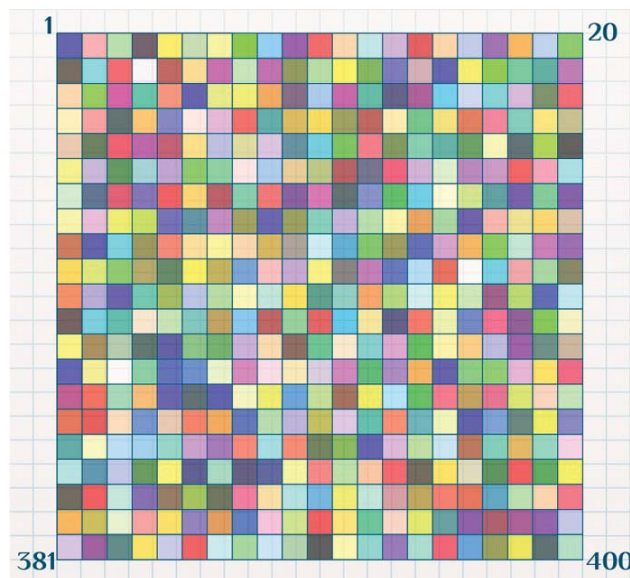


Рисунок 13.37 – Ініціалізація карти Кохонена

Оскільки вагові коефіцієнти проініціалізовані випадковими величинами, то кольори нейронів аналогічно випадкові.

За навчальну вибірку використовуємо згадані 100 000 об'єктів лише не якихось конкретних, а випадково згенерованих. Це дозволить одержати рівномірне розподілення їх ознак. Навчання карти

Кохонена проводять методом почергового подання на вхід мережі кожного з об'єктів навчальної вибірки. У результаті отримуємо карту Кохонена, як показано на рисунку 13.38.

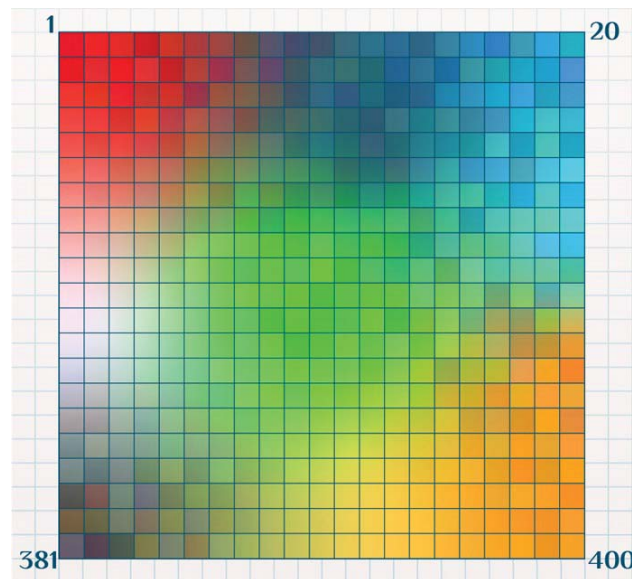


Рисунок 13.38 – Результат навчання карти Кохонена

У разі подачі на вхід одного з елементів відбувається коригування нейрона-переможця та сусідів, що приводить до «переміщення» цих нейронів «ближче» до вхідного вектора. У цьому разі це означає зміну кольору. Тобто, якщо на вході в нас об'єкт з ознаками $\{0, 255, 0\}$ (колір – зелений), то нейрон-переможець та його сусіди змінять свої вагові коефіцієнти так, щоб їх значення стали «ближчими» до $\{0, 255, 0\}$, тобто їх кольори стануть «зеленішими».

Під час вивчення нейронних мереж Кохонена на початку теми ми використовували термін «кластер», причому кожен вихідний нейрон відповідав одному кластеру. У прикладі з картами Кохонена, що самоорганізуються, сенс кластера стає трохи іншим. Кластером буде група нейронів вихідного шару, відстань між якими менша, ніж відстань до сусідніх груп. Отже, для цього прикладу можна ефективно виділити 6 кластерів за кольорами, як показано на рисунку 13.39. Групи нейронів, що мають схожі значення ознак (а значить і колір при такому графічному відображенні), згруповані в кластери.

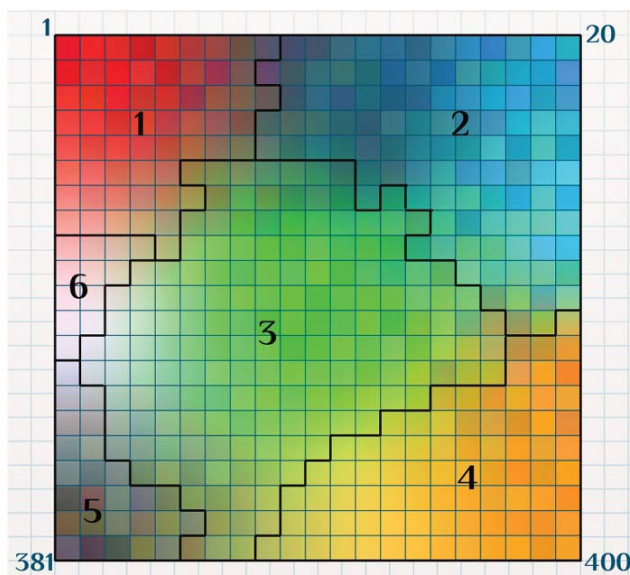


Рисунок 13.39 – Кластери на карті Кохонена

Приклад № 2: аналіз за ознаками та прогнозування

У попередньому прикладі ми використовували забарвлення нейронів відповідно до всіх трьох його ознак. Зазвичай буває дещо інакше. Нехай цього разу об'єктами для дослідження є деякі організації. Ознаками будуть певні дані фінансової звітності – доходи, оборот, кількість співробітників та ще щось аналогічне. Усього, припустимо, ознак 10, а об'єктів, які будуть використані для навчання – 10 000. Сформуємо картку Кохонена, що самоорганізується, з вихідними нейронами у вигляді сітки 20 на 20.

Навчаємо мережу, одержуємо змінені значення вагових коефіцієнтів, при цьому структурування нейронів не змінюється.

Шляхом забарвлення одержаної карти відповідно до значення однієї з 10-ти ознак, ми матимемо наочне подання результатів. Наприклад, перша ознака – це прибуток підприємства за попередній звітний період. Серед 10 000 об'єктів, якими ми оперували під час навчання, мінімальне значення становить 0.1 млрд, максимальне – 1 млрд. Прийнемо, що 0 млрд відповідає чорному кольору, а 1 млрд – червоному, і пофарбуємо вихідні нейрони відповідно до значення першої ознаки кожного з них. Так вихідний нейрон із координатами-ознаками $\{1, x_2, x_3, \dots, x_n\}$ буде забарвлений у чер-

воний колір $\{255, 0, 0\}$, а нейрон з координатами $\{0.5, x_2, x_3, \dots, x_n\}$ – у проміжний між чорним і червоним кольором, $\{127, 0, 0\}$ (0.5 тут – це дохід 0.5 млрд), як показано на рисунку 13.40.

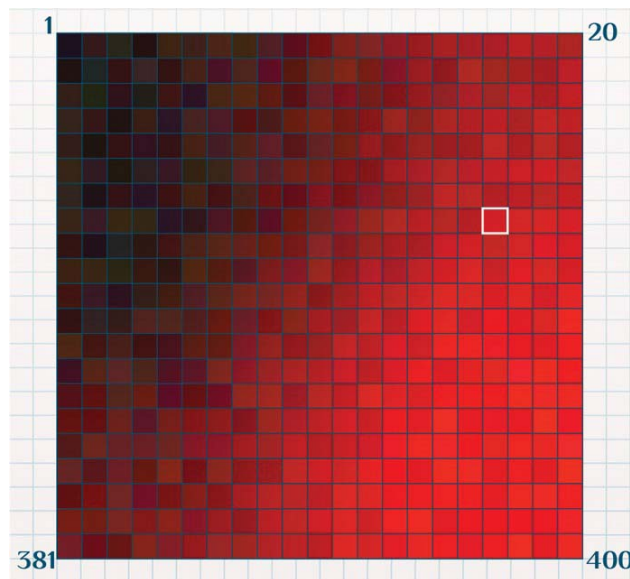


Рисунок 13.40 – Результат навчання

Отже, з використанням карти Кохонена ми, по-перше, зменшили вихідний набір із 10 000 об'єктів із 10 ознаками кожен до наочного подання у вигляді сітки з 400 комірок. По-друге, можемо аналізувати дані шляхом проведеного забарвлення комірок-нейронів залежно від значення тієї чи іншої ознаки. У результаті ми одержуємо карту Кохонена, за допомогою якої можемо оцінити доходи організацій.

Побудуємо ще одну карту Кохонена, у якій забарвлення будемо проводити залежно від того, скільки об'єктів із вибірки потрапили в ту чи іншу комірку. Оскільки вихідних об'єктів більше (до того ж на порядок), ніж комірок-нейронів, то логічно, що в більшість комірок потрапило кілька об'єктів.

Нехай максимальна кількість об'єктів в одній комірці дорівнює 45, а мінімальна – 4. Задамо для значення 45 синій колір, а для значення 4 – білий колір. Одержуємо наочну демонстрацію, як розподілені об'єкти по комірках карти Кохонена, як на рисунку 13.41. Це надає відмінний механізм для аналізу даних.

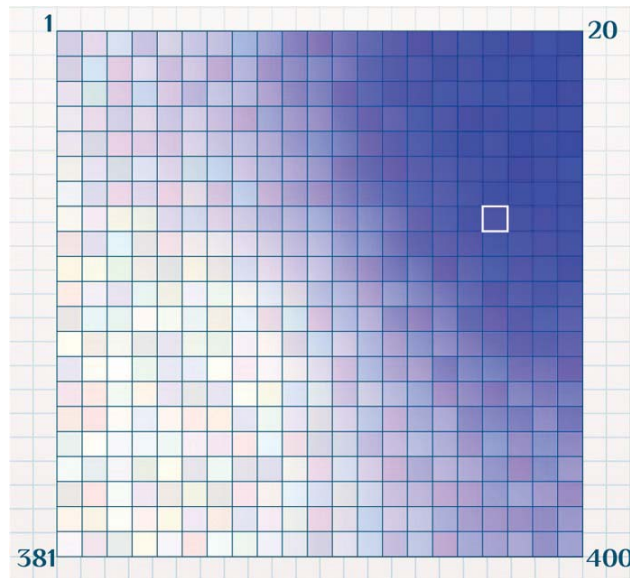


Рисунок 13.41 – Результат навчання

У цьому конкретному прикладі, на основі всього двох варіантів подання карти Кохонена ми можемо зробити певні висновки.

1. Із другої карти бачимо, що переважна кількість об'єктів розташовані у верхньому правому кутку. А за першою картою робимо висновок, що дохід у цій галузі середній, оскільки колір комірок між червоним та чорним.

2. Розглянемо комірку, позначену білим кольором. За першою картою можемо зробити висновок про середній дохід, за другою – про кількість організацій, що потрапили до цієї групи. Далі цю інформацію можна використовувати для подальшого фінансового аналізу.

3. Із першої карти видно дві області: мінімальний дохід – верхній лівий кут, максимальний – нижній правий. Із другої ж можна дійти висновку, що успішних організацій більше, ніж збиткових (колір нижнього правого кута біль синій ніж верхнього лівого).

Крім того, з урахуванням одержаних даних можна робити певні прогнозування. Наприклад, крім набору з 10-ти ознак, що брали участь у навчанні, існує ще один фактор – перспективність вкладень в акції компанії. Для наявних 10 000 об'єктів цей чинник відомий (хоча й не брав участі в навчанні), але є низка організацій, для яких цей чинник не визначено.

Для кожної комірки знайдемо середнє значення «перспективності», беручи значення віднесених до комірки об'єктів та усереднюючи їх. Далі навчену карту Кохонена пофарбуємо відповідно до нового чинника: перспективніші комірки – золотий колір, менш перспективні – чорний колір, як показано на рисунку 13.42а.

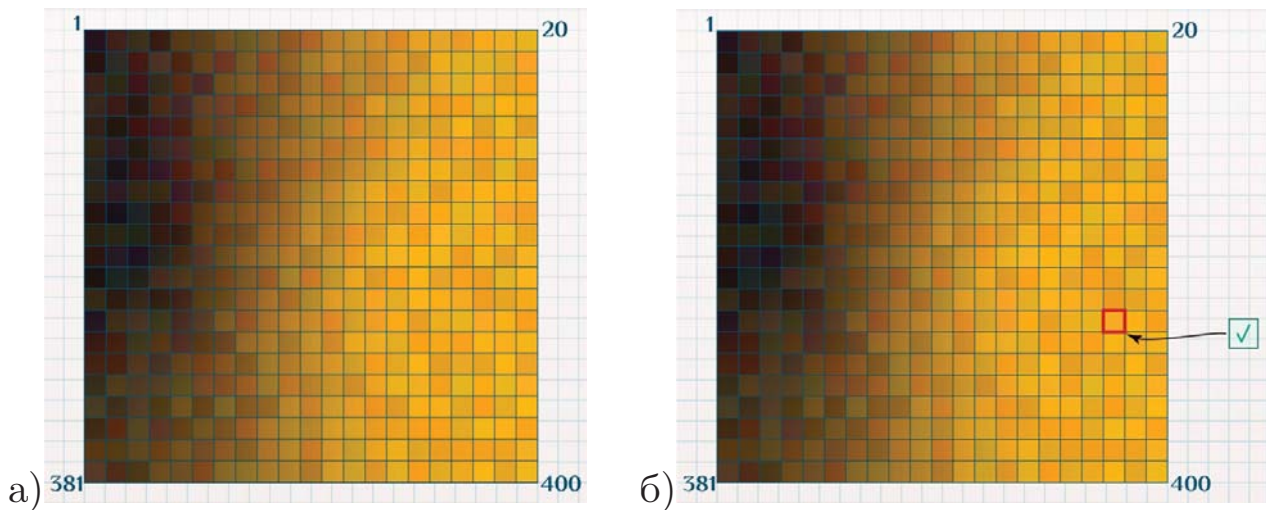


Рисунок 13.42 – Прогнозування за допомогою карт Кохонена

Беремо нову організацію, для якої з фінансової звітності взято 10 ознак, але «перспективність» не відома, і подаємо її на вхід мережі. У результаті цей новий об'єкт буде віднесений до якоїсь із комірок карти. Нова точка розміщена досить близько до найперспективніших (золотих) комірок, як показано на рисунку 13.42б. Це сигнал, що цілком може виявитися прогноз про те, що і в цю організацію є сенс інвестувати.

Приклад № 3: стискання даних

Завдання: є 500 об'єктів, кожен із яких має дві ознаки. Причому якщо відобразити об'єкти на двовимірній площині, де за віссю абсцис відкладено значення першої ознаки, а за віссю ординат – другої, то одержуємо таке розподілення об'єктів у просторі ознак, як показано на рисунку 13.43а. На осях – значення ознак досліджуваних об'єктів (діапазон для обох ознак: від 0 до 1). Карта Кохонена матиме на виході сітку розміром 10×10 зі 100 нейронів. Вагові коефіцієнти зв'язків знову проініціалізуємо випадковими величинами,

нехай з інтервалу $(0.4, 0.6)$, і відобразимо вихідні нейрони на тій самій площині зі значеннями ознак за осями, як показано на рисунку 13.43б.

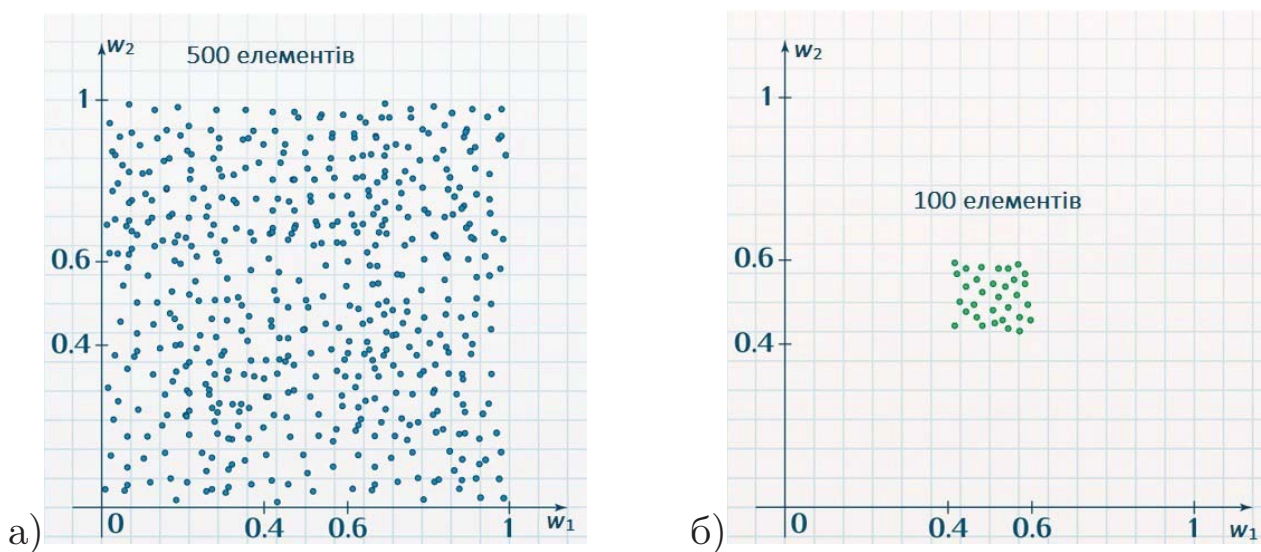


Рисунок 13.43 – Об'єкти (а) та карта Кохонена (б) до навчання

Проводимо навчання, у результаті якого знову нейрон-переможець і сусідні до нього нейрони зміщуватимуться ближче до елементів, що подаються на вхід, що в глобальному результаті приведе до нових значень вагових коефіцієнтів. У результаті навчання відображення вихідних нейронів матиме вигляд, як на рисунку 13.44. Нейрони зміщувалися в процесі навчання до зразків, що подаються, і в результаті повторюють форму, яка відповідає розміщенню початкових об'єктів. Отже, карта Кохонена, дозволяє «спростити» (стиснути) дані (в цьому разі замість 500 елементів одержали 100), зберігаючи взаємозалежності та тип цих даних. Водночас структура не змінилася, а значення ознак значно скоригувалися.

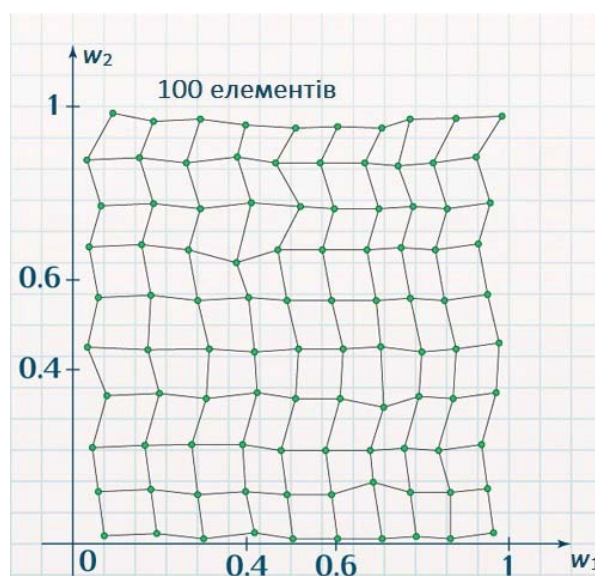


Рисунок 13.44 – Результат навчання

Приклад № 4: одновимірне подання вихідних нейронів

Ключовою відмінністю від попереднього прикладу є механізм визначення сусідів. На вході ті самі 500 об'єктів, на виході ті самі 100 нейронів, проініціалізовані значеннями ознак з діапазону (0.4, 0.6) як на рисунку 13.45.

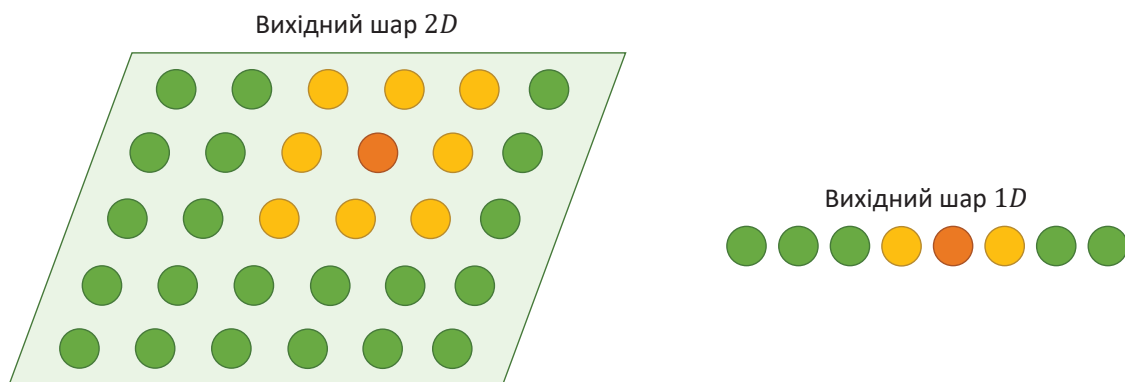


Рисунок 13.45 – Одновимірне подання нейронів вихідного шару

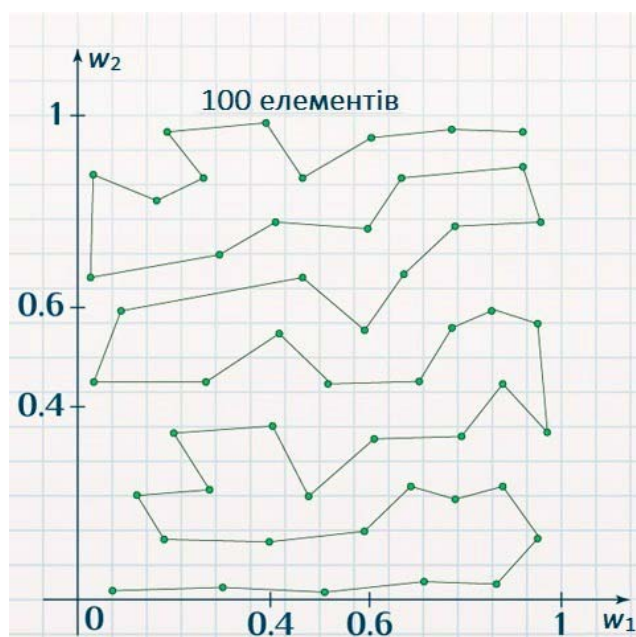


Рисунок 13.46 – 1D карта Кохонена

Результат навчання буде дещо іншим, як показано на рисунку 13.46. Проте мережа чітко працює, тобто значення вагових коефіцієнтів дозволяють вихідним нейронам успішно відбивати структуру вхідних даних.

Розділ 4. Глибоке навчання. Комп'ютерний зір. Згорткові нейронні мережі

Тема 14. Глибокі нейронні мережі та глибоке навчання

Поняття *неглибока нейронна мережа* (або просто нейронна мережа (НМ)) та *глибока нейронна мережа* (ГНМ) є дещо абстрактними. Під неглибокою НМ зазвичай розуміють мережу з одним прихованим шаром. Відповідно, глибока НМ – це мережа з двома та більше прихованими шарами. Чим більше прихованих шарів, тим більш глибокою вважається нейронна мереж. Із 2006 року було розроблено кілька технологій, що дозволяють навчати глибокі НМ. Вони ґрунтуються на методах стохастичного градієнтного спуску та зворотного поширення помилки, але містять і нові ідеї. Це дозволило навчати більш глибокі мережі – сьогодні без проблем навчають мережі з 5–10 шарами. І виявляється, що вони набагато краще вирішують більшість проблем, ніж неглибокі НМ, тобто мережі з одним прихованим шаром.

14.1. Сенс прихованих шарів

Розглянемо процес навчання «не дуже» глибокої нейромережі, що складається з вхідного шару, двох прихованих шарів та вихідного шару. Припустимо, завдання мережі – розпізнати рукописний символ (цифру) зображення якої має розмір $28 \times 28 = 784$ пікселів (приклад рукописних цифр із бібліотеки `mnist`). У такому разі вхідний шар складається з 784 нейронів, вихідний шар містить 10 нейронів, як показано на рисунку 14.1.

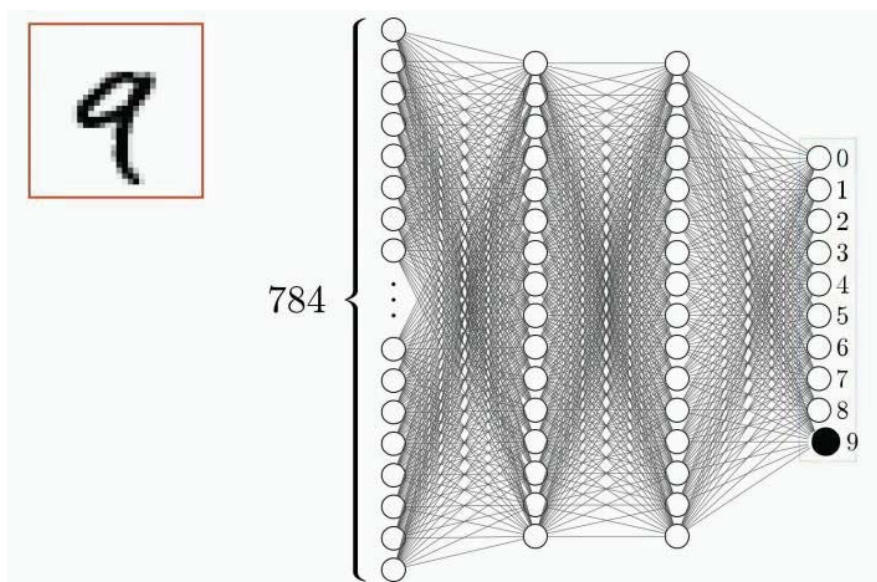


Рисунок 14.1 – Нейромережа, що складається зі вхідного шару, двох прихованих шарів та вихідного шару, для розпізнавання рукописних цифр

Принцип роботи нейромережі полягає в тому, що активація в одному шарі визначає активацію в наступному. Збуджуючись, деяка група нейронів викликає збудження іншої групи. Якщо передати навченій нейронній мережі на перший шар значення активації згідно яскравості кожного пікселя картинки, ланцюжок активацій від одного шару нейромережі до наступного приведе до переважної активації одного з нейронів останнього шару, що відповідає розпізнаній цифрі – вибору нейронної мережі.

Визначимо що роблять проміжні шари між вхідним та вихідним шарами. Для цього спробуємо уявити процеси, які відбуваються всередині нейромережі. Природно припустити, що в процесі розпізнавання цифр різні компоненти зображення зводять воєдино. Наприклад, дев'ятка складається з кола зверху та лінії праворуч; вісімка також має коло вгорі, але замість лінії праворуч, у неї є коло знизу; четвірку можна уявити як три певним чином з'єднані лінії (як показано на рисунку 14.2).

В ідеалізованому випадку переважно можна очікувати, що кожен нейрон із другого прихованого шару співвідноситься з одним із цих компонентів. Коли нейромережі передається зображення з

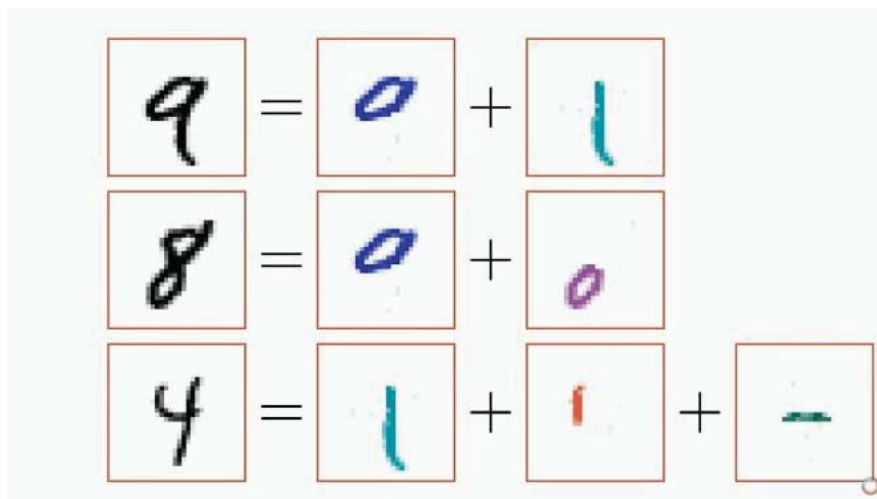


Рисунок 14.2 – Різні компоненти зображення рукописних цифр

колом у верхній частині, існує певний нейрон, чия активація стане ближче до одиниці. Отже, перехід від другого прихованого шару до вихідного відповідає знанням про те, який набір компонентів якій цифрі відповідає. Схематично активацію певних нейронів нейромережі при розпізнаванні рукописних цифр репрезентовано на рисунку 14.3.

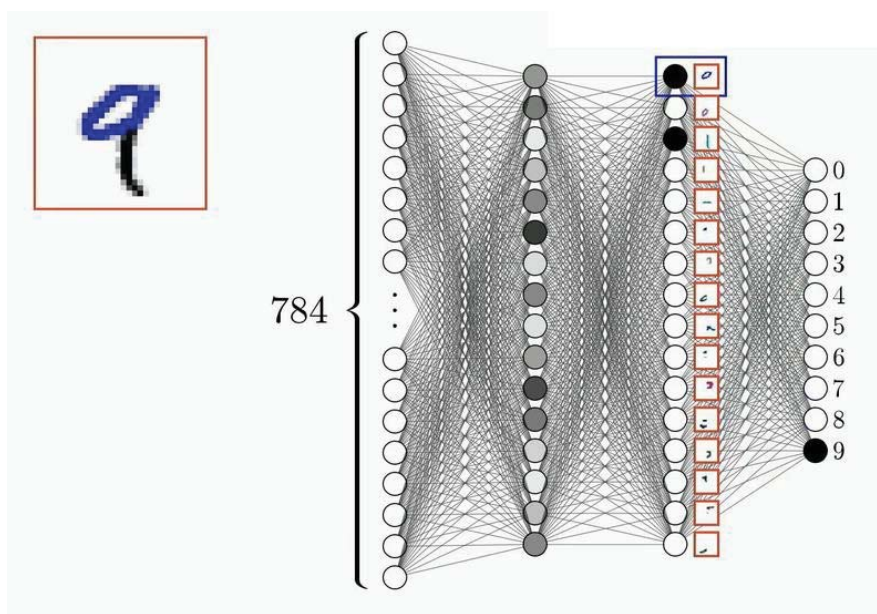


Рисунок 14.3 – Активація певних нейронів нейромережі при розпізнаванні рукописних цифр

Завдання розпізнавання кола так само можна розбити на підзавдання. Наприклад, розпізнавати різні маленькі грані, із яких воно утворене (див. рис. 14.4а). Аналогічно довгу вертикальну лінію можна подати як шаблон з'єднання кількох менших відрізків (див. рис. 14.4б).

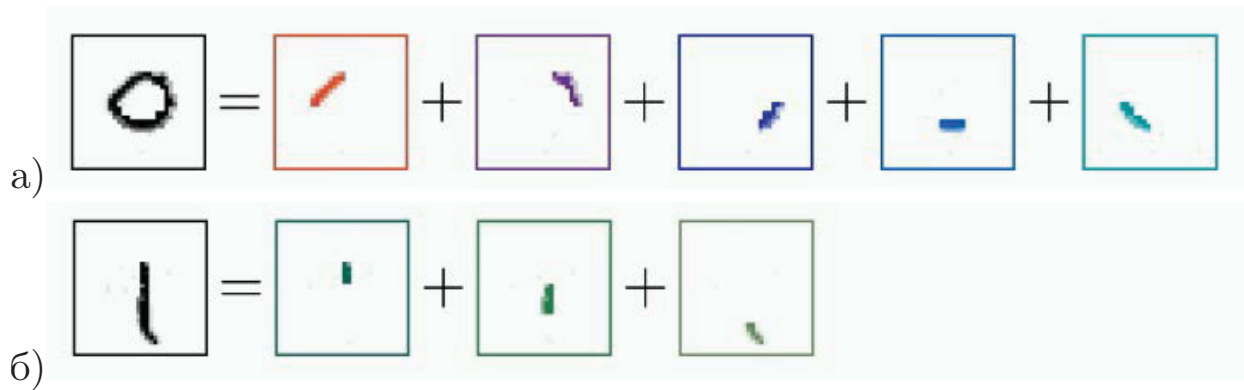


Рисунок 14.4 – Виділення примітивів для компонентів зображення.

Отже, можна сподіватися, що кожен нейрон із першого прихованого шару нейромережі здійснює операцію розпізнавання цих малих граней. Вхідне зображення призводить до активації певних нейронів першого прихованого шару, які визначають характерні малі деталі. Ці нейрони зі свого боку активують більш великі форми, у результаті активуючи нейрон вихідного шару, асоційований із певною цифрою. У цьому разі зображення цифри на кожному прихованому шарі нейронної мережі бкде виглядати як показано на рисунку 14.5.

Чи буде діяти нейромережа саме так, чи ні – складне питання, однак наведені вище міркування можуть бути певним орієнтиром для розуміння сенсу шарованої структури нейромережі. Якщо припустити, що мережа працює саме так, як описано вище, тоді вагові коефіцієнти зв'язків нейронів вхідного шару з певним нейроном із першого прихованого шару (припустимо, що цей нейрон відповідає за коротку вертикальну лінію) будуть позитивними, якщо цей елемент (вертикальна лінія) присутня на зображенні. У разі наявності позитивних зв'язків даний нейрон буде активований. Отже, сформувавши з вагових коефіцієнтів зв'язків цього нейрона матрицю розміром 28×28 та візуалізувавши її, ми повинні побачи-

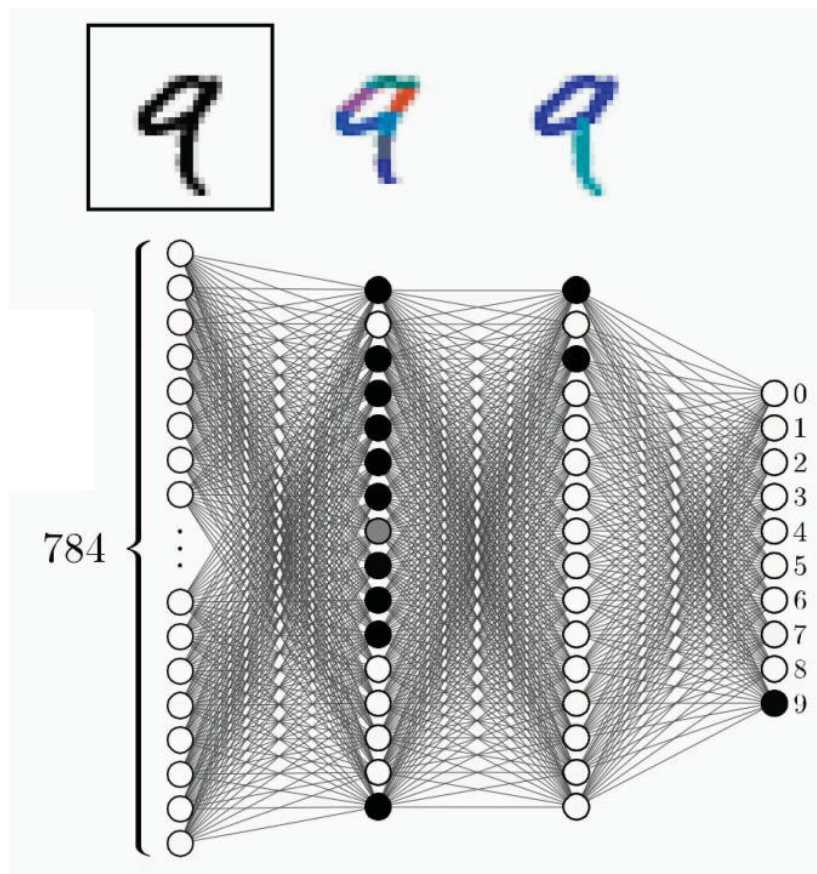


Рисунок 14.5 – Зображення цифри на кожному прихованому шарі нейронної мережі

ти елемент зображення на екрані. Не саме зображення, звичайно, а його подання в мережі, однак на практиці ми не побачимо очікуваних фігур, які б відповідали малим елементам цифр. Ми побачимо набагато менш ясні структури, що відповідають тому, як нейронна мережа мінімізувала функцію вартості. На рисунку 14.6 показано структури для різних нейронів першого прихованого шару після навчання мережі.

На зображеннях відсутні явно виражені елементи цифр тому що це усереднені результати по всьому набору прикладів зображень цифр із навчальної вибірки, однак на малюнках явно видно візерунки з темних та світлих плям. Кожен із цих візерунків є візитною карткою того чи іншого елемента зображення й мережа навчилася розуміти який із візерунків якому елементу відповідає.

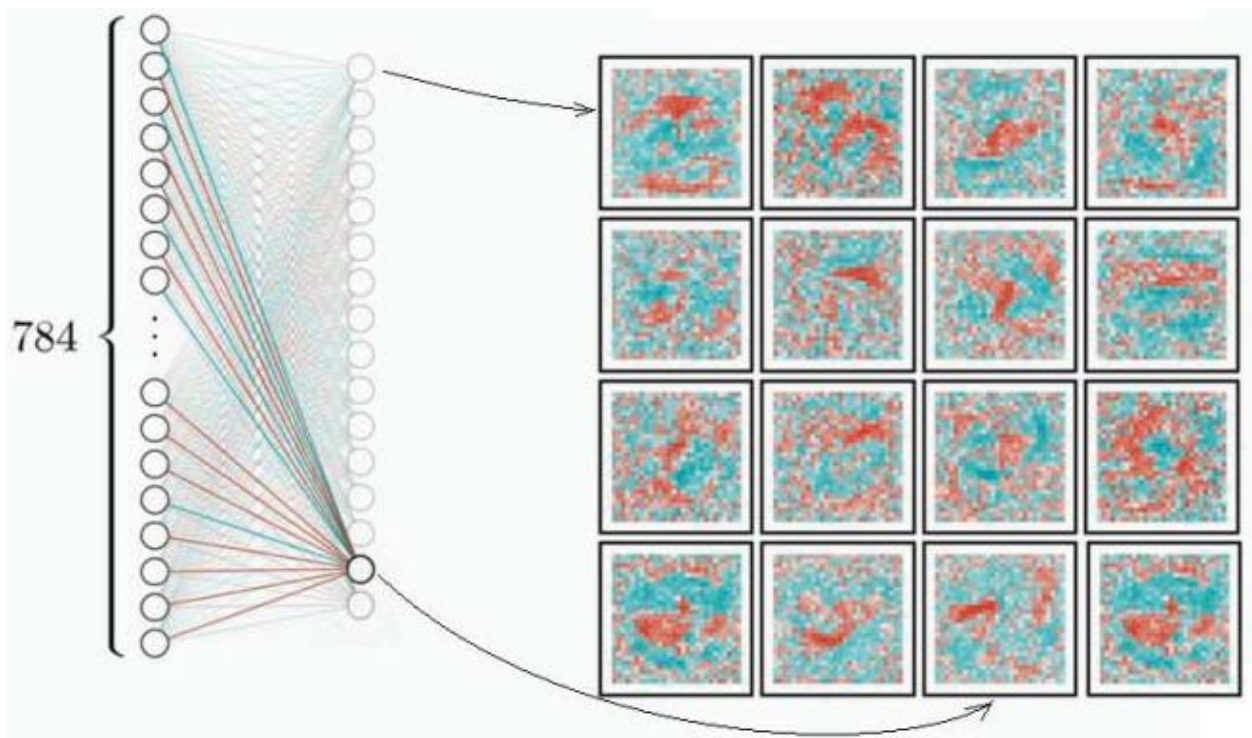


Рисунок 14.6 – Зображення структури для різних нейронів першого прихованого шару після навчання мережі

14.2. Застосування глибокого навчання в задачах комп'ютерного зору

Існує безліч завдань комп'ютерного зору, які вирішують за допомогою нейронних мереж глибокого навчання. Серед них можна виділити класифікацію зображень, виявлення об'єктів, семантичну сегментацію та сегментацію екземплярів. Розглянемо детальніше наведені задачі.

Класифікація зображень передбачає присвоєння мітки або класу цифрового зображення. Наприклад, знімок безпілотно-го літального апарата на рисунку 14.7 зліва може бути помічений як натовп, а цифрова фотографія на рисунку 14.7 справа – як кішка. Цей тип класифікації також відомий як класифікація об'єктів або розпізнавання зображень, і він може бути використаний для категоризації об'єктів на зображенні.



Рисунок 14.7 – Класифікація зображень із використанням глибокого навчання

Виявлення об'єктів – це процес пошуку об'єктів на зображенні. Наприклад, на знімку дистанційного зондування наведеному на рисунку 14.8 зліва нейронна мережа виявила місцезнаходження літака. У більш загальному варіанті використання комп'ютерного зору модель може виявляти розміщення різних тварин (див. рис. 14.8 справа). Цей процес зазвичай передбачає побудову обмежу-



Рисунок 14.8 – Виявлення об'єктів із використанням глибокого навчання

вального прямокутника навколо об'єктів, що цікавлять. Він може бути використаний для визначення розміщення конкретних об'єктів на супутникових, аерофотознімках або знімках з БПЛА та нанесення цих об'єктів на карту.

Семантична сегментація відбувається, коли кожен піксель у зображенні класифікується як той, що належить якомусь класу. Наприклад, на зображенні зліва на рисунку 14.9 дорожні пікселі класифікують окремо від недорожніх пікселів. На рисунку 14.9 справа пікселі, що зображують кішку на фотографії, класифікують як кішку, тоді як інші пікселі на цьому зображенні належать до інших класів. Класифікацією пікселів часто називають сегментацією зображень чи класифікацією зображень. Це часто використовують для створення карток класифікації землекористування.

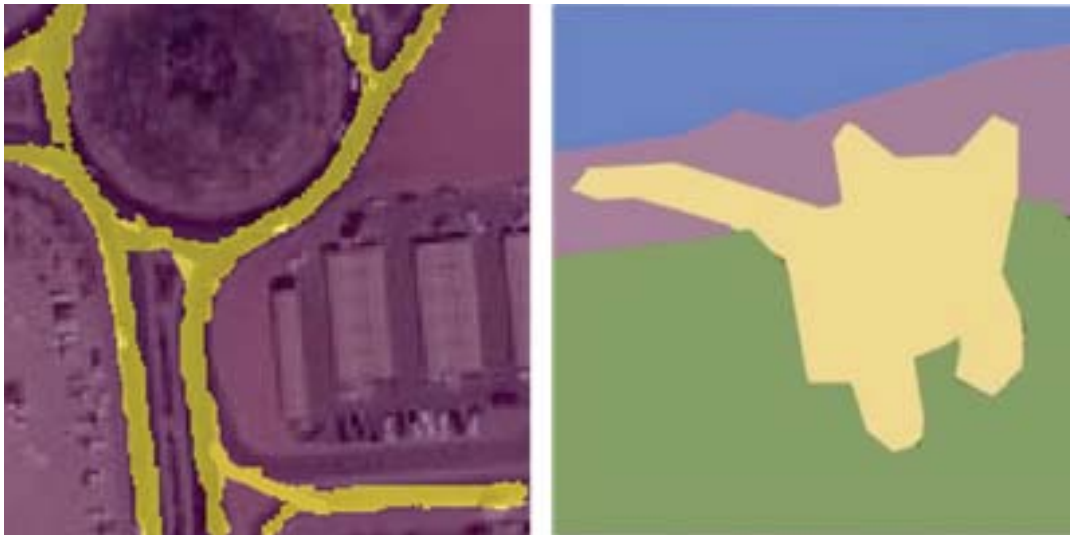


Рисунок 14.9 – Семантична сегментація з використанням глибокого навчання

Сегментація екземпляра – це точніший метод виявлення об'єктів, у якому малюють межу кожного екземпляра об'єкта. Наприклад, на зображенні зліва на рисунку 14.10 видно дахи будинків, зокрема точний контур даху. На рисунку 14.10 справа видно машини, і можна чітко бачити форму машин. Застосування цього глибокого навчання також відоме як сегментація об'єктів.



Рисунок 14.10 – Сегментація екземпляра з використанням глибокого навчання

Загальна сегментація поєднує в собі як семантичну сегментацію, так і сегментацію екземплярів. Наприклад, на зображенні на рисунку 14.11 показано, що всі пікселі класифіковані, і кожен унікальний об'єкт, наприклад кожен автомобіль, є окремим унікальним об'єктом.



Рисунок 14.11 – Загальна сегментація з використанням глибокого навчання

Перетворення зображення – це завдання перетворення зображення з одного можливого уявлення або стилю сцени в інший, наприклад зменшення шуму або суперроздільна здатність. Наприклад, зображення на рисунку 14.12 зліва показує вихідне зображення низької роздільної здатності, а зображення праворуч показує результат застосування глибокої нейронної мережі.



Рисунок 14.12 – Приклад збільшення контрастності знімка

Виявлення змін – завдання глибокого навчання виявити зміни в об'єктах, що цікавлять, у проміжку між двома датами й створити логічну карту змін. Наприклад, на рисунку 14.13 зображення зліва внизу показує будівництво п'ять років тому, зображення посередині показує це будівництво зараз, а зображення праворуч показує логічну карту змін, де нові будинки відображені білим.



Рисунок 14.13 – Приклад виявлення змін на знімках

Тема 15. Згорткові нейронні мережі

15.1. Структура згорткової мережі та основні парадигми

Досі не існує методів, що дозволяють однозначно визначити структуру та склад нейромережі виходячи з огляду на опис завдання. Одне з емпіричних правил свідчить, що кількість нейронів у прихованому шарі повинна бути хоча б на порядок більша від кількості входів. Якщо взяти до уваги що саме собою перетворення із зображення в індикатор класу досить складне та істотно нелінійне, одним шаром тут не обійтися. Беручи до уваги вищесказане можна оцінити, що кількість нейронів у прихованих шарах буде приблизно 15 000 (10 000 у другому шарі та 5 000 у третьому). Водночас для конфігурації з двома прихованими шарами кількість зв'язків, які налаштовуються та навчаються буде 10 млн між входами та першим прихованим шаром +50 млн між першим та другим +50 тис. між другим і вихідним, якщо рахувати що в нас 10 виходів, кожен із яких позначає цифру від 0 до 9 (завдання розпізнавання рукописних цифр). Разом приблизно 60 000 000 зв'язків. Під час навчання до кожного зв'язку потрібно буде обчислювати градієнт помилки.

Коли зображення перетворюється на лінійний ланцюжок байтів, то щось безповоротно втрачається, а саме – топологія зображення, тобто, взаємозв'язок між окремими його частинами. Крім того, завдання розпізнавання передбачає вміння нейромережі бути стійкою до невеликих зсувів, поворотів та зміни масштабу зображення, тобто, вона повинна отримувати з даних деякі інваріанти, які не залежать від почерку людини (для завдання розпізнавання рукописних букв чи цифр). Отже, оптимальна нейромережа повинна бути не дуже обчислювально складною і більше інваріантною до різних спотворень зображення.

Розглянута нами в розділі 2 мережа прямого поширення вміє розпізнавати символи однакового формату, відцентровані та попередньо оброблені. У разі, коли символ буде зменшено й написано

в кутку всього зображення, то така мережа мережа не зможе його правильно класифікувати, оскільки кожен нейрон внутрішнього шару жорстко прив'язаний до певних координат пікселів зображення.

Вирішення цієї проблеми було знайдено американським ученим французької походження Яном Ле-Куном, натхненним роботами нобелівських лауреатів в області медицини Torsten Nils Wiesel та David H. Hubel. Ці вчені досліджували зорову кору головного мозку кішки та виявили, що існують так звані прості клітини, які особливо сильно реагують на прямі лінії під різними кутами та складні клітини, які реагують на рух ліній в одному напрямі. Ян Ле-Кун запропонував використовувати так звані **згорткові нейронні мережі**. Ідея згорткових нейронних мереж полягає в чергуванні прихованих згорткових шарів **C-layers**, прихованих субдискретизуючих шарів **S-layers** та наявності повнозв'язних шарів **F-layers** на виході. Така архітектура містить у собі три основних парадигми:

- 1) локальне сприйняття;
- 2) розділення вагових коефіцієнтів;
- 3) субдискретизація.

Розглянемо ці парадигми детальніше.

Локальне сприйняття передбачає, що на вхід одного нейрона внутрішнього шару подається не все зображення (або виходи попереднього шару), а лише деяка його область. Такий підхід дозволяє зберігати топологію зображення від шару до шару. Тобто, якщо на вході зображення має розмір 32×32 пікселя, то кожен із нейронів наступного шару прийме на вхід лише невелику ділянку цього зображення розміром, наприклад, 5×5 . Водночас вхідне зображення не переводиться на одновимірний вектор-стовпець, аналізується двовимірний масив, а отже, топологія зображення не губиться.

Концепція **розділення вагових коефіцієнтів** припускає, що для великої кількості зв'язків використовують дуже невеликий набір вагових коефіцієнтів, як це схематично подано на рисунку 15.1. Фактично кожен нейрон наступного внутрішнього шару (наприклад, нейрон 1 на рисунку 15.1 справа) «відповідає» за певну ділянку попереднього шару (матриця A). Вхід цього нейрона розраховують аналогічно тому, як це робили раніше. Маємо 25 зв'язків,

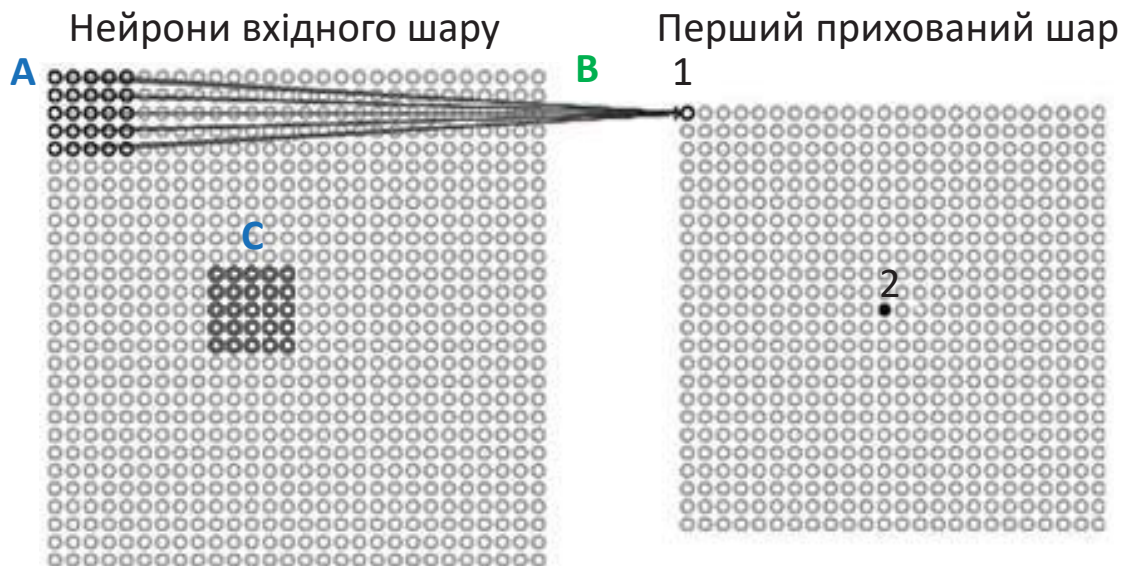


Рисунок 15.1 – Схематичне подання концепції розділення вагових коефіцієнтів

причому вагові коефіцієнти цих зв'язків занесені в матрицю розміром 5×5 (на початковому етапі ця матриця формується за допомогою генератора випадкових чисел). Для одержання входу нейрона 1 необхідно поелементно перемножити вагові коефіцієнти на вхідний сигнал ($A \cdot B$) і підсумувати, за аналогією до звичайних нейронетворів. Аналогічну процедуру виконують для кожної ділянки вхідної матриці (попереднього шару), наприклад, добуток $C \cdot B$ формує вхідний сигнал нейрона 2 на рисунку 15.1 справа.

Потрібно зазначити, що матрицю B – матрицю вагових коефіцієнтів називають фільтром чи ядром; вона не змінюється під час формування сигналів на входах наступного шару (перший прихований шар на рисунку 15.1). Фактично вагові коефіцієнти зв'язків нейронів наступного шару з нейронами попереднього шару однакові. У такому разі перший прихований шар формується за таким принципом. Відбувається послідовне переміщення фільтра вправо на одиницю. Переміщення може відбуватися й на декілька одиниць. Головне, щоб відбувалося перекриття фільтрами зображення: кількість кроків переміщення фільтра повинна бути меншою за лінійний розмір фільтра. Кожна унікальна позиція уведеного зображення генерує число. Після проходження фільтра по всіх позиціях

виходить матриця розміром 28×28 (перший прихований шар), яку називають **карткою ознак**. Перший прихований шар (або карта ознак) являє собою матрицю розміром 28×28 , оскільки існує 784 різних позиції, що можуть пройти через фільтр розміром 5×5 зображення розміром 32×32 . Ці 784 числа перетворюються на матрицю розміром 28×28 . Матрицю B застосовують до зображення за допомогою математичної операції, яку називають згорткою (звідси назва – згортковий шар *C-layer*). Суть цієї операції в тому, що кожен фрагмент зображення множать на матрицю B – ядро згортки поелементно й результат підсумовують та записують в аналогічну позицію вихідного зображення, що подається на вхід нейрона наступного шару. Приклад згортки наведено на рисунку 15.2.

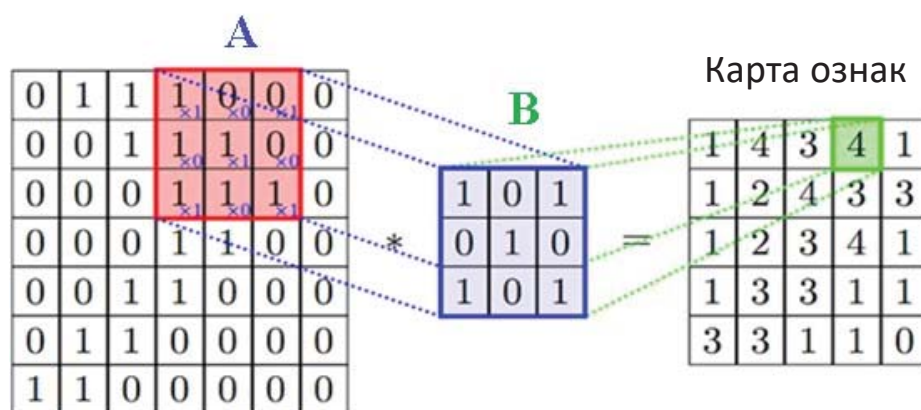


Рисунок 15.2 – Приклад згортки

Основна властивість фільтрів полягає в тому, що значення їх виходу тим більше, чим більше фрагмент зображення схожий на сам фільтр. Отже, зображення, згорнуте з якимось ядром (фільтром, матрицею вагових коефіцієнтів) B дасть інше зображення, кожен піксель якого означатиме ступінь схожості фрагмента зображення на фільтр B . Саме тому одержаний прихований шар називають карткою ознак.

Фільтрів може бути дуже багато, причому кожен із них відповідає за елементи, з яких складається вихідне зображення. Отже, прихований шар буде являти собою набір двовимірних карт ознак (буде мати шаровану структуру), причому кількість карт буде збігатися з кількістю фільтрів.

Розглянемо простий приклад, що подано на рисунку 15.3. Припустимо, у нас є зображення розміром 8×8 , і нашою метою буде визначення горизонтальних та вертикальних кордонів (зелені клітини на рисунку 15.3). Для досягнення поставленої мети будемо використовувати два фільтри розміром 2×2 (дві ознаки, які ми хочемо знайти на вихідному зображенні).

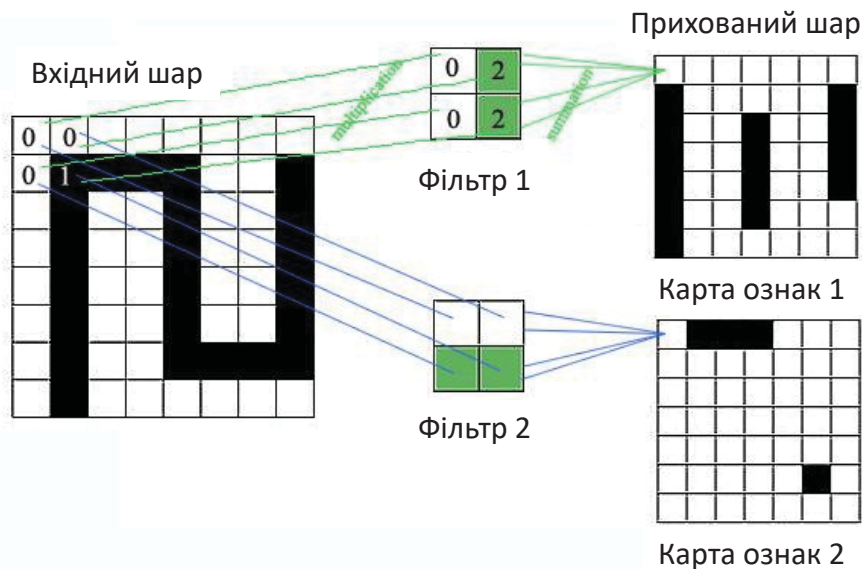


Рисунок 15.3 – Приклад використання фільтрів

Із наведеного прикладу видно, що перший прихований згортковий шар призначений для розкладання зображення на окремі елементи – примітиви. Причому кожна карта ознак містить інформацію про те, де на зображенні є той або інакший примітив. Кількість ядер (наборів вагових коефіцієнтів) визначає розробник і залежить від того, яка кількість ознак необхідно виділити. Зрозуміло, що для розпізнавання геометричних фігур, достатньо, наприклад, десяти ядер. Для розпізнавання фотографій кількість ядер можна обчислювати сотнями.

Неважко порахувати, що навіть для 10 ядер (фільтрів) розміром 5×5 для вхідного зображення розмірами 32×32 кількість зв'язків виявиться рівною приблизно 256 000 (порівнюємо з 10 млн. для мережі прямого поширення), а кількість налаштовуваних параметрів (значень елементів матриці згортки) всього $5 \times 5 \times 10 = 250$. Отже, швидкість навчання порівняно зі звичайною мережею пря-

мого поширення буде збільшуватися в рази. Водночас із використанням згорткової мережі якість розпізнавання підвищується. Справа в тому, що таке штучно уведенне обмеження на вагові коефіцієнти покращує узагальнювальні характеристики мережі, що в результаті позитивно позначається на можливості мережі шукати інваріанти в зображенні та реагувати здебільшого на них, не звертаючи уваги на інший шум.

Суть **субдискретизації** та шарів **S-layers** полягає в зменшенні просторової розмірності зображення. Тобто, вхідне зображення грубо (усередненням) зменшується в задану кількість разів. Частіше всього у 2 рази (крок дорівнює 2), хоча може бути й не рівномірна зміна, наприклад, 2 за вертикаллю та 3 за горизонталлю. Субдискретизація потрібна для забезпечення інваріантності до масштабу, прискорення процесу навчання та зменшення споживання обчислювальних ресурсів. Існує кілька способів виконати субдискретизацію. Розглянемо найбільш простий та поширений із них – максимальне об'єднання (процедура **Max Pool**). Операція максимального об'єднання полягає в тому, що вздовж даних переміщається так зване вікно просіювання. Із пікселів, що потрапляють у його поле зору, відбирається максимальний і переміщується в результуючу матрицю, як це показано на рисунках 15.4 та 15.5. Крок для цього вікна може бути відмінним від одиниці; усе залежить від того, якого розміру вихідну матрицю потрібно одержати, і з яким ступенем потрібно «стиснути» дані.

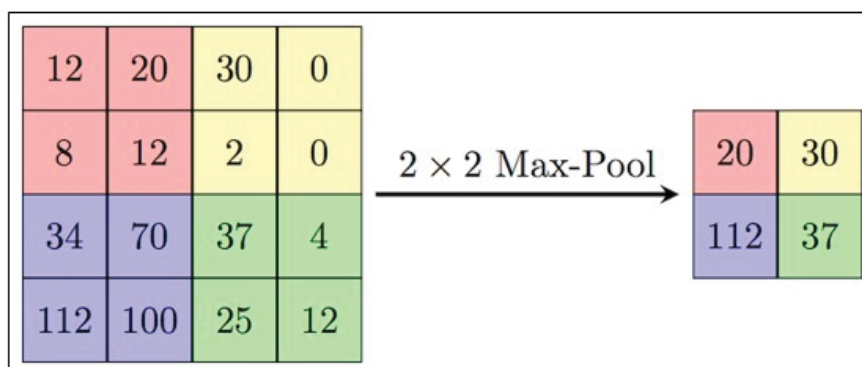


Рисунок 15.4 – Приклад використання процедури максимального об'єднання **Max-pool**

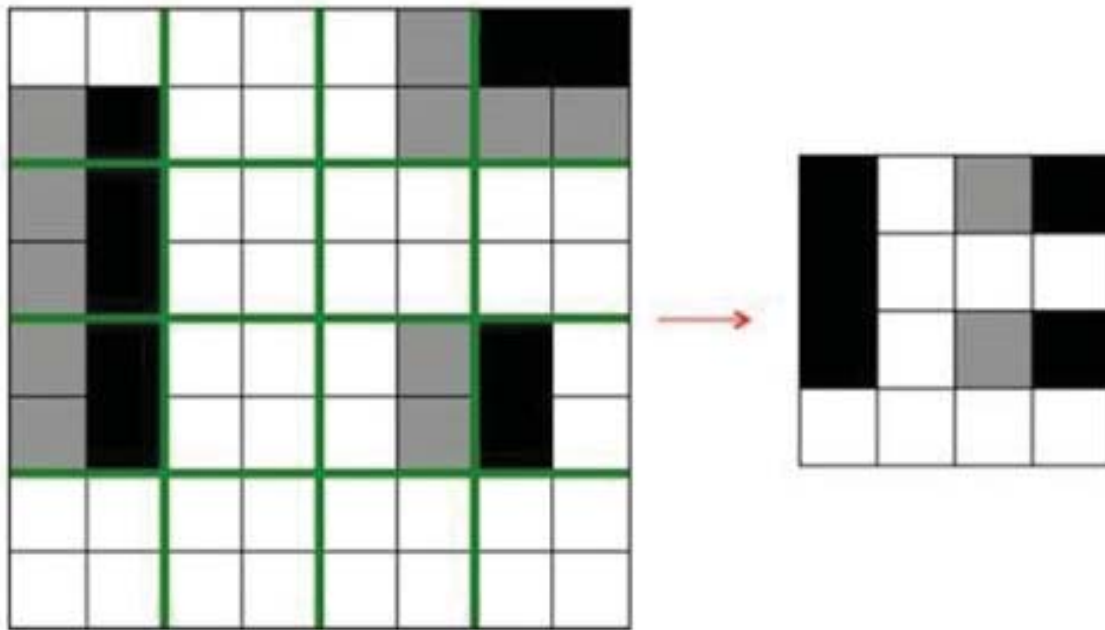


Рисунок 15.5 – Приклад використання процедури максимального об'єднання Max-pool

Необхідно зауважити, що максимальне об'єднання карти ознак робить процес розпізнавання більше точним, очищаючи зображення від «тіней» навколо елементів зображення та скорочуючи кількість параметрів згорткової нейронної мережі.

Отже, фільтри першого згорткового шару виявляють властивості базового рівня, такі як межі та криві. У разі використання декількох згорткових шарів вхідних значенням для другого згорткового шару буде одна або кілька карт ознак – результат оброблення попереднім шаром. Кожен набір вхідних даних визначає позиції, де на вихідному зображенні трапляються певні базові ознаки. Тоді застосовують новий набір фільтрів (пропускається зображення через другий згортковий шар) – на виході будуть одержані нові карти ознак, що демонструють властивості більше високого рівня. Типами таких властивостей можуть бути півкола (комбінація прямої межі з вигином) або квадратів (поєднання кількох прямих ребер). Чим більше згорткових шарів проходить зображення й чим далі воно рухається по мережі, тим більше складні характеристики виводяться в картах активації.

Чергування шарів дозволяє складати карти ознак із карт ознак, що на практиці означає здатність розпізнавання складних ієрархій ознак. Наприкінці мережі можуть бути фільтри, що активуються за наявності рукописного тексту на зображенні за наявності рожевих об'єктів тощо.

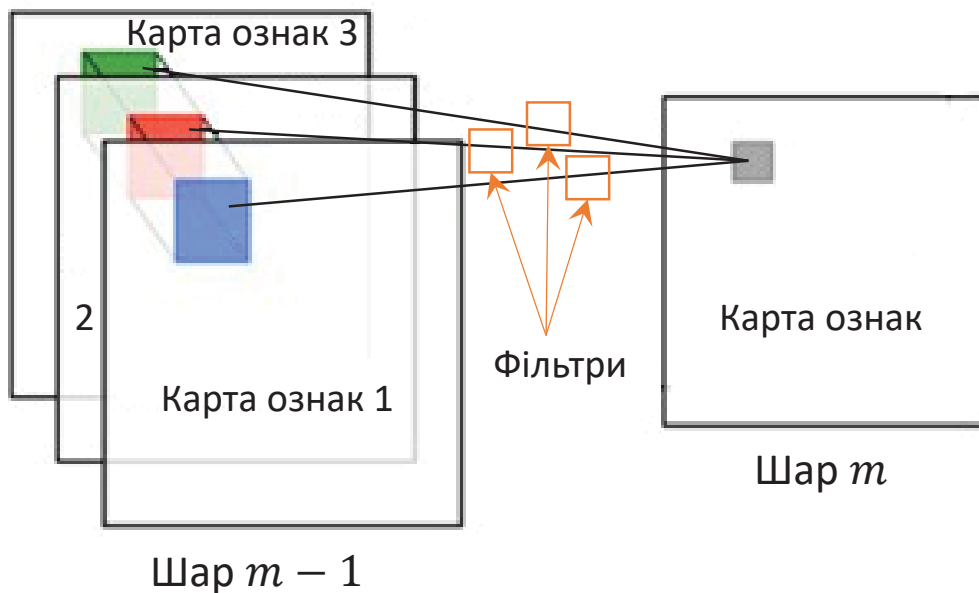


Рисунок 15.6 – Приклад побудови карти ознак

Розглянемо простий приклад розпізнавання обличчя, припускаючи що воно характеризується трьома ознаками: очима, носом та ротом. За допомогою трьох фільтрів, які виділяють ці три ознаки, одержимо три карти ознак, як показано на рисунку 15.6 на шарі $m - 1$. Для того щоб одержати на шарі m карту ознак, що розпізнає обличчя, потрібно одержати інформацію з трьох різних карт ознак, поданих на шарі нейронів ($m - 1$). Це означає, що необхідно створити фільтри, які залежать від усього обсягу інформації та перетинають різні карти ознак у межах шару. Такі зв'язки можуть бути визначені за допомогою повного згорткового шару.

Ще один цікавий момент: під час руху вглиб мережі (розмірності матриць скорочуються), фільтри працюють зі все більшим полем сприйняття, а значить, вони в змозі обробляти інформацію з більшої площі початкового зображення. Простими словами, вони краще пристосовані до оброблення більшої області піксельного простору.

Повнозв'язні шари F-layers.

Після того, як високорівневі властивості мережі виявлено, найважливіше – це прикріплення повнозв'язного шару в кінці мережі для визначення результату роботи мережі. Цей шар бере вхідні дані й виводить N -просторовий вектор, де N – кількість класів з яких мережа обирає необхідний (під час розпізнавання рукописних цифр $N = 10$). Спосіб, за допомогою якого працює повнозв'язний шар – це звернення до виходу попереднього шару та визначення властивостей, які більше пов'язані з певним класом. Наприклад, якщо нейронна мережа повинна передбачати, що на якомусь зображенні собака, то в карт властивостей, які відбивають високорівневі характеристики, такі як лапи або хвіст, повинні бути високі значення. Так само, якщо мережу застосовують для розпізнавання того, що на зображенні птах, у неї будуть високі значення в картах властивостей, репрезентованих високорівневими характеристиками такими як крила або дзьоб. Фактично згорткова мережа складається з двох частин:

- перша частина являє собою комбінацію згорткових та об'єднувальних шарів S-layers, при чому кількість цих шарів та порядок чергування визначає архітектор нейромережі;
- друга частина – звичайна нейромережа прямого поширення з одного або кількох повнозв'язних шарів та вихідного шару.

15.2. Побудова згорткової нейромережі для розпізнавання рукописних цифр

Як приклад побудови згорткової нейронної мережі розглянемо просту модель згорткової мережі з двох згорткових шарів, одного об'єднувального шару Max-pool, одного прихованого повнозв'язного шару та вихідного шару. Незважаючи на її простоту (лише два згорткові шари) така мережа здатна давати близько 98.5 % правильних результатів розпізнавання рукописних цифр під час навчання за набором із бібліотеки `mnist` (див. розділ 2). Структура такої мережі подана на рисунку 15.7.

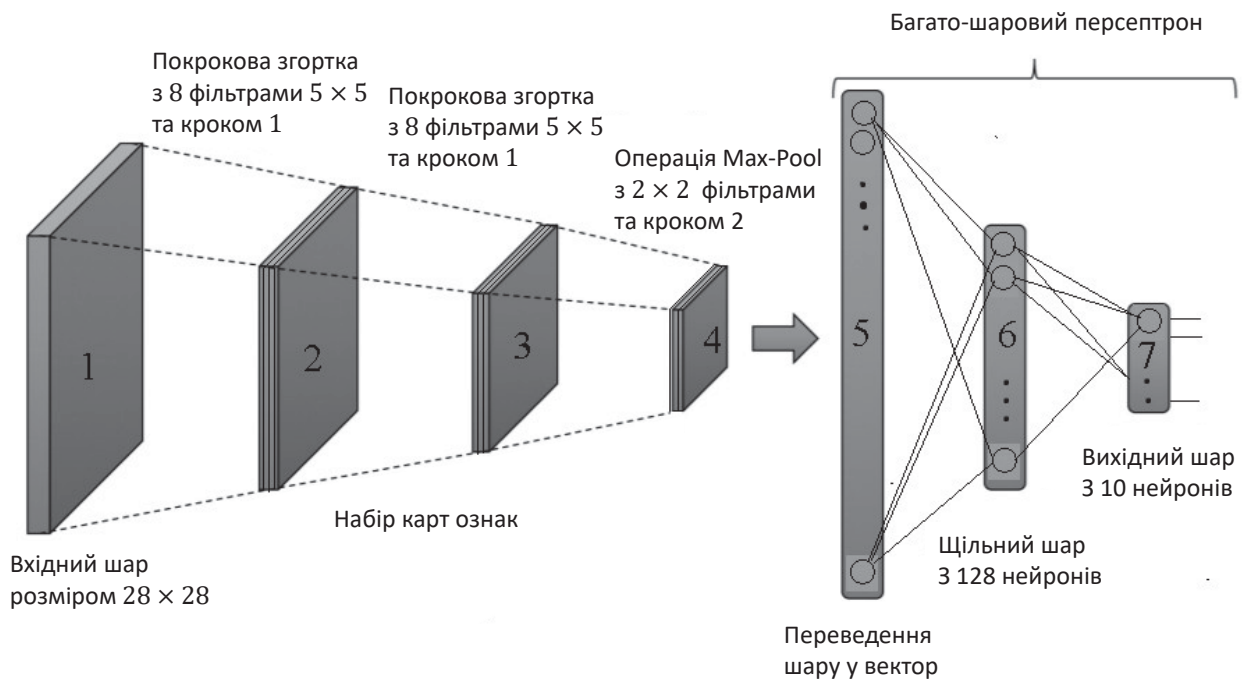


Рисунок 15.7 – Структура згорткової нейронної мережі для завдання розпізнавання рукописних цифр

Для такої мережі алгоритм прямої передачі сигналу буде таким:

- 1) вилучаємо з набору даних (датасета) чергове зображення (або пакет із певною кількістю прикладів);
- 2) перший шар згорткової мережі приймає зображення й на виході віддає набір карт ознак (шар 2 на рисунку 15.7);
- 3) другий шар приймає одержані на попередньому кроці карти та на виході дає інші карти ознак (шар 3 на рисунку 15.7);
- 4) шар максимального об'єднання Max-Pool, який знижує розмірність карт ознак (шар 4 на рисунку 15.7);
- 5) додавання одержаних на попередньому кроці карт та перемішування результату в один вектор-стовпець (шар 5 на рисунку 15.7) (в один стовпець розкладаються всі карти, одержані на виході шару максимального об'єднання Max-Pool, після чого стовпці конкатенуються в один великий стовпець, як показано на рисунку 15.8;

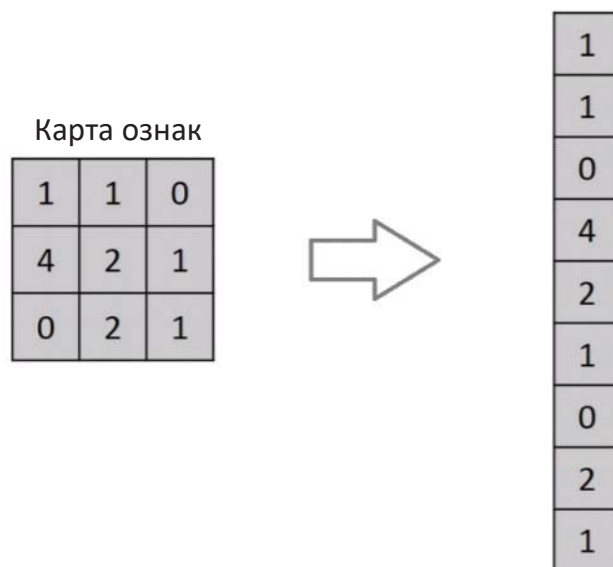


Рисунок 15.8 – Генерація вектора-стовпця з карт ознак

6) перший шар повнозв'язної мережі приймає вектор, проводить обчислення, які дають значення для прихованого повно-зв'язного шару (шар 6 на рисунку 15.7);

7) другий (вихідний) шар повнозв'язної мережі (шар 7 на рисунку 15.7) призначений для формування результату роботи мережі; кількість вихідних нейронів цього шару визначається кількістю класів у використовуваному наборі даних;

8) розрахунок помилки та корегування вагових коефіцієнтів. Далі розглянемо покрокову процедуру передачі сигналу по нейромережі зі структурою 15.7 слідуючи зазначеному вище алгоритму.

15.2.1. Налаштування параметрів згортковій мережі

1. Імпортуємо матрицю навчальних даних (наприклад, 20 000 зразків розміром $d_i \times d_i$), $d_i = 28$, $\mathbf{X} : 20000 \times 784$ та мітки класів (правильні відповіді) $\mathbf{Y} : 20000 \times 1$, елементи масиву – цифри від 0 до 9. Виконуємо попереднє оброблення даних – проводимо нормалізацію та центрування:

$$X = \frac{X - \langle X \rangle}{\langle (\delta X)^2 \rangle},$$

де $\langle X \rangle$ – середнє значення; $\langle (\delta X)^2 \rangle$ – дисперсія.

2. Задаємо архітектуру мережі:

- $n_{f1} = 8$ – кількість фільтрів у першому згортковому шарі (кількість карт на шарі 2 на рисунку 15.7);

- $n_{f2} = 7$ – кількість фільтрів у другому згортковому шарі (кількість карт на шарі 3 на рисунку 15.7);

- $d = 1$ – глибина зображення: чорно-біле зображення (один канал) має глибину 1, для кольорового зображення кількість каналів (глибина зображення) дорівнює 3;

- $f = 5$ – розміри фільтрів (ядер) 5×5 ;

- $f_1 = \{n_{f1}, d, f, f\}$ – параметри фільтрів першого шару;

- $f_2 = \{n_{f2}, d, f, f\}$ – параметри фільтрів другого шару;

- $w_3 = (128, 700)$ – параметри масиву вагових коефіцієнтів першого (прихованого) повнозв'язного шару;

- $w_4 = (10, 128)$ – параметри масиву вагових коефіцієнтів другого (вихідного) повнозв'язного шару.

3. Аналізуємо обрані числові значення для параметрів мережі:

- зображення розміром 28×28 (шар 1 на рисунку 15.7) пропускається через $n_{f1} = 8$ фільтрів; отже, шар 2 на рисунку 15.7 має шаровану структуру та складається з восьми карт ознак;

- якщо фільтр має розмір 5×5 і він переміщується з кроком 1, то на другому шарі карти ознак будуть мати розмір 24×24 (легко перевірити);

- на третьому шарі з тим же фільтром і кроком 1 карти ознак матимуть розмір 20×20 ;

- після пропускання через шар максимального об'єднання Max-pool з кроком 2 на виході будуть одержані стиснені карти розміром 10×10 ;

- оскільки карт 7 (7 фільтрів другого шару), то після їх об'єднання в один вектор-стовпець, його розмірність буде 700×1 (шар 5 на рисунку 15.7);

- у повнозв'язному шарі 128 нейронів – індивідуальний вибір архітектора мережі.

4. Ініціалізуємо вагові коефіцієнти. На початковому етапі всі вагові коефіцієнти зв'язків (зокрема фільтри) розподілені випадковим чином. Коефіцієнти зміщення для кожного шару можна ініціалізувати нулями.

5. Далі запускаємо цикл за епохами навчання. Їх може бути 2 і більше. Упродовж кожної епохи беремо по черзі зразки з масиву Y й пропускаємо через мережу. Одержуємо помилку та за допомогою алгоритму зворотного поширення помилки коригуємо ваги повнозв'язних шарів, фільтри та зміщення (фільтр – це набір вагових коефіцієнтів): масиви $f_1, f_2, w_3, w_4, b_1, b_2, b_3, b_4$.

15.2.2. Прямий хід сигналу по нейронній мережі

Розглянемо процес прямого ходу більше докладно на прикладі першого прикладу:

- $i = 1$ – номер зразка;
- $d = 1$ – кількість каналів;
- $d_i = 28$ – розмір зображення;
- $N = 10$ – кількість цифр.

1. Перетворюємо одномірний масив із 784 рядків у матрицю розміром 28×28 . Масив Y містить мітки класів (цифри від 0 до 9). Нехай пред'явлений i -й зразок містить зображення цифри 5. Отже $Y[i] = 5$. Але останній шар нейромережі містить 10 нейронів – набір нулів з однією одиницею на шостому виході: $[0, 0, 0, 0, 0, 1, 0, 0, 0, 0]$. Для цього виконують відповідні перетворення і замінюють мітку, наприклад 5, на 1:

$$y = 5 \Rightarrow y = [0, 0, 0, 0, 0, 1, 0, 0, 0, 0].$$

2. Пропускаємо зображення через фільтри першого шару. Оскільки фільтрів 8, то в першому згортковому шарі буде 8 карт ознак:

- фіксуємо крок з яким переміщаємо фільтр ($s = 1$);
- розраховуємо розмір карт ознак (24×24);
- створюємо новий масив ($8 \times 24 \times 24$) для зберігання карт ознак;
- переміщаємо фільтр за горизонталлю та вертикаллю (змійкою);
- виконуємо згортку (поелементне множення з подальшим підсумовуванням) і додаємо зміщення (в кожного фільтра є своє (одне) зміщення).

3. Пропускаємо одержані карти через активаційну функцію. У згорткових мережах часто використовують просту функцію ReLU

(Rectified Linear Unit): $f(z) = \max(0, z)$. Її графік має вигляд як показано на рисунку 15.9. Відповідно до цієї функції від'ємні елементи матриць-карт зануляються.

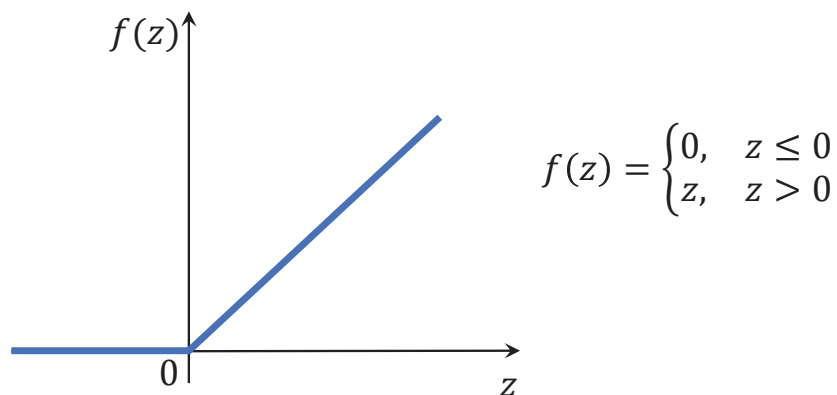


Рисунок 15.9 – Активаційна функція ReLU (Rectified Linear Unit)

4. Використовуючи аналогічну процедуру пропускаємо карти ознак через другий згортковий шар. Кожен фільтр другого шару (їх 7) – це набір із 8 (за кількістю вхідних карт ознак) підфільтрів розміром 5×5 . 8 карт розміром 24×24 пропускаємо через 7 фільтрів другого шару:

- фіксуємо крок із яким переміщаємо фільтр ($s = 1$);
- розраховуємо розмір карт ознак (20×20);
- створюємо новий масив ($7 \times 20 \times 20$) для зберігання карт ознак другого шару;
- переміщаємо фільтр за горизонталлю та вертикаллю (змійкою);
- виконуємо згортку (поелементне множення з подальшим підсумовуванням) і додаємо зміщення (в кожного фільтра є своє (одне) зміщення).

Алгоритм згортки другого шару: беремо черговий фільтр; витягуємо з нього 8 підфільтрів (оскільки 8 карт ознак). Кожен підфільтр накладаємо на відповідну йому карту ознак і виконуємо глобальну згортку. Тим самим пронизуємо всі карти ознак, об'єднуючи їх, як показано на рисунку 15.10. Тут, наприклад, три карти ознак і відповідно у кожному наборі фільтрів другого шару буде по 3 підфільтри. Після згортки одержується одну – нову карту ознак, що відповідає одному фільтру (набору) другого шару.

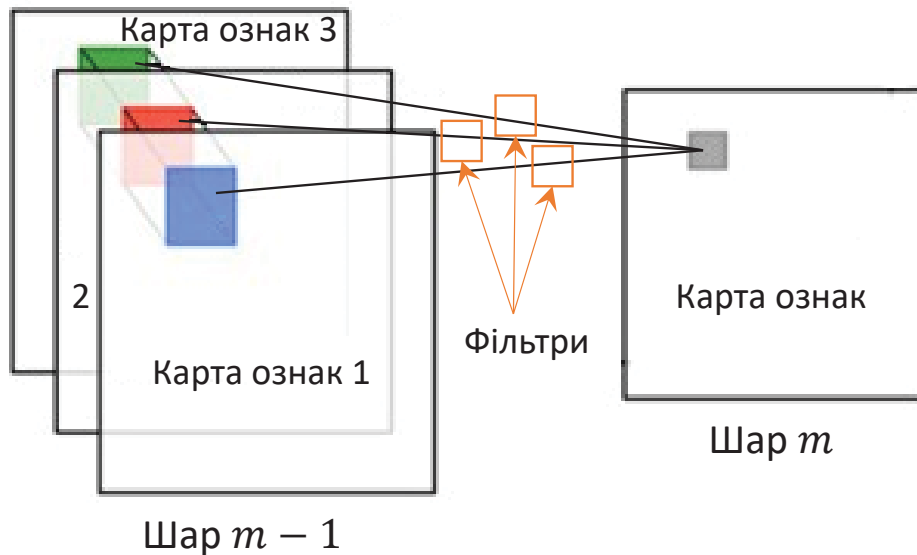


Рисунок 15.10 – Приклад побудови карти ознак.

Цей фільтр використовують для сканування всієї області карт ознак та після згортки одержують нову (одну) карту ознак розміром 20×20 . Далі беруть наступний фільтр у якого теж 8 своїх підфільтрів і процедуру повторюють.

На другому згортковому шарі існує 7 фільтрів, а отже, буде сформовано сім нових карт ознак розміром 20×20 кожна. Але ці карти, за рахунок об'єднання (див. рис. 15.10), на відміну від карт першого шару, містять більш високорівневі елементи вихідного зображення.

5. Пропускаємо держані карти через активаційну функцію ReLU.

6. Зменшуємо розмір карт ознак (шар 4 на рис. 15.7), проводячи субдискретизацію (максимальне об'єднання – Max-pool).

7. Розгортаємо всі карти в один вектор-стовпець, послідовно одна за одною.

8. Подаємо одержаний сигнал на перший повнозв'язний шар із 128 нейронів.

9. Перемножуємо на вагові коефіцієнти й підсумовуємо за допомогою матричного множення та додаємо вектор коефіцієнтів зміщення.

10. Пропускаємо результат через активаційну функцію ReLU та одержуємо вихід першого повнозв'язного шару із 128 нейронів.

11. Подаємо сигнал на вихідний повнозв'язний шар-стовпець розміром 10×1 .

12. Укінці пропускаємо сигнал через активаційну функцію. Сигмоїду використовують лише якщо класів (для завдання класифікації) не більше двох: вихід моделі буде числом від нуля (перший клас) до одиниці (другий клас). Для більшої кількості класів, щоб вихід моделі відбивав імовірність цих класів (і сума імовірностей з виходів мережі дорівнювала одиниці), використовують функцію `SoftMax`

$$\hat{y}_i = f(z_i) = \frac{\exp(z_i)}{\sum_{k=1}^N \exp(z_k)}. \quad (15.1)$$

13. Завершальний етап – розрахунок якості роботи мережі. Для згорткових мереж як функцію втрат (функцію помилки) зазвичай використовують не сумарну різницю між виходом і бажаним результатом, як у прикладі мереж прямого поширення помилки, а крос-ентропію

$$C = - \sum_{k=1}^N y_k \ln(\hat{y}_k), \quad (15.2)$$

де y_k та \hat{y}_k це відповідно відомий та одержаний мережею результати.

15.2.3. Зворотний хід: корегування значень вагових коефіцієнтів

Згорткова мережа навчається за алгоритмом зворотного поширення помилки. Ідея методу докладно розглянута в розділі 2 цього посібника. Якщо розрахунок помилок у повнозв'язній частині мережі є тривіальним (його було розглянуто раніше), то розрахунок помилок на згорткових шарах є доволі заплутаним. Розглянемо цей процес більш детально.

1. Починається зворотний шлях із коригуванням вагових коефіцієнтів. Вагові коефіцієнти w_{ij} та зміщення b_i повнозв'язних шарів коригуються за (6.7–6.15), одержаними в розділі 2, відповідно до алгоритму зворотного поширення помилки, що ґрунтується на методі градієнтного спуску. Отже,

$$w_{ij} = w_{ij} - \eta \frac{\partial C}{\partial w_{ij}}. \quad (15.3)$$

Розраховуємо $\partial C / \partial w_{ij}$

$$\frac{\partial C}{\partial w_{ij}} = \frac{\partial C}{\partial z_j} \frac{\partial z_j}{\partial w_{ij}}. \quad (15.4)$$

Для другого множника маємо

$$\frac{\partial z_j}{\partial w_{ij}} = z_i^j, \quad (15.5)$$

де z_i^j – сигнал, що надійшов на нейрон j з i -го виходу попереднього шару.

Оскільки функція **SoftMax** містить суму по всіх нейронах вихідного шару, відповідно множник $\frac{\partial C}{\partial z_j}$ набуде вигляду

$$\frac{\partial C}{\partial z_j} = \sum_{k=1}^n \frac{\partial C}{\partial \hat{y}_k} \frac{\partial \hat{y}_k}{\partial z_j}, \quad (15.6)$$

де

$$\frac{\partial C}{\partial \hat{y}_j} = -\frac{y_j}{\hat{y}_j},$$

$$\frac{\partial \hat{y}_j}{\partial z_i} = \begin{cases} \frac{\exp(z_j)}{\sum_{k=1}^n \exp(z_k)} - \left(\frac{\exp(z_j)}{\sum_{k=1}^n \exp(z_k)} \right)^2 = \hat{y}_j(1 - \hat{y}_j), & i = j; \\ -\frac{\exp(z_j) \exp(z_i)}{\left(\sum_{k=1}^n \exp(z_k) \right)^2} = -\hat{y}_i \hat{y}_j, & i \neq j. \end{cases} \quad (15.7)$$

Відповідно

$$\frac{\partial C}{\partial z_j} = -y_j(1 - \hat{y}_j) + \sum_{k \neq j} y_k \hat{y}_j = -y_j + \hat{y}_j \sum_{k=1}^n y_k = \hat{y}_j - y_j, \quad (15.8)$$

оскільки $\sum_k y_k = 1$. Остаточо маємо

$$\frac{\partial C}{\partial w_{ij}} = (\hat{y}_j - y_j) z_i^j. \quad (15.9)$$

Використовуючи формулу (15.9), обчислюємо градієнт для всіх вагових коефіцієнтів зовнішнього (вихідного) шару.

2. Далі, використовуючи формули (6.7–6.15), одержані в розділі 2, розрахуємо помилку δ на першому прихованому шарі зі 128 нейронів. Схематично поширення помилки від зовнішнього до прихованого повнозв'язного шару показано на рисунках 6.12–6.13.

3. Розраховуємо помилку на шарі зі 700 нейронів та збираємо назад у тривимірний масив розміром $7 \times 10 \times 10$ (7 карт розміром 10×10).

4. Тепер завдання протягнути набір помилок δ назад через шар субдискретизації (шар 4 на рисунку 15.7), щоб дістатися до фільтрів та скоригувати їх.

Помилка «проходить» лише через ті значення вихідної матриці, що були обрані максимальними на кроці максимального об'єднання Max-pool. Інші значення помилки для матриці будуть рівні нулеві, як це схематично показано на рисунку 15.11.

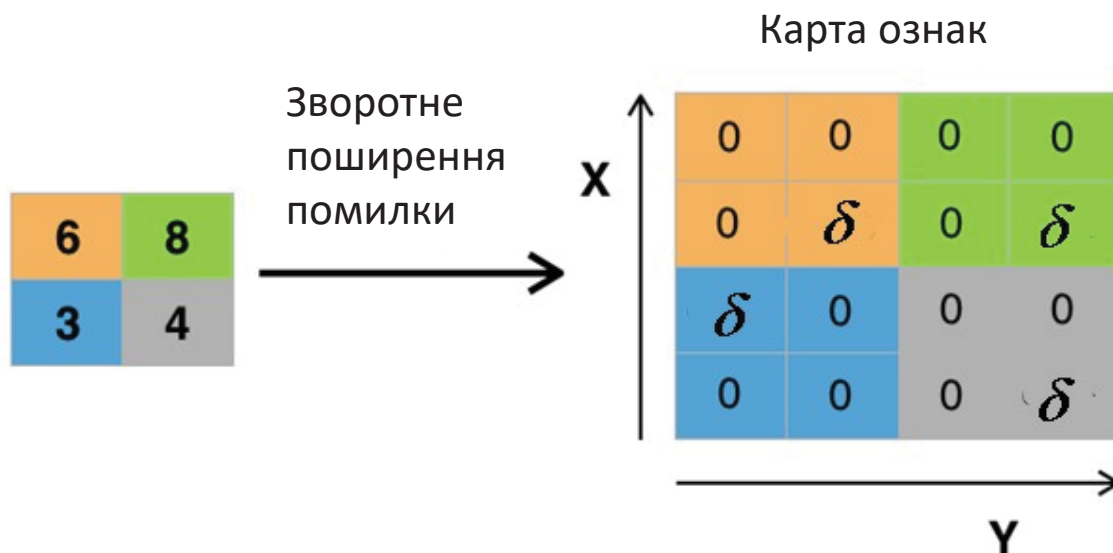


Рисунок 15.11 – Схематичне подання процесу «протягування» помилки через шар Max-pool.

Такий вибір є обґрунтованим, оскільки значення за цими елементами не були обрані функцією Max-pool під час прямого проходження через мережу й відповідно ніяк не вплинули на підсумковий результат. Операція протягування помилки нагадує просіювання піску через сито з частково закритими комірками. Шар Max-pool не впливає на помилку, а лише перенаправляє помилку в потрібні комірки.

5. Далі коригують фільтри (вагові коефіцієнти зв'язків між картами першого згорткового шару та картами другого згорткового шару), з використанням тієї самої формули

$$w_{ij} = wij - \eta \frac{\partial C}{\partial w_{ij}}. \quad (15.10)$$

Розглянемо процес коригування більш детально. Якщо би фільтр мав розмір 20×20 , він цілком би лягав на карту розміром 20×20 і, в результаті, вийшла б звичайна нейронна мережа, де кожен нейрон попереднього шару (попередньої карти) пов'язаний із кожним нейроном наступного шару (нової карти). Усі вагові коефіцієнти в такій мережі унікальні. У такому разі для коригування вагових коефіцієнтів ми б скористалися формулою

$$\frac{\partial C}{\partial w_{ij}} = \frac{\partial C}{\partial z_j} \frac{\partial z_j}{\partial w_{ij}}, \quad (15.11)$$

де перший множник – локальний градієнт (який був протягнутий через шар субдискретизації з першого повнозв'язного шару), а другий множник – вихід нейрона i попереднього шару (15.9), проте фільтр має розмір 5×5 . Крім того, він ковзає по карті ознак із кроком s . Отже, один і той самий зв'язок (ваговий коефіцієнт) будуть застосовувати до різних нейронів карти, якою ковзає фільтр. Більше того, якщо крок невеликий (наприклад, 1), то сусідні області карти, до яких застосовують фільтр, перекриваються. Отже, у формулу для розрахунку $\partial C / \partial w_{ij}$ необхідно ввести суму за всім можливим положенням фільтра на карті ознак

$$\begin{aligned} \frac{\partial C}{\partial w_{ij}} &= \sum_q \sum_w \frac{\partial C}{\partial \hat{y}_{qw}} \frac{\partial \hat{y}_{qw}}{\partial z_{qw}} \frac{\partial z_{qw}}{\partial w_{ij}} \\ &= \sum_q \sum_w \frac{\partial C}{\partial \hat{y}_{qw}} \frac{\partial \hat{y}_{qw}}{\partial z_{qw}} \frac{\partial \left(\sum_r \sum_t w_{rt} z_{(q \cdot s - r)(w \cdot s - t)}^j + b \right)}{\partial w_{ij}} \\ &= \sum_q \sum_w \frac{\partial C}{\partial \hat{y}_{qw}} \frac{\partial \hat{y}_{qw}}{\partial z_{qw}} z_{(q \cdots - r)(w \cdot s - t)}^j; \\ \frac{\partial C}{\partial b} &= \sum_q \sum_w \frac{\partial C}{\partial \hat{y}_{qw}} \frac{\partial \hat{y}_{qw}}{\partial z_{qw}}. \end{aligned} \quad (15.12)$$

Отже, множник $\frac{\partial C}{\partial \hat{y}_{qw}} \frac{\partial \hat{y}_{qw}}{\partial z_{qw}}$ – локальний градієнт, що «перетягується з попереднього шару».

Для побудованої мережі для однієї карти ознак існує одне значення коефіцієнта зміщення, пов'язаного з усіма елементами цієї карти. Відповідно під час коригування значення коефіцієнта зміщення потрібно враховувати всі значення з карти, одержані під час зворотного поширення помилки. Як альтернативний варіант можна використовувати кількість параметрів зміщення, що дорівнює кількості елементів у цій карті, але в такому разі параметрів зміщення буде занадто багато – більше, ніж параметрів самих ядер згортання. Далі роблять аналогічну процедуру для розрахунку величини для коригування фільтрів та зміщення першого шару.

На цьому зворотній хід завершено. За допомогою одержаних похідних коригують вагові коефіцієнти та зміщення за формулою (15.3). Потім беруть наступне зображення та знову пропускаються через мережу. Прискорити процес можна, використовуючи пакетне навчання.

15.3. Лістинг коду для побудови мережі

Ініціалізація

Вилучаємо навчальні дані (наприклад, 20 000 зразків) із файлу «1» та мітки класів (правильні відповіді) з файлу «2» (extract – умовне позначення для функції вилучення даних).

```
X = extract ('1', 20000, 28) # Розмір X: 20000x784. Розмір
                             зображення 28x28 = 784.
Y = extract ('2', 20000) # Розмір Y: 20000x1. Елементи
                           масиву - цифри від 0 до 9
```

Виконуємо попереднє оброблення даних – проводимо нормалізацію, перетворюємо одномірний масив із 784 рядків у матрицю, переформатовуємо цільові значенні у вектори-стовпці.

```
i = 1 # номер зразка
img_d = 1 # кількість каналів
img_s = 28 # розмір зображення
num_classes = 10 # кількість цифр
X -= int(np.mean(X))
X /= int(np.std(X))
x = X[i].reshape(img_d, img_s, img_s) # X: [1x784]->[28x28]
y = np.eye(num_classes)[int(Y[i])].reshape(num_classes, 1)
```

Задаємо архітектуру мережі

```
num_filt1 = 8 # кількість фільтрів у 1-ому шарі
num_filt2 = 7 # кількість фільтрів у 2-ому шарі
f = 5 # розміри фільтрів (ядер) 5x5
f1=(num_filt1, img_d, f, f) # Фільтри 1-ого шару
f2=(num_filt2, num_filt1, f, f) # Фільтри 2-ого шару
w3=(128,700) # ваги першого повнозв'язного шару.
w4 = (10, 128) # ваги другого повнозв'язного шару.
```

Ініціалізуємо вагові коефіцієнти та зміщення.

```
def initializeFilter(size, scale = 1.0):
    stddev = scale / np.sqrt(np.prod(size))
    return np.random.normal(loc=0, scale=stddev, size=size)

def initializeWeight(size):
    return np.random.standard_normal(size = size)*0.01
```

```

f1 = initializeFilter(f1)
f2 = initializeFilter(f2)
w3 = initializeWeight(w3)
w4 = initializeWeight(w4)

b1 = np.zeros((f1.shape[0],1))
b2 = np.zeros((f2.shape[0],1))
b3 = np.zeros((w3.shape[0],1))
b4 = np.zeros((w4.shape[0],1))

```

Прямий хід

Пропускаємо зображення $X[i]$ через фільтри першого шару (8 фільтрів генерує 8 карт ознак).

```

s=1 # крок із яким переміщуємо фільтр
(n_f, n_c_f, f, _) = f1.shape # результат (8,1,5,5)
n_c, in_dim, _ = x.shape # результат (1,28,28)
out_dim = int((in_dim - f)/s) + 1 # результат 24
out = np.zeros((n_f, out_dim, out_dim)) # створюємо новий
                                         масив розміром (8x24x24) для
                                         зберігання карт ознак

for curr_f in range(n_f):
    curr_y = out_y = 0
    # переміщаємо фільтр за горизонталлю та вертикаллю
    while curr_y + f <= in_dim:
        curr_x = out_x = 0
        while curr_x + f <= in_dim:
            # виконуємо згортку
            out[curr_f, out_y, out_x] =
                np.sum(f1[curr_f]*x[:,curr_y:curr_y + f,
                    curr_x:curr_x + f]) + b1[curr_f]

            curr_x += s
            out_x += 1
            curr_y += s
            out_y += 1
conv1 = out
conv1[conv1<=0] = 0 # пропускаємо одержані карти через
                    активаційну функцію

```

Пропускаємо одержані карти через другий згортковий шар.

```
s=1 # крок із яким переміщуємо фільтр
(n_f, n_c_f, f, _) = f2.shape # результат (7,8,5,5)
n_c, in_dim, _ = conv1.shape # результат (8,24,24)
out_dim = int((in_dim - f)/s) + 1 # результат 20
out = np.zeros((n_f, out_dim, out_dim)) # масив (7x20x20)
                                     для зберігання карт ознак
for curr_f in range(n_f): #
    curr_y = out_y = 0
# переміщаємо фільтр по горизонталі та вертикалі
    while curr_y + f <= in_dim:
        curr_x = out_x = 0
        while curr_x + f <= in_dim:
            out[curr_f, out_y, out_x] =
                np.sum(f2[curr_f]*conv1[:,curr_y:curr_y + f,
                    curr_x:curr_x + f]) + b2[curr_f]

            curr_x += s
            out_x += 1
        curr_y += s
        out_y += 1
conv2 = out
conv2[conv2<=0] = 0 # пропускаємо одержані карти через
                    активаційну функцію
```

Зменшуємо розмір карт ознак (Max-pool).

```
f = 2 # крок переміщення вікна за картами ознак по x
s = 2 # крок переміщення вікна за картами ознак по y
n_c, h_prev, w_prev = conv2.shape
h = int((h_prev - f)/s) + 1 # розміри карт
w = int((w_prev - f)/s) + 1 # після стиснення
pooled = np.zeros((n_c, h, w))
for i in range(n_c):
    curr_y = out_y = 0
    while curr_y + s <= h_prev:
        curr_x = out_x = 0
        while curr_x + f <= w_prev:
            pooled[i, out_y, out_x] =
                np.max(conv2[i, curr_y:curr_y+s, curr_x:curr_x+f])
            curr_x += f
            out_x += 1
        curr_y += s
        out_y += 1
(nf2, dim2, _) = pooled.shape
```

Пропускаємо карти ознак через повнозв'язні шари.

```
fc = pooled.reshape((nf2 * dim2 * dim2, 1)) # розгортаємо
      всі карти в один стовпець,
      послідовно одна за одною.
      Розмір fc : 700x1.
z = w3.dot( fc ) + b3 # подаємо одержаний сигнал на перший
      повнозв'язний шар. z - вектор
      стовпець розміром 128x1.
z[z<=0] = 0 # пропускаємо через активаційну функцію ReLU і
      одержуємо вихід першого
      повнозв'язного шару з 128
      нейронів
out = w4.dot(z) + b4 # подаємо сигнал на вихідний повнозв'
      язний шар. out - стовпець
      розміром 10x1.
out = np.exp(out) # пропускаємо сигнал через активаційну
      функцію softmax
probs = out / np.sum(out) # на виході одержуємо відповідь
      мережі
loss = - np.sum(y * np.log(probs)) # розраховуємо помилку
```

Зворотний хід

Обчислюємо градієнт для всіх вагових коефіцієнтів зовнішнього шару.

```
dout = probs - y # помилка на кожному нейроні
dw4 = dout.dot(z.T) # розраховуємо градієнти всіх вагових
      коефіцієнтів вихідного шару
      за формулою. z - вектор
      стовпець розміром (128,1) -
      сигнал на виході повнозв'
      язного шару
# dw4 має розмір (10,1) x (1,128) = (10x128)
db4 = np.sum(dout, axis = 1).reshape(b4.shape) # рахуємо
      помилку на параметрі зміщення
      Розмір 10x1
```


Розраховуємо помилку на першому прихованому шарі.

```
dz = w4.T.dot(dout) # сумуємо добутки всіх вихідних помилок
                    # на вагові коефіцієнти
                    # відповідних нейронів.
                    # Розмірність dz 128x1
dz [z<=0] = 0 # множимо на похідну від функції ReLU =1,
              # якщо x>0, або нуль, якщо x<0.
# обчислюємо градієнт для вагових коефіцієнтів та зміщень
# першого повнозв'язного шару
# із 128 нейронів
dw3 = dz.dot(fc.T) # розмір dw3: 128x700
db3 = np.sum(dz, axis = 1).reshape(b3.shape) # розмір db3:
      128x1
dfc = w3.T.dot(dz) # розраховуємо помилку на шарі з 700
                  # нейронів. Розмір dfc : 700x1
dpool = dfc.reshape(pooled.shape) # збираємо назад у
                                   # тривимірний масив 7x10x10 (7
                                   # карт розміром 10x10)
```

Протягування помилок (dpool) через шар Max-pool.

```
def nanargmax(arr):
    idx = np.nanargmax(arr)
    idxs = np.unravel_index(idx, arr.shape)
    return idxs

f = 2
s = 2
(n_c, orig_dim, _) = conv2.shape
dout = np.zeros(conv2.shape)
for curr_c in range(n_c):
    curr_y = out_y = 0
    while curr_y + f <= orig_dim:
        curr_x = out_x = 0
        while curr_x + f <= orig_dim :
            (a, b) = nanargmax(conv2[curr_c, curr_y: curr_y + f,
                                     curr_x: curr_x + f])
            dout[curr_c, curr_y + a, curr_x + b] = dpool[curr_c,
                                                           out_y, out_x]

            curr_x += s
            out_x += 1
        curr_y += s
        out_y += 1
dconv2 = dout # помилка. Розмір 7x20x20
dconv2[conv2<=0] = 0 # множимо на похідну від функції ReLU
```

Коригуємо фільтри (вагові коефіцієнти зв'язків між картами першого та другого згорткового шару).

```
s=1
(n_f, n_c, f, _) = f2.shape
(_, orig_dim, _) = dconv2.shape
dout = np.zeros(conv1.shape)
dfilt = np.zeros(f2.shape)
dbias = np.zeros((n_f, 1))
for curr_f in range (n_f):
    curr_y = out_y = 0
    while curr_y + f <= orig_dim:
        curr_x = out_x = 0
        while curr_x + f <= orig_dim:
            dfilt[curr_f] += dconv2[curr_f, out_y, out_x]
                * conv1[:, curr_y:curr_y + f, curr_x:curr_x + f]
            dout[:, curr_y:curr_y + f, curr_x:curr_x + f] +=
                dconv2[curr_f, out_y, out_x] * f2[curr_f]
            curr_x += s
            out_x += 1
        curr_y += s
        out_y += 1
# loss gradient of the bias
dbias [ curr_f ] = np.sum (dconv2[ curr_f ])

df2= dfilt # одержали коригування для фільтрів
db2= dbias # коригування для зміщення
dconv1 = dout # помилка на першому згортковому шар. Розмір
                8x24x24
dconv1[conv1<=0] = 0 # локальний градієнт: помилку
                    помножена на похідну від
                    активаційної функції
```

Коригування фільтрів та зміщення першого шару.

```
s=1
(n_f, n_c, f, _) = f1.shape
(_, orig_dim, _) = x.shape
dout = np.zeros(x.shape)
dfilt = np.zeros(f1.shape)
dbias = np.zeros((n_f, 1))
for curr_f in range(n_f):
    curr_y = out_y = 0
    while curr_y + f <= orig_dim :
        curr_x = out_x = 0
        while curr_x + f <= orig_dim :
            dfilt[curr_f] += dconv1[curr_f, out_y, out_x]
                * x[:, curr_y:curr_y + f, curr_x:curr_x + f]
            curr_x += s
            out_x += 1
        curr_y += s
        out_y += 1
    dbias[curr_f] = np.sum(dconv1[curr_f])

df1 = dfilt
db1= dbias
```

За допомогою одержаних $df1$, $db1$, $df2$, $db2$, $dw3$, $db3$, $dw4$, $db4$ проводять коригування вагових коефіцієнтів зв'язків.

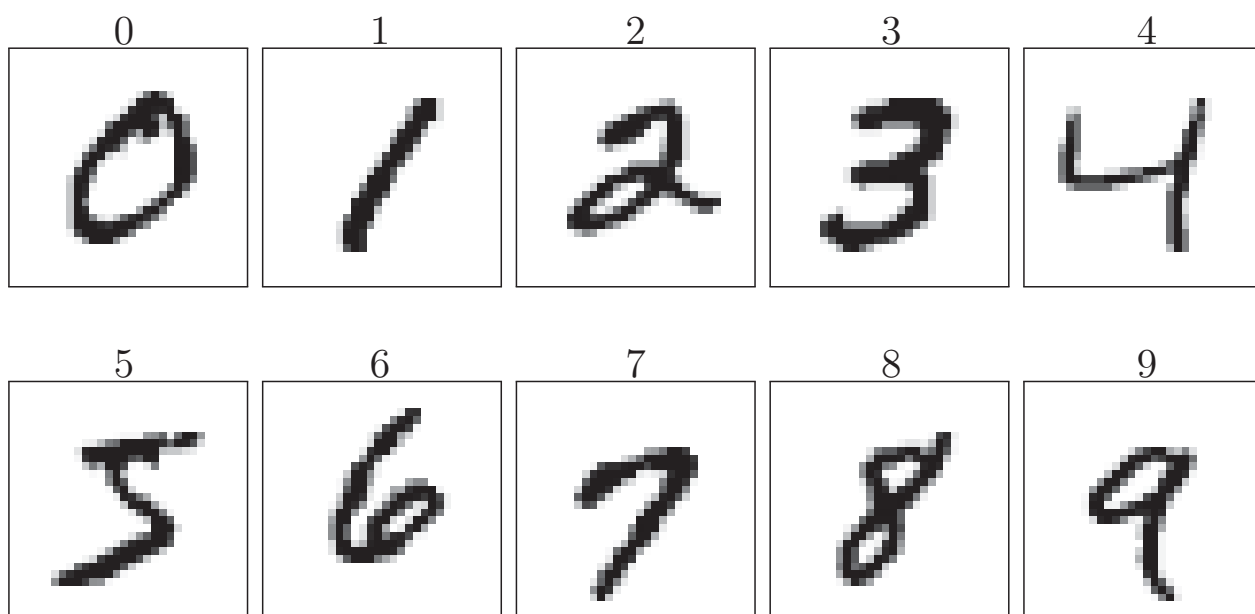
```
f1 = f1 - eta * df1
f2 = f2 - eta * df2
w3 = w3 - eta * dw3
w4 = w4 - eta * dw4

b1 = b1 - eta * db1
b2 = b2 - eta * db2
b3 = b3 - eta * db3
b4 = b4 - eta * db4
```

15.4. Застосування побудованої мережі для розпізнавання зображень

Як приклад використання побудованої згорткової мережі для завдання розпізнавання зображень розглянемо три набори даних:

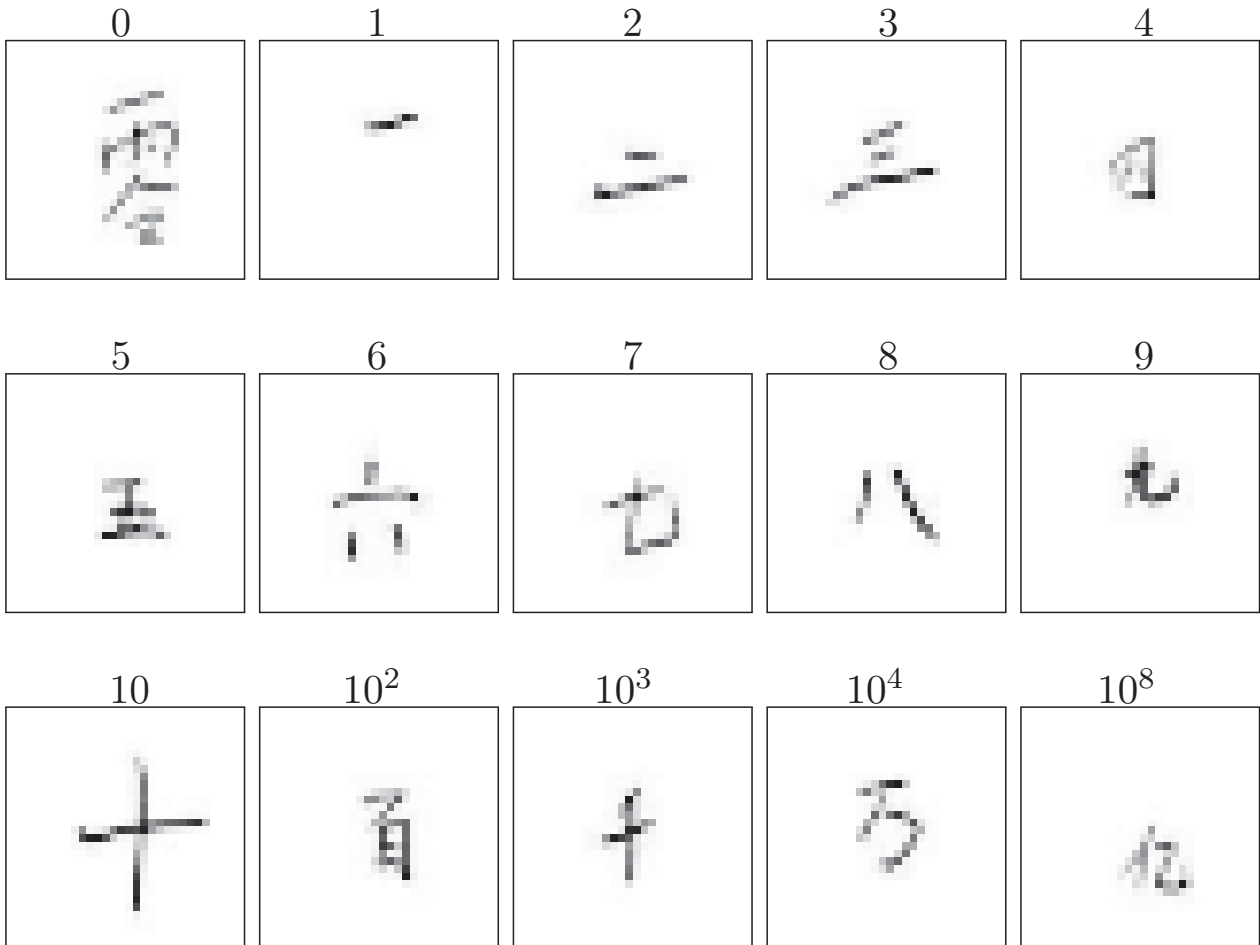
- зображення рукописних арабських цифр з бібліотеки `mnist`;



- зображення одягу та взуття з бібліотеки `mnist`;



• зображення рукописних китайських ієрогліфів цифр з бібліотеки `mnist`.

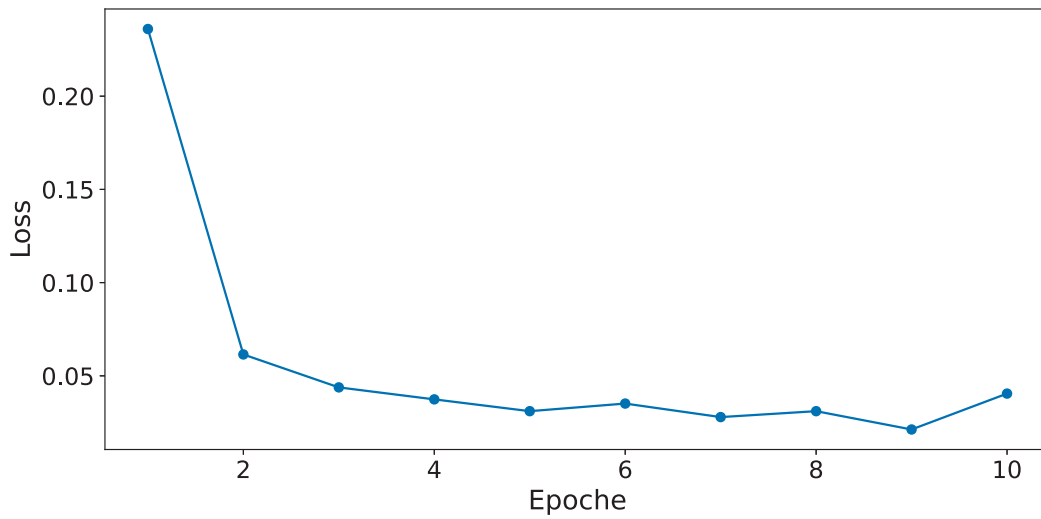


Кожне зображення в усіх трьох наборах переведемо до розміру 28×28 . Для зображень рукописних арабських цифр та зображень одягу і взуття набір навчальних даних містить 60 000 екземплярів, тоді як тестовий набір містить 10 000 екземплярів. Набір зображень рукописних китайських ієрогліфів цифр містить всього 15 000 екземплярів по 100 зображень для кожного числа. Для цього набору всю вибірку розділимо на навчальну та тестову у пропорції 80 % на 20 %.

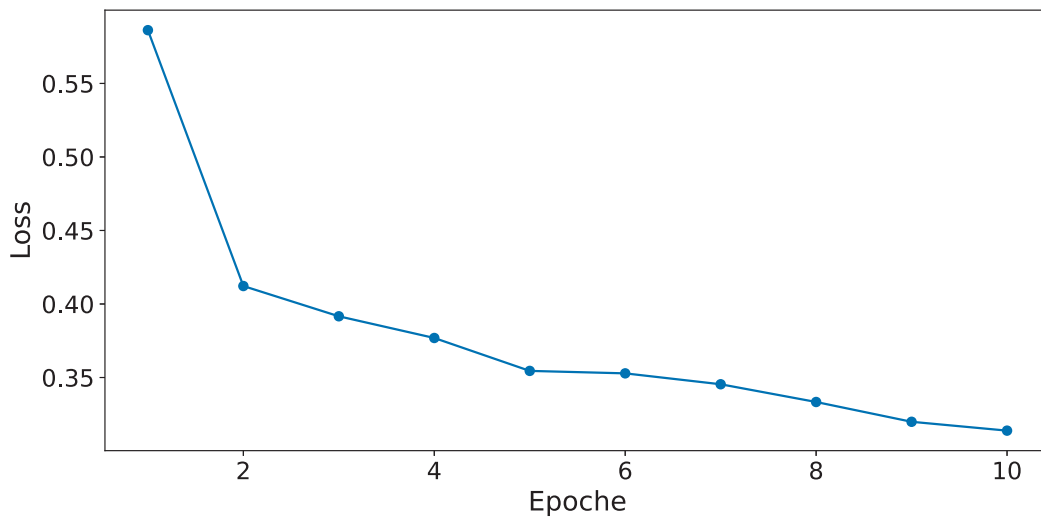
Структура мережі буде однаковою для всіх трьох наборів з різною кількістю нейронів вихідного шару: 10 – для зображень рукописних арабських цифр та зображень одягу і взуття; 15 – для зображень рукописних китайських ієрогліфів цифр. Навчання будемо проводити упродовж 10 епох для всього набору відповідних навчальних прикладів.

Залежність втрат від епохи у процесі навчання

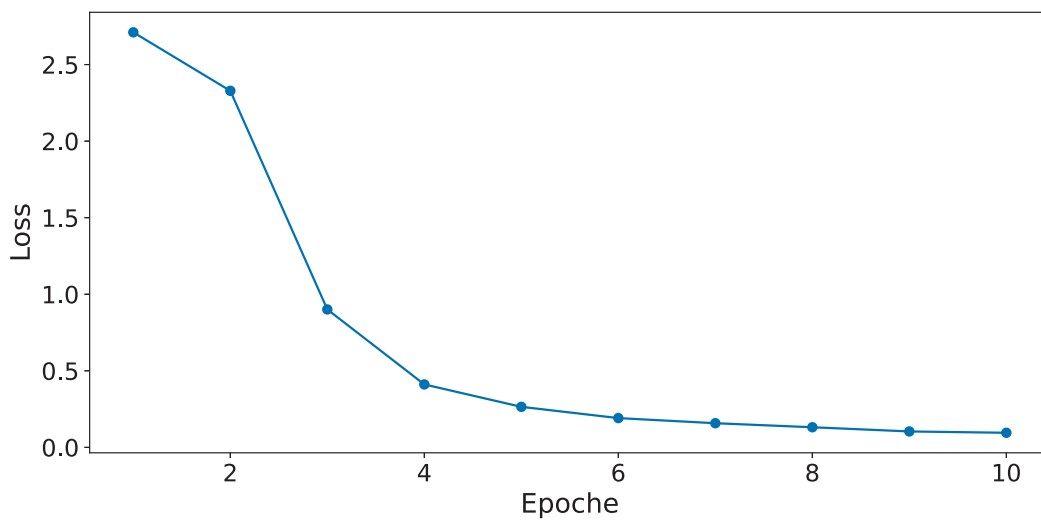
Зображення рукописних арабських цифр



Зображення одягу та взуття

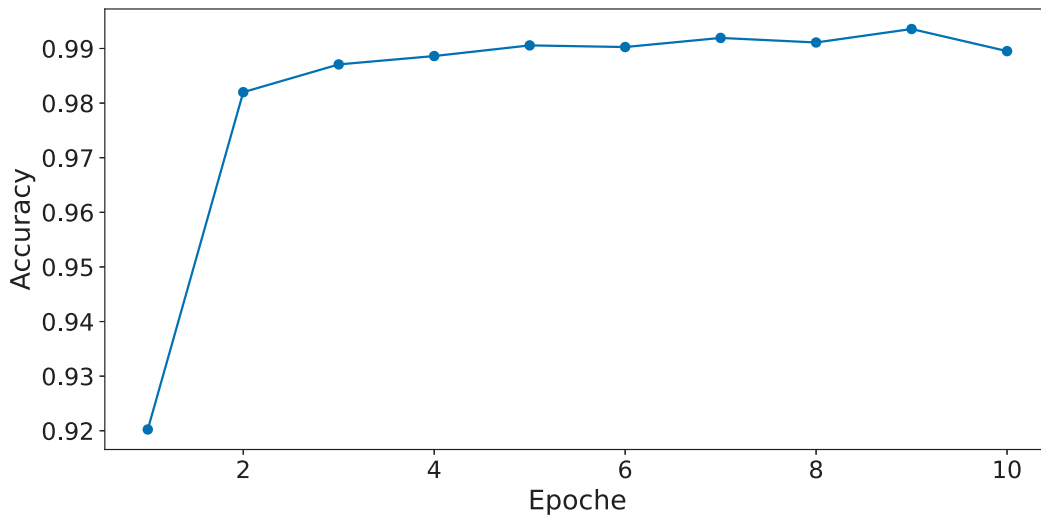


Зображення рукописних китайських ієрогліфів цифр

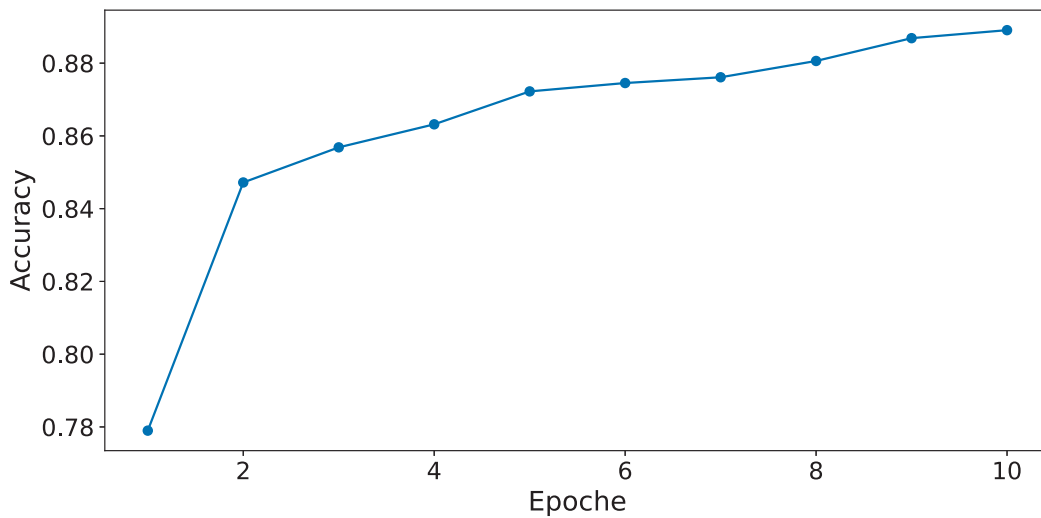


Залежність якості розпізнавання від епохи навчання

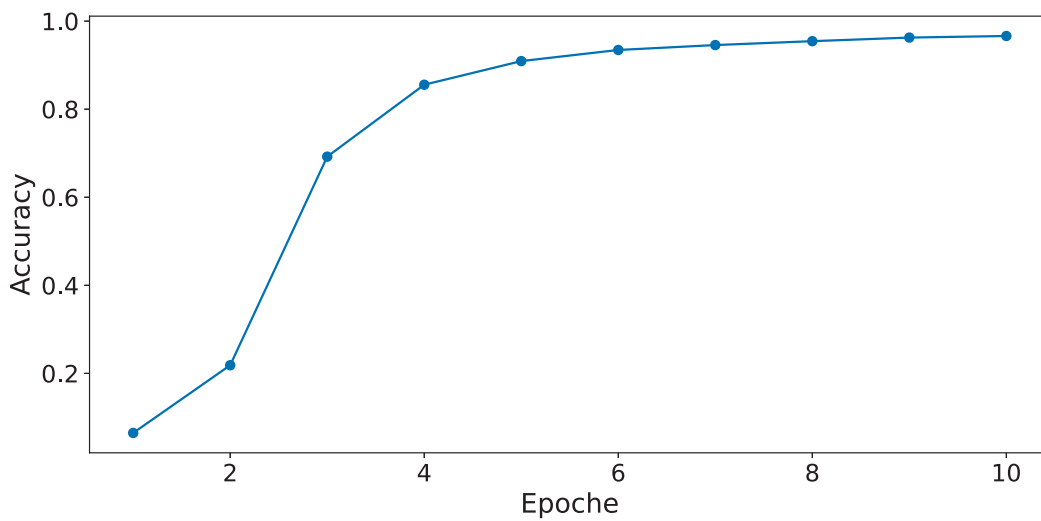
Зображення рукописних арабських цифр



Зображення одягу та взуття

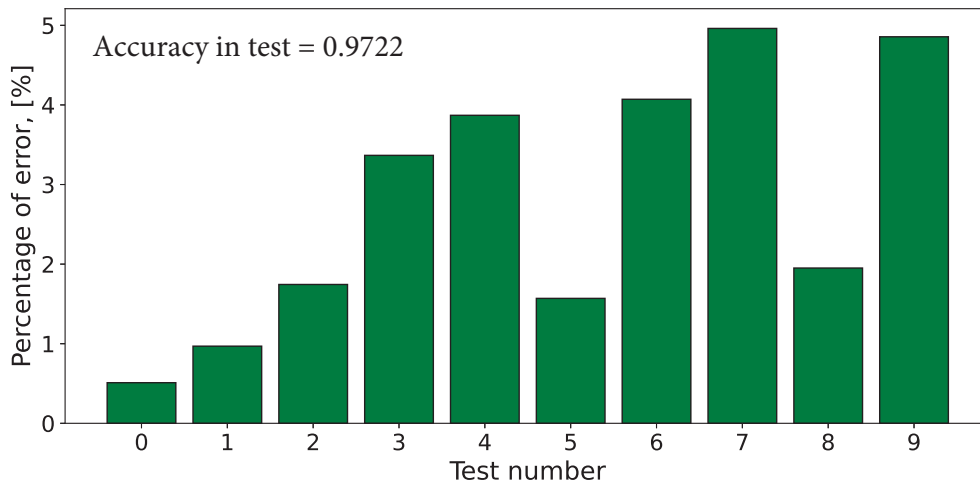


Зображення рукописних китайських ієрогліфів цифр

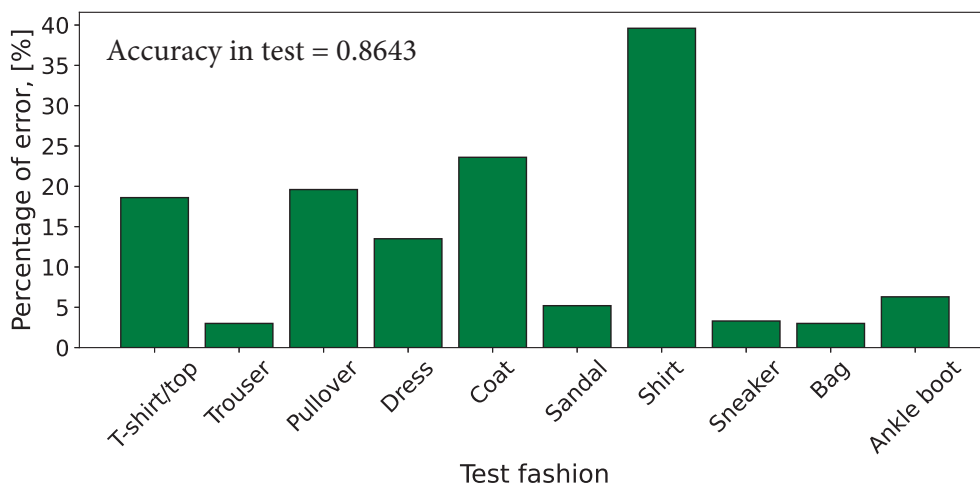


Перевіряємо роботу мережі на тестовій вибірці: аналізуємо кількість помилок для кожного класу для кожного набору.

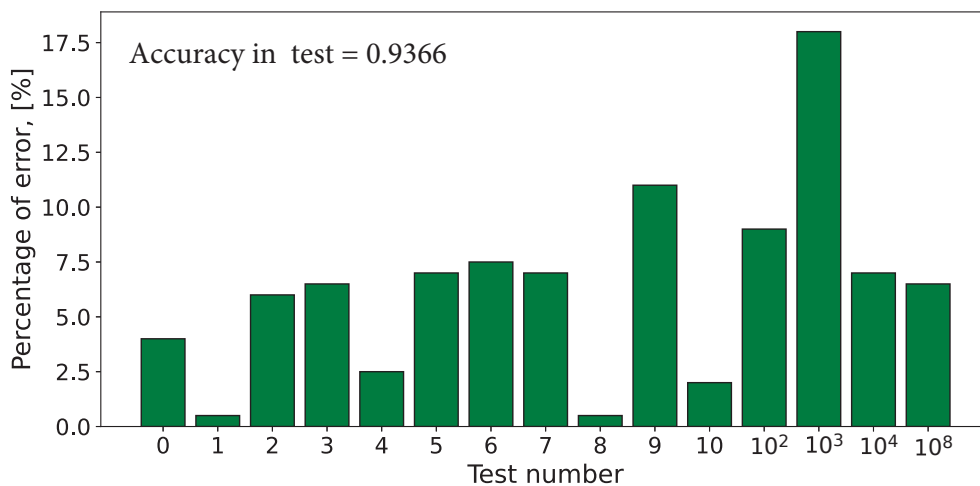
Зображення рукописних арабських цифр



Зображення одягу та взуття



Зображення рукописних китайських ієрогліфів цифр



Розділ 5. Рекурентні нейронні мережі: робота з текстом

Тема 16. Рекурентні нейронні мережі в задачах оброблення текстів

Рекурентні нейронні мережі (RNN) – це клас нейронних мереж, які обробляють послідовності даних, зберігаючи інформацію про попередні стани. Їх широко використовують для роботи з послідовними даними, такими як тексти, оброблення природної мови (Natural Language Processing), часові ряди та інші.

Основна ідея RNN полягає в тому, що вони мають внутрішню пам'ять, яка дозволяє їм запам'ятовувати інформацію про попередні входи та використовувати її для оброблення подальших входів. Цього досягають методом додавання зворотних зв'язків у мережу, що дозволяють інформації «протікати» через час.

У RNN кожен нейрон має два набори вагових коефіцієнтів: один для поточного входу та один для попереднього стану. Це дозволяє мережі враховувати контекст та залежності між послідовними елементами даних.

Однією з ключових властивостей RNN є здатність моделювати довгострокові залежності в послідовних даних. Це означає, що RNN можуть уловлювати та використовувати інформацію, репрезентовану в минулому, щоб приймати рішення в сьогоденні.

Однак RNN має свої обмеження. Однією з проблем є зникнення та «вибух» градієнта, які можуть виникнути під час навчання мережі на довгих послідовностях. Це може призвести до того, що мережа не зможе ефективно використовувати інформацію про далекі входи. Для вирішення цієї проблеми було розроблено різні модифікації RNN, такі як LSTM (Long Short-Term Memory) та GRU (Gated Recurrent Unit), які мають механізми для збереження та використання інформації на довгі періоди часу.

16.1. Архітектура RNN та особливості їх використання

Архітектура RNN може бути різною. Наведемо деякі з можливих варіантів архітектури рекурентних нейронних мереж. Типові структури подано на рисунку 16.1.

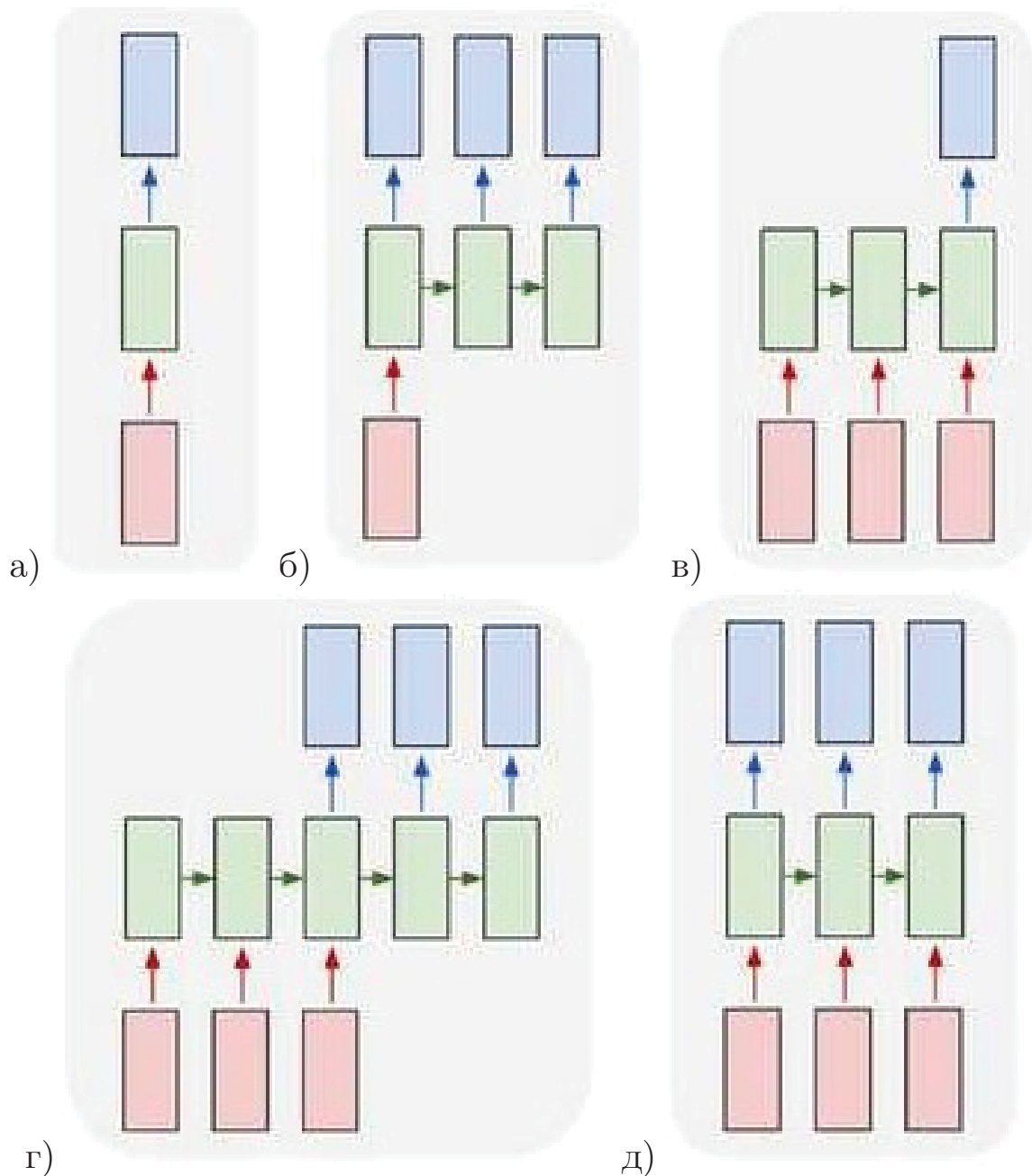


Рисунок 16.1 – Типові структури рекурентних нейронних мереж різної архітектури

Один-до-одного: це стандартна загальна нейронна мережа, для цього нам не потрібна RNN. Цю нейронну мережу використовують для вхідних даних фіксованого розміру та для вихідних даних фіксованого розміру, наприклад, для класифікації зображень (рисунок 16.1а).

Один-до-багатьох: підписи до зображень. Для підписів до зображень вхід – це зображення, а вихід – підпис до зображення. Цю нейронну мережу використовують, наприклад, для генерації музики. Для створення музики одна вхідна нота подається в мережу й вона передбачає наступну ноту в послідовності (рисунок 16.1б).

Багато-до-одного: вхідні дані – це рецензія на фільм (кілька слів у вхідних даних), а вихідні дані – це настрої, пов’язані з рецензією (рисунок 16.1в). **Багато-до-багатьох:** машинний переклад речення однією мовою на речення іншою мовою; розпізнавання мови (рисунок 16.1г).

Багато-до-багатьох: розпізнавання іменних сутностей, де вхідна довжина та вихідна довжина ідентичні (рисунок 16.1д).

Основні властивості рекурентних нейронних мереж.

- **Рекурентність:** RNN мають зворотні зв’язки, що дозволяють їм передавати інформацію з попередніх кроків у часі (епох навчання) до наступних. Це дозволяє моделювати залежності в послідовностях даних.

- **Внутрішня пам’ять:** RNN мають внутрішню пам’ять, що дозволяє їм запам’ятовувати інформацію про попередні кроки в часі та використовувати її для прийняття рішень на поточному кроці.

- **Параметри:** RNN мають параметри, які можуть бути навчені з використанням алгоритмів градієнтного спуску. Це дозволяє моделі адаптуватися до конкретних даних та задач.

Приклади задач, які можуть бути розв’язані з використанням рекурентних нейронних мереж.

- **Мовне моделювання:** RNN можуть бути використані для передбачення наступного слова в тексті або генерації нового тексту.

- **Машинний переклад (приклад Google Translate):** RNN можуть бути використані для перекладу тексту з однієї мови на іншу за принципом «багато-до-багатьох». Оригінальну послідовність

тексту подають у рекурентну нейронну мережу, яка потім створює перекладений текст як результат виведення.

- Аналіз настроїв часто виконується за допомогою рекурентних нейронних мереж із принципом «багато-до-одного» для встановлення позитивності чи негативності відгуку, чи визначення емоційного забарвлення тексту. Аналізований текст подається в нейронну мережу, яка потім створює єдину класифікацію виведення. Наприклад, цей відгук позитивний.

- Розпізнавання мови: RNN можуть бути використані для розпізнавання мови та її транскрибування.

- Аналіз часових рядів: RNN можуть бути використані для прогнозування майбутніх значень часових рядів, наприклад, ціни акцій або погоди.

16.2. Побудова рекурентної нейронної мережі

Припустимо, що в нас є нейронна мережа, яка працює за принципом «багато-до-багатьох». Вхідні дані: $x_0, 1, \dots, x_n$, а результати виведення (виходи): y_0, y_1, \dots, y_n . Дані x_i та y_i є векторами й можуть бути довільних розмірів.

Рекурентні нейронні мережі RNN працюють методом ітерованого оновлення прихованого стану h , який є вектором, що може мати довільний розмір. Приклад рекурентної мережі з архітектурою «багато-до-багатьох» наведено на рисунку 16.2.

Потрібно враховувати, що на будь-якому заданому етапі (часовій ітерації) t :

- значення прихованого стану h_t в наступний момент часу t підраховують за допомогою значення прихованого стану в попередній момент часу h_{t-1} та значення входу в наступний момент часу x_t ;

- вихід у наступний момент часу y_t підраховується за допомогою значення прихованого стану h_t у наступний момент часу.

На кожному кроці рекурентна нейронна мережа використовує ті самі вагові коефіцієнти. Іншими словами, типова класична

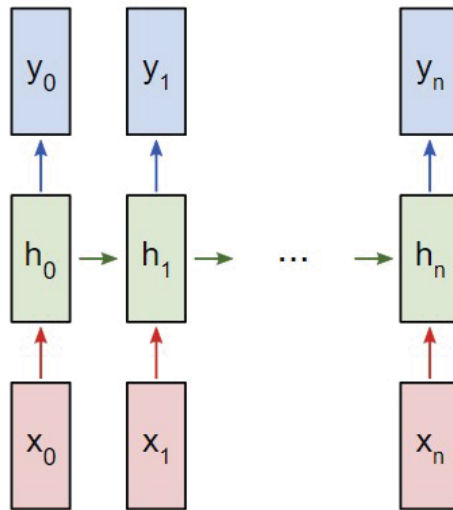


Рисунок 16.2 – Рекурентна нейронна мережа RNN з архітектурою «багато-до-багатьох»

рекурентна нейронна мережа використовує лише три набори параметрів вагових коефіцієнтів для виконання необхідних підрахунків:

- W_{xh} використовують для всіх зв'язків $x_t \rightarrow h_t$;
- W_{hh} використовують для всіх зв'язків $h_{t-1} \rightarrow h_t$;
- W_{hy} використовують для всіх зв'язків $h_t \rightarrow y_t$.

Для цієї рекурентної нейронної мережі також необхідно використовувати два параметри зміщення:

- b_h додають під час підрахунку h_t ;
- b_y додають під час підрахунку y_t .

У цьому разі вагові коефіцієнти W будуть подаватися як матриці, а параметри зміщення b як вектори. В цьому разі випадку рекурентна нейронна мережа складається з трьох матриць вагових коефіцієнтів та двох векторів зміщень.

16.3. Застосування рекурентних нейронних мереж для генерації тексту

Генерація тексту – це створення нового тексту, що має сенс і структуру з використанням комп'ютерних алгоритмів. Це завдання є однією з ключових областей штучного інтелекту та оброблення природної мови.

Генерація тексту може бути корисною в багатьох областях, таких як автоматичне створення контенту для сайтів, генерація діалогових систем, створення літературних творів та багато іншого. Вона дозволяє комп'ютерам створювати тексти, які можуть бути схожими на тексти, створені людьми.

Для генерації тексту використовують різні методи та алгоритми, зокрема статистичні моделі, марківські ланцюги, глибоке навчання та рекурентні нейронні мережі.

Одним з основних підходів до генерації тексту є використання рекурентних нейронних мереж (RNN) з урахуванням контексту та залежності в тексті під час генерації нового тексту.

Генерація тексту з використанням RNN ґрунтується на навчанні моделі на великому наборі текстових даних. Модель обробляє послідовність вхідних символів і передбачає наступний символ у послідовності. Цей процес повторюється для створення цілого тексту.

Рекурентні нейронні мережі (RNN) широко застосовують у завданнях генерації тексту. Вони дозволяють моделювати ймовірнісне розподілення послідовності символів або слів і генерувати нові тексти, які можуть бути схожими на навчальний набір даних.

Одним з основних застосувань генерації тексту з використанням RNN є автоматичне створення текстових описів зображень. Наприклад, модель може бути навчена на наборі даних, що містить зображення та відповідні їм описи. Потім, використовуючи навчену модель, можна згенерувати опис нового зображення.

Іншим застосуванням є створення тексту на основі заданого початкового фрагмента. Наприклад, модель може бути навчена на наборі даних, що містить вірші або прозу, а потім використовуватися для генерації нових текстових фрагментів, починаючи із заданого початкового слова або речення.

Генерація тексту з використанням RNN також може бути корисною в завданнях автокомпліту або передбачення наступного символу або слова в тексті. Наприклад, під час введення тексту в пошуковий рядок або під час написання повідомлення модель може пропонувати варіанти продовження тексту, ґрунтуючись на попередньому контексті.

Однак, під час генерації тексту за допомогою RNN можуть виникати деякі проблеми, такі як повторення фраз, нелогічність або неправильна граматика. Для покращення якості генерації тексту можуть застосовувати різні техніки, такі як використання більш складних архітектур RNN, додавання додаткових контекстних ознак або використання алгоритмів навчання з підкріпленням.

16.4. Архітектури рекурентних нейронних мереж для генерації тексту

Для генерації тексту з використанням рекурентних нейронних мереж (RNN) існує кілька різних архітектур, які можуть бути застосовані залежно від конкретного завдання та вимог.

Проста рекурентна нейронна мережа (Simple RNN)

Проста рекурентна нейронна мережа є базовою архітектурою для генерації тексту. Вона складається з одного шару рекурентних нейронів, які передають інформацію від попереднього кроку до наступного. Ця архітектура може бути використана для створення тексту посимвольно або за словами.

Довга короткострокова пам'ять (Long Short-Term Memory, LSTM)

Архітектура LSTM є розширенням простої RNN і дозволяє моделювати довгострокові залежності в тексті. Вона складається з трьох основних компонентів: вхідного гейту, гейта що забуває та вихідного гейту. Вхідний гейт вирішує, яка інформація буде збережена в пам'яті, гейт, що забуває, визначає, яка інформація буде забута, а вихідний гейт визначає, яка інформація буде використана для генерації наступного символу.

Глибока рекурентна нейронна мережа (Deep RNN)

Глибока рекурентна нейронна мережа складається з кількох шарів рекурентних нейронів, які передають інформацію від одного шару до іншого. Ця архітектура дозволяє моделювати складніші залежності в тексті та може покращити якість генерованого тексту.

Мережа з довго- та короткостроковою пам'яттю (Gated Recurrent Unit, GRU)

Архітектура GRU є альтернативою LSTM і має простішу структуру. Вона складається з двох гейтів: оновлення та скидання. Гейт оновлення вирішує, яку інформацію буде збережено в пам'яті, а гейт скидання визначає, яку інформацію буде забуто.

Вибір конкретної архітектури залежить від завдання та вимог. Деякі архітектури можуть бути більш застосовними для створення тексту посимвольно, у той час як інші можуть краще підходити для створення тексту за словами. Також можна експериментувати з різними комбінаціями архітектур та налаштуваннями гіперпараметрів для досягнення найкращих результатів.

16.5. Проблеми та виклики у генерації тексту з використанням RNN

Генерація тексту з використанням рекурентних нейронних мереж може зіткнутися з кількома проблемами та викликами, які можуть ускладнити процес та впливати на якість результатів. Розглянемо деякі з них.

Обмежена пам'ять

Рекурентні нейронні мережі мають обмежену пам'ять, і це означає, що вони можуть мати труднощі запам'ятовування довгих залежностей у тексті. Це може призвести до проблеми «загасаючого градієнта», коли градієнти, необхідні для оновлення вагових коефіцієнтів, стають занадто малими й мережа втрачає здатність ураховувати далекі залежності в тексті.

Проблема вибору

Генерація тексту вимагає прийняття рішень щодо вибору наступного символу або слова на основі попереднього контексту. Це може бути складним завданням, особливо, коли є кілька можливих варіантів. Рекурентні нейронні мережі можуть зіткнутися з проблемою «експоненційного зростання можливостей», коли кількість можливих варіантів зростає експоненційно зі збільшенням довжини генерованого тексту.

Повторення та недомовленість

Рекурентні нейронні мережі можуть мати тенденцію повторюватися або генерувати недомовлений текст. Це може бути викликано недостатнім розмаїттям у навчальних даних або недостатньою складністю моделі. Для вирішення цієї проблеми можна використовувати різні методи, такі як додавання випадковості до процесу генерації або використання більш складних архітектур мереж.

Збереження сенсу та граматичної коректності

Генерація тексту вимагає збереження сенсу та граматичної коректності. Рекурентні нейронні мережі можуть мати труднощі в генерації тексту, який є логічним та граматично правильним. Для вирішення цієї проблеми можна використовувати додаткові методи, такі як використання мовних моделей або включення правил граматики в процес генерації.

У цілому, генерація тексту з використанням рекурентних нейронних мереж є складним завданням, яке вимагає врахування безлічі факторів та вирішення різних проблем. Однак, із правильним підходом та налаштуваннями, рекурентні нейронні мережі можуть бути потужним інструментом для створення якісного тексту.

16.6. Приклади застосування генерації тексту з використанням рекурентних нейронних мереж

Генерація тексту з використанням рекурентних нейронних мереж має широкий спектр застосувань у різних галузях. Нижче наведено деякі приклади використання цієї технології.

Генерація тексту для машинного перекладу

Рекурентні нейронні мережі можуть бути використані для генерації тексту завдання машинного перекладу. Нейронна мережа може бути навчена на парах речень різними мовами, щоб навчитися перекладати одне речення іншою мовою. Після навчання мережа може бути використана для створення перекладу для нових речень.

Генерація тексту для автоматичного створення контенту

Рекурентні нейронні мережі можуть використовуватися для автоматичного створення контенту, такого як статті, новини або рекламні тексти. Мережа може бути навчена на великому обсязі текстових даних, а потім використовуватися для генерації нових текстів на основі цього навчання.

Генерація тексту для чат-ботів

Рекурентні нейронні мережі можуть бути використані для створення тексту в чат-ботах. Мережа може бути навчена на великому обсязі діалогів і використовуватися для створення відповідей на запитання або коментарі користувачів.

Генерація тексту для музики

Рекурентні нейронні мережі можуть бути використані для створення тексту для музики. Мережа може бути навчена на музичних композиціях і використовуватися для створення нових музичних фрагментів або текстів пісень.

Генерація тексту для автоматичної генерації коду

Рекурентні нейронні мережі можуть бути використані для створення тексту для автоматичної генерації коду. Мережа може бути навчена на великому обсязі програмного коду та використовуватися для генерації нових кодових фрагментів на основі цього навчання.

Це лише деякі приклади застосування генерації тексту з використанням нейронних рекурентних мереж. Можливості цієї технології є величезними й вона може бути застосована в багатьох інших областях, де потрібна генерація тексту.

Тема 17. Аналіз емоційного забарвлення тексту за допомогою рекурентної нейронної мережі

Завдання нейронної мережі – визначення настрою (емоційного забарвлення) фраз: позитивний чи негативний характер вона має. Для цього спершу нам треба згенерувати набір навчальних даних, який може мати вигляд, як наведено в таблиці 17.1.

Фраза	Позитивна ?
Я гарний	Так
Я поганий	Ні
Це дуже добре	Так
Це непогано	Так
Я поганий, а не добрий	Ні
Я нещасний	Ні
Це було добре	Так
Я почуваюся непогано, мені не сумно	Так
...	...

Таблиця 17.1 – Приклад набору навчальних даних для визначення емоційного забарвлення фрази

17.1. Побудова нейронної мережі

Для задачі визначення емоційного забарвлення тексту (фрази) доцільно використати класифікацію рекурентної мережі «багато-до-одного». Принцип її використання нагадує роботу схеми «багато-до-багатьох», що була описана раніше. Однак цього разу буде задіяно лише прихований стан для одного виходу y . Схему мережі подано на рисунку 17.1.

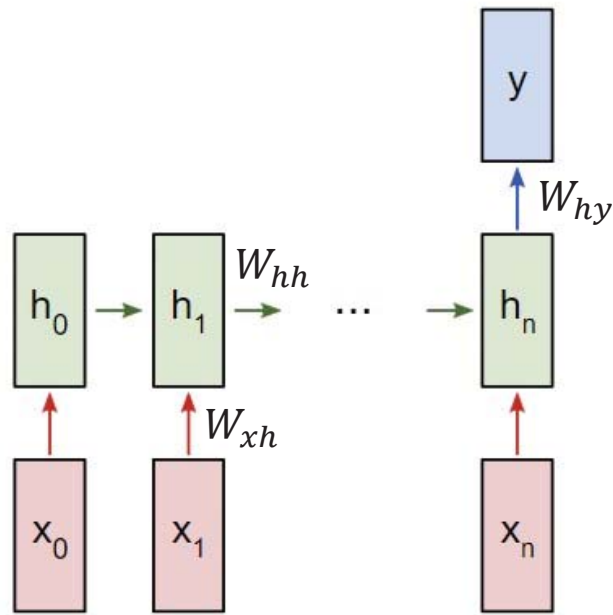


Рисунок 17.1 – Рекурентна нейронна мережа з архітектурою «багато-до-одного» для задачі визначення емоційного забарвлення тексту

Тут кожен вхід мережі x_i буде вектором, репрезентує певне слово з тексту. Результат y буде вектором, що містить два числа. Одне репрезентує позитивний настрій, а друге – негативний. Для того, щоб перетворити ці значення на ймовірність доцільно використати функцію **SoftMax**, або ж нормовану експоненційну функцію, що є узагальненням логістичної функції, що «стискає» K -вимірний вектор z із довільним значеннями компонент до K -вимірного вектора $\sigma(z)$ з дійсними значеннями компонент в області $[0, 1]$, що в сумі дають одиницю

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{k=1}^K e^{z_k}}. \quad (17.1)$$

У результаті оберуть найімовірніший результат між позитивним і негативним.

17.2. Попереднє оброблення даних

Рекурентна нейронна мережа не розрізняє слів – лише числа. Тому перед тим як будувати нейронну мережу треба провести попереднє оброблення даних. Наведений у таблиці 17.1 набір даних (подібний набір даних англійською мовою наведено наприкінці цієї теми) доцільно поділити на навчальну та тестову вибірки, які зручно подати у вигляді двох словників Python:

```
train_data = {
    'good': True,
    'bad': False,
    # ... більше даних
}
test_data = {
    'this is happy': True,
    'i am good': True,
    # ... більше даних
}
```

У цьому наборі значення цільового вектора: **True** – позитивне забарвлення; **False** – негативне забарвлення.

Для одержання даних у зручному форматі потрібно зробити певне попереднє оброблення. Для початку необхідно створити словник Python з усіх **унікальних** слів, які вживаються в наборі даних. Для цього потрібно розділити слова за пробілом та зручно використовувати функцію `set()` яка додає до словника лише ті слова, яких там ще було.

```
from data import train_data, test_data

# Створення словника
vocab = list(set([w for text in train_data.keys() for w in
                  text.split(' ')]))
vocab_size = len(vocab)

print('%d unique words found' % vocab_size) # знайдено 18
                                             унікальних слів
```

Словник `vocab` тепер містить перелік всіх слів, які вживають щонайменше в одному навчальному тексті. Для репрезентування слів у вигляді чисел надамо кожному слову з `vocab` індекс типу `integer`

(ціле число).

```
# Призначаємо індекс кожному слову
word_to_idx = {w: i for i, w in enumerate(vocab)}
idx_to_word = { i: w for i, w in enumerate (vocab) }

print(word_to_idx['good']) # 16
print(idx_to_word[0]) # сумно
# Індекс та слово можуть відрізнятися у конкретній
# реалізації
```

Тепер можна відобразити будь-яке слово за допомогою індексу цілого числа. Кожна фраза з набору, наприклад наведеного в таблиці 17.1 містить певну кількість слів і є входом нейронної мережі, який репрезентовано набором слів (вектором) x_i . Для переформатування входів x_i нейронної мережі зручно використати підхід *one-hot encoding* – подання вхідного вектора у вигляді унітарного коду¹. У цьому разі кожний вхід x_i рекурентної нейронної мережі є вектором. Оскільки в отриманому словнику `vocab` міститься 18 унікальних слів, то кожен вхід мережі x_i буде 18-ти мірним унітарним вектором.

```
import numpy as np

def createInputs(text):

    inputs = []
    for w in text.split(' '):
        v = np.zeros((vocab_size, 1))
        v[word_to_idx[w]] = 1
        inputs.append(v)

    return inputs
```

Функція `createInputs()` буде використана для створення вхідних даних у вигляді векторів та подальшого їх передавання в рекурентну нейронну мережу RNN.

¹У машинному навчанні унітарний код, або прямиий унітарний код (англ. *one-hot*) – це група бітів, серед яких дозволеними комбінаціями значень є лише ті, у яких установлено «1» лише один біт, а всі інші вимкнено «0». Подібне втілення, в якому всі біти є «1», крім одного «0», іноді називають зворотним (інверсним) унітарним кодом (англ. *one-cold*).

17.3. Фаза прямого поширення

Відповідно до структури рекурентної нейронної мережі поданої на рисунку 17.1 для поставленої задачі визначення емоційного забарвлення тексту (фрази) вона характеризується трьома матрицями вагових коефіцієнтів та двома векторами параметрів зміщення. Спершу проведемо ініціалізацію цих параметрів випадковими числами.

```
import numpy as np
from numpy.random import randn

class RNN:
    # Класична рекурентна нейронна мережа

    def __init__(self, input_size, output_size,
                 hidden_size=64):
        # Матриці вагових коефіцієнтів
        self.Whh = randn(hidden_size, hidden_size) / 1000
        self.Wxh = randn(hidden_size, input_size) / 1000
        self.Why = randn(output_size, hidden_size) / 1000

        # Параметри зміщення
        self.bh = np.zeros((hidden_size, 1))
        self.by = np.zeros((output_size, 1))
```

Тут для того, щоб прибрати внутрішню варіативність вагових коефіцієнтів, використовують ділення на 1 000. Це не найкращий спосіб ініціалізації вагових коефіцієнтів, проте він досить простий і непогано спрацює для цього прикладу. Для ініціалізації вагових коефіцієнтів випадковими числами зі стандартного нормального розподілу тут використано `np.random.randn()`.

Прямий хід по нейронній мережі відбувається методом розрахунку значень прихованого стану та виходу з використанням стандартних формул

$$\begin{aligned}h_t &= \tanh(W_{xh}x_t + W_{hh}h_{t-1} + b_h), \\y_t &= W_{hy}h_t + b_y.\end{aligned}\tag{17.2}$$

Тут для розрахунку значення прихованого стану використано активаційну функцію у вигляді гіперболічного тангенсу

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}},\tag{17.3}$$

хоча й інші типи активаційних функцій (наприклад, сигмоїдна функція активації) також можуть бути використані.

```
class RNN:
    # ...

    def forward(self, inputs):

        h = np.zeros((self.Whh.shape[0], 1))

        # Виконання кожного кроку в нейронній мережі RNN
        for i, x in enumerate(inputs):
            h = np.tanh(self.Wxh @ x + self.Whh @ h + self
                        .bh)

        # Розрахунок виходу
        y = self.Why @ h + self.by

        return y, h
```

Потрібно звернути увагу на те, що значення прихованого шару h для нульового вектора h_0 в першому кроці ініціалізовано нулями, оскільки на першій ітерації відсутнє попереднє значення h .

Отже, тепер необхідно визначити функцію SoftMax відповідно до математичного виразу (17.1) для визначення ймовірності позитивного чи негативного забарвлення тексту (фрази), і пропустити вхідний сигнал через побудовану нейронну мережу.

```
# ...

def softmax(xs):
    # Застосування функції Softmax для вхідного масиву
    return np.exp(xs) / sum(np.exp(xs))

# Ініціалізація нашої рекурентної нейронної мережі RNN
rnn = RNN (vocab_size, 2)

inputs = createInputs('i am very good')
out, h = rnn.forward(inputs)
probs = softmax(out)
print(probs) # [[0.50000095], [0.49999905]]
```

Отже, рекурентна нейронна мережа побудована та працює, проте її важко назвати корисною, оскільки одержані ймовірності щодо позитивності чи негативності поданої на вхід мережі фрази є майже однаковими.

17.4. Фаза зворотного поширення

Існує парадигма навчання, яку називають оцінкою за методом найбільшої ймовірності, яка навчає модель оцінювати її параметри, щоб вивчити базовий розподіл даних. Отже, ми використовуємо функцію втрат, щоб оцінити, наскільки добре модель відповідає розподілу даних.

Для навчання (тренування) рекурентної нейронної мережі у якості функції втрат буде використано **перехресну ентропію** (cross-entropy loss), яка здебільшого сумісна з функцією SoftMax.

17.4.1. Крос-ентропія

Крос-ентропія (перехресна ентропія) – це Функція втрат (Loss Function), яку можна використовувати для кількісної оцінки різниці між двома розподілами ймовірностей (Probability Distribution). Крос-ентропія говорить про те, що якщо ми маємо події та ймовірності, то наскільки ймовірно, що події відбудуться на основі ймовірностей. Якщо це дуже можливо, то крос-ентропія є малою та навпаки, якщо якщо це малоймовірно, то крос-ентропія буде високою.

Використовуючи перехресну ентропію, можна оцінити помилку (чи різницю) між двома ймовірнісними розподілами: заданим (визначеним за допомогою цільового вектора – відомих відповідей або міток класів) та одержаного за допомогою нейронної мережі. У разі бінарної класифікації для задачі встановлення емоційного забарвлення тексту (фрази) перехресну ентропію визначають як

$$L = -(y \log(p) + (1 - y) \log(1 - p)), \quad (17.4)$$

де

- p – імовірність що передбачається нейронною мережею;
- y – індикатор («так» або «ні») або (0 або 1) у прикладі бінарної класифікації.

Отже, якщо p_c є передбачуваною ймовірністю рекурентної нейронної мережі для правильно класифікованого речення (класу `correct`: позитивний або негативний) то правильний індикатор набуває значення $y = 1$. У цьому разі рівняння (17.4) спрощується та набуває вигляду

$$L = -\log(p_c). \quad (17.5)$$

Метою навчання нейронної мережі, яке відбувається за методом градієнтного спуску, є зменшення втрат перехресної ентропії (17.5).

17.4.2. Параметри нейронної мережі

Побудована нейронна мережа характеризується таким набором параметрів:

- y – необроблені вихідні дані нейронної мережі;
- p – імовірність, розрахована за формулою (17.1): $p = \sigma(y)$;
- c – відома мітка певного зразка тексту, так званий «правильний» клас;
- L – втрата перехресної ентропії: $L = -\log(p_c)$;
- W_{xh} , W_{hh} і W_{hy} – три матриці вагових коефіцієнтів нейронної мережі, що розглядається;
- b_h і b_y – два вектори параметрів зміщення нейронної мережі.

Спочатку треба налаштувати фазу прямого поширення для кешування окремих даних, оскільки будуть використовувати у фазі зворотного поширення нейронної мережі.

```
class RNN:
    # ...

    def forward(self, inputs):

        h = np.zeros((self.Whh.shape[0], 1))
        self.last_inputs = inputs
        self.last_hs = { 0: h }

        # Виконання кожної ітерації нейронної мережі RNN
        for i, x in enumerate(inputs):
            h = np.tanh(self.Wxh @ x + self.Whh @ h + self.
                        bh)

            self.last_hs[i + 1] = h

        # Розрахунок виходу
        y = self.Why @ h + self.by
        return y, h

    def backprop(self, d_y, learn_rate=2e-2):
        '''
        Фаза зворотного поширення
        '''
        pass
```

Для реалізації методу градієнтного спуску необхідно розрахувати градієнти

$$\frac{\partial L}{\partial W_{xh}}, \quad \frac{\partial L}{\partial W_{hh}}, \quad \frac{\partial L}{\partial W_{hy}}, \quad \frac{\partial L}{\partial b_y}, \quad \frac{\partial L}{\partial b_h}. \quad (17.6)$$

Оскільки функція втрат L явно залежить від імовірності p_c за формулою (17.5), яка зі свого боку явно залежить від виходу y за формулою (17.1). Вихід y явно залежить від матриці вагових коефіцієнтів W_{hy} , вектора зміщень b_y та прихованого стану h_t , який у свою чергу явно залежить від матриць вагових коефіцієнтів W_{xh} і W_{hh} та вектору параметрів зміщень b_h , відповідно до формули (17.2). Отже, під час розрахунків похідних (17.6) маємо справу з похідними від складних функцій.

Спочатку визначимо градієнти, що використовують лише для переходу від кінцевого прихованого стану h_n (n – кількість ітерацій) до виходу нейронної мережі RNN, а саме $\frac{\partial L}{\partial W_{hy}}$ та $\frac{\partial L}{\partial b_y}$. З урахуванням виразу для визначення значення виходу (друге рівняння з формули (17.2))

$$y = W_{hy}h_n + b_y \quad (17.7)$$

для похідної $\frac{\partial L}{\partial W_{hy}}$ маємо такий вираз:

$$\frac{\partial L}{\partial W_{hy}} = \frac{\partial L}{\partial y} \times \frac{\partial y}{\partial W_{hy}}. \quad (17.8)$$

Спершу обчислимо похідну $\frac{\partial L}{\partial y}$

$$L'_y \equiv \frac{\partial L}{\partial y_i} = \begin{cases} p_i, & \text{якщо } i \neq c, \\ p_i - 1, & \text{якщо } i = c. \end{cases} \quad (17.9)$$

Для похідної $\frac{\partial y}{\partial W_{hy}}$ з використанням зв'язку (17.7) маємо

$$\frac{\partial y}{\partial W_{hy}} = h_n. \quad (17.10)$$

Отже, для похідної $\frac{\partial L}{\partial W_{hy}}$ одержуємо вираз

$$\frac{\partial L}{\partial W_{hy}} = L'_y h_n. \quad (17.11)$$

Аналогічно знаходимо похідну $\frac{\partial L}{\partial b_y}$

$$\frac{\partial L}{\partial b_y} = \frac{\partial L}{\partial y} \times \frac{\partial y}{\partial b_y}. \quad (17.12)$$

Ураховуючи, (17.7) отримуємо $\frac{\partial y}{\partial b_y} = 1$, що в результаті дає

$$\frac{\partial L}{\partial b_y} = L'_y. \quad (17.13)$$

Далі перейдемо до розрахунку похідних $\frac{\partial L}{\partial W_{xh}}$, $\frac{\partial L}{\partial W_{hh}}$ і $\frac{\partial L}{\partial b_h}$, які використовують на кожній ітерації роботи нейронної мережі. Із використанням визначень (17.2) для похідної $\frac{\partial L}{\partial W_{xh}}$ маємо

$$\frac{\partial L}{\partial W_{xh}} = \frac{\partial L}{\partial y} \times \sum_{t=0}^n \frac{\partial y}{\partial h_t} \times \frac{\partial h_t}{\partial W_{xh}}. \quad (17.14)$$

Тут зміна W_{xh} впливає не лише на кожен h_t , але й на всі y , що зі свого боку приводить до зміни в L . Для того, щоб повністю підрахувати цей градієнт необхідно провести зворотне поширення через усі часові кроки (ітерації). Цей метод також має назву **зворотного поширення в часі**, або *Backpropagation Through Time* (BPTT). Схематично процес зворотного поширення показано на рисунку 17.2. Матрицю вагових коефіцієнтів W_{xh} використовують для всіх прямих посилянь $x_t \rightarrow h_t$, тому нам потрібно провести зворотне поширення назад до кожного із цих посилянь. Наблизившись до заданого кроку t , потрібно підрахувати градієнт $\frac{\partial h_t}{\partial W_{xh}}$ з урахуванням першого рівняння з формули (17.2)

$$h_t = \tanh(W_{xh}x_t + W_{hh}h_{t-1} + b_h). \quad (17.15)$$

Похідна гіперболічного тангенса має такий вигляд:

$$\frac{d \tanh(x)}{dx} = 1 - \tanh^2(x). \quad (17.16)$$

Використовуючи правила диференціювання складної функції та зв'язок (17.15), одержуємо

$$\frac{\partial h_t}{\partial W_{xh}} = (1 - h_t^2) x_t. \quad (17.17)$$

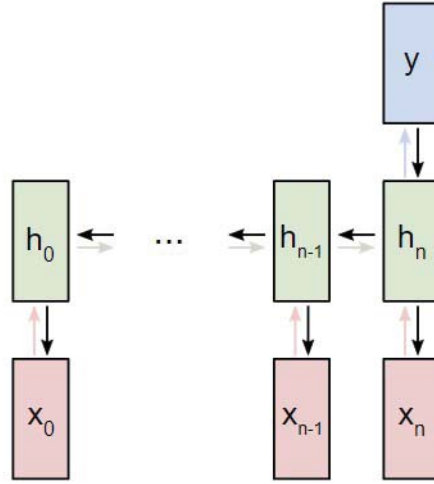


Рисунок 17.2 – Схематичне подання процесу зворотного поширення у часі для рекурентної нейронної мережі з рис. 17.1

Аналогічним способом обчислюємо

$$\begin{aligned} \frac{\partial h_t}{\partial W_{hh}} &= (1 - h_t^2) h_{t-1}, \\ \frac{\partial h_t}{\partial b_h} &= (1 - h_t^2). \end{aligned} \quad (17.18)$$

Останній потрібний градієнт $\frac{\partial y}{\partial h_t}$ можна підрахувати рекурсивно

$$\frac{\partial y}{\partial h_t} = \frac{\partial y}{\partial h_{t+1}} \times \frac{\partial h_{t+1}}{\partial h_t} = \frac{\partial y}{\partial h_{t+1}} (1 - h_t^2) W_{hh}. \quad (17.19)$$

Реалізуємо зворотне поширення в часі, або ВРТТ, відштовхуючись від прихованого стану як початкової точки. Далі працюватимемо у зворотному порядку. Тому на момент підрахунку $\frac{\partial y}{\partial h_t}$ значення $\frac{\partial y}{\partial h_{t+1}}$ буде відоме. Винятком стане лише останній прихований стан h_n , для якого

$$\frac{\partial y}{\partial h_n} = W_{hy}. \quad (17.20)$$

Тепер у нас є все необхідне для того щоб реалізувати зворотне поширення в часі (ВРТТ). Водночас у разі розрахунку всіх градієнтів (17.6) використовують похідну (17.9). Тому для її розрахунку варто зробити окремий програмний блок.

```

# Розрахунок похідної dL/dy

# Цикл для кожного прикладу з навчальної вибірки
for x, y in train_data.items():
    inputs = createInputs(x)
    target = int(y)

    # Пряме поширення
    out, _ = rnn.forward(inputs)
    probs = softmax(out)

    # Розрахунок dL/dy
    d_L_d_y = probs
    d_L_d_y[target] -= 1

    # Зворотнє поширення
    rnn.backprop(d_L_d_y)

```

Розрахунок всіх похідних (17.6) та оновлення значень матриць вагових коефіцієнтів W_{xh} , W_{hh} , W_{hy} та векторів параметрів зміщення b_h , b_y проводять в одній функції `backprop`.

```

class RNN:
    # ...

    def backprop(self, d_y, learn_rate=2e-2):

        n = len(self.last_inputs)

        # Розрахунок dL/dWhy та dL/dby.
        d_Why = d_y @ self.last_hs[n].T
        d_by = d_y

        # Ініціалізація dL/dWhh, dL/dWxh, та dL/dbh.
        d_Whh = np.zeros(self.Whh.shape)
        d_Wxh = np.zeros(self.Wxh.shape)
        d_bh = np.zeros(self.bh.shape)

        # Розрахунок dL/dh для останнього h.
        d_h = self.Why.T @ d_y

```

```

# Зворотнє поширення у часі.
for t in reversed(range(n)):
    # Середнє значення: dL/dh * (1 - h^2)
    temp = ((1 - self.last_hs[t + 1] ** 2) * d_h)

    # dL/db = dL/dh * (1 - h^2)
    d_bh += temp

    # dL/dWhh = dL/dh * (1 - h^2) * h_{t-1}
    d_Whh += temp @ self.last_hs[t].T

    # dL/dWxh = dL/dh * (1 - h^2) * x
    d_Wxh += temp @ self.last_inputs[t].T

    # Далі dL/dh = dL/dh * (1 - h^2) * Whh
    d_h = self.Whh @ temp

# Відсікаємо, щоб унеможливити розрив градієнтів.
for d in [d_Wxh, d_Whh, d_Why, d_bh, d_by]:
    np.clip(d, -1, 1, out=d)

# Оновлюємо вагові коефіцієнти та зміщення за
# методом градієнтного спуску.

self.Whh -= learn_rate * d_Whh
self.Wxh -= learn_rate * d_Wxh
self.why -= learn_rate * d_Why
self.bh -= learn_rate * d_bh
self.by -= learn_rate * d_by

```

Моменти, на які необхідно звернути увагу.

- Для зручності градієнти $\frac{\partial L}{\partial y} \times \frac{\partial y}{\partial h}$ об'єднані у $\frac{\partial L}{\partial h}$.
- Змінна d_h , яка тримає останню версію градієнта $\frac{\partial y}{\partial h_{t+1}}$, який необхідний для підрахунку градієнта $\frac{\partial L}{\partial h_t}$ постійно оновлюється.
- Закінчивши зі зворотним поширенням у часі (ВРТТ), використовується `np.clip()` на значеннях градієнта нижче -1 або вище 1 . Це допоможе позбавитися проблеми з вибуховими градієнтами. Таке трапляється, коли градієнти стають занадто великими через велику кількість помножених параметрів. Вибух, а також зникнен-

ня градієнтів не є рідкістю для класичних рекурентних нейронних мереж.

- Коли всі градієнти підраховані, вагові коефіцієнти та параметри зміщення оновлюються за допомогою градієнтного спуску.

17.5. Тестування нейронної мережі

Для початку, напишемо допоміжну функцію для оброблення даних рекурентної нейронної мережі.

```
import random

def processData(data, backprop=True):
    items = list(data.items())
    random.shuffle(items)

    loss = 0
    num_correct = 0

    for x, y in items:
        inputs = createInputs(x)
        target = int(y)

        # Пряме поширення
        out, _ = rnn.forward(inputs)
        probs = softmax(out)

        # Розрахунок втрат / точності
        loss -= np.log(probs[target])
        num_correct += int(np.argmax(probs) == target)

    if backprop:
        # Розрахунок dL/dy
        d_L_d_y = probs
        d_L_d_y[target] -= 1

        # Зворотне поширення
        rnn.backprop(d_L_d_y)

    return loss / len(data), num_correct / len(data)
```

Цикл для тренування мережі.

```
# Цикл тренування
for epoch in range(1000):
    train_loss, train_acc = processData(train_data)
    if epoch % 100 == 99:
        test_loss, test_acc = processData(test_data,
                                          backprop=False)
```

Результати навчання побудованої нейронної мережі після 1000 епох навчання наведено на рисунку 17.3.

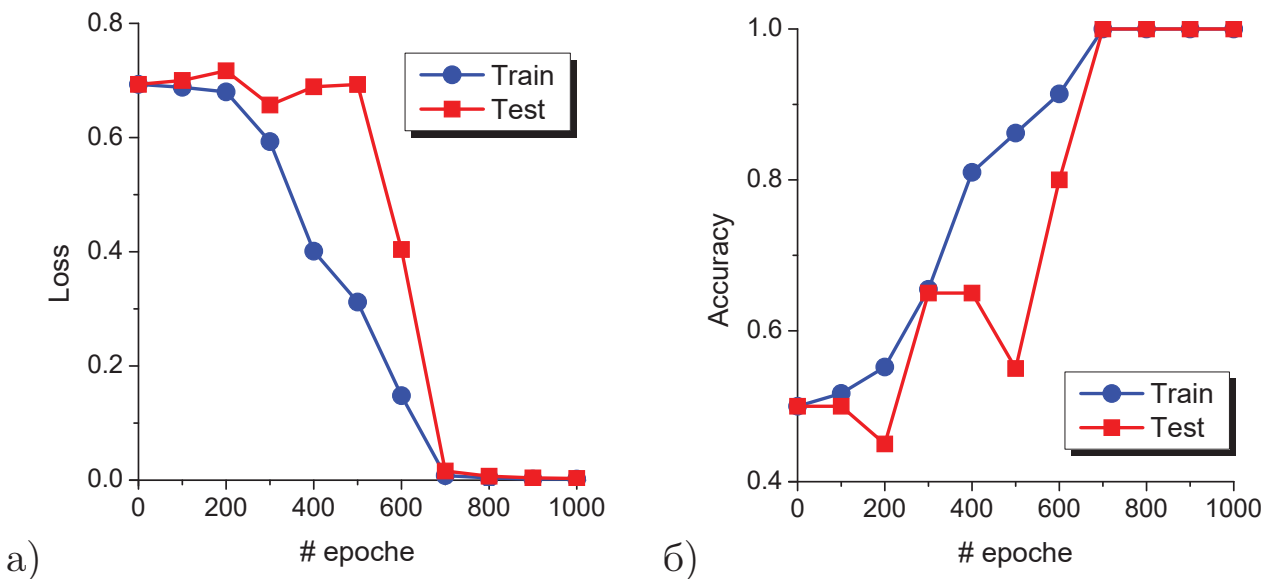


Рисунок 17.3 – Результати навчання побудованої нейронної мережі після 1000 епох навчання

З рисунка видно, що на навчальних даних (сині кружечки) мережа поступово вчиться, причому втрати монотонно зменшуються (рис. 17.3а), тоді як точність класифікації – правильного визначення емоційного забарвлення фрази – монотонно зростає (рис. 17.3б) і після проходження 700 епох мережу можна вважати повністю навченою. На тестових даних залежності як втрат так і точності класифікації ведуть себе немонотонно упродовж 600 епох навчання. Після 700 епох навчена рекурентна нейронна мережа показує 100 % результат і на тестових даних.

Приклад даних для тренування рекурентної нейронної мережі визначати емоційне забарвлення тексту (фрази).

```
train_data = {
    'good': True,
    'bad': False,
    'happy': True,
    'sad': False,
    'not good': False,
    'not bad': True,
    'not happy': False,
    'not sad': True,
    'very good': True,
    'very bad': False,
    'very happy': True,
    'very sad': False,
    'i am happy': True,
    'this is good': True,
    'i am bad': False,
    'this is bad': False,
    'i am sad': False,
    'this is sad': False,
    'i am not happy': False,
    'this is not good': False,
    'i am not bad': True,
    'this is not sad': True,
    'i am very happy': True,
    'this is very good': True,
    'i am very bad': False,
    'this is very sad': False,
    'this is very happy': True,
    'i am good not bad': True,
    'this is good not bad': True,
    'i am bad not good': False,
    'i am good and happy': True,
    'this is not good and not happy': False,
    'i am not at all good': False,
    'i am not at all bad': True,
    'i am not at all happy': False,
    'this is not at all sad': True,
    'this is not at all happy': False,
    'i am good right now': True,
    'i am bad right now': False,
    'this is bad right now': False,
```

```
'i am sad right now': False,
'i was good earlier': True,
'i was happy earlier': True,
'i was bad earlier': False,
'i was sad earlier': False,
'i am very bad right now': False,
'this is very good right now': True,
'this is very sad right now': False,
'this was bad earlier': False,
'this was very good earlier': True,
'this was very bad earlier': False,
'this was very happy earlier': True,
'this was very sad earlier': False,
'i was good and not bad earlier': True,
'i was not good and not happy earlier': False,
'i am not at all bad or sad right now': True,
'i am not at all good or happy right now': False,
'this was not happy and not good earlier': False,
}
```

```
test_data = {
    'this is happy': True,
    'i am good': True,
    'this is not happy': False,
    'i am not good': False,
    'this is not bad': True,
    'i am not sad': True,
    'i am very good': True,
    'this is very bad': False,
    'i am very sad': False,
    'this is bad not good': False,
    'this is good and happy': True,
    'i am not good and not happy': False,
    'i am not at all sad': True,
    'this is not at all good': False,
    'this is not at all bad': True,
    'this is good right now': True,
    'this is sad right now': False,
    'this is very bad right now': False,
    'this was good earlier': True,
    'i was not happy and not good earlier': False,
}
```

Список літератури

1. Hand D. Principles of Data Mining / D. Hand, H. Mannila, P. Smyth. – Cambridge : MIT Press, 2001. – 578 p.
2. Donald M. Machine learning, neural and statistical classification / M. Donald, D. J. Spiegelhalter, C. C. Taylor. – New York : Ellis Horwood, 1994. – 298 p.
3. Deisenroth M. P. Mathematics for machine learning / M. P. Deisenroth, A. A. Faisal, C. S. Ong. – New York : Cambridge University Press, 2020. – 412 p.
4. Anderson J. A. An introduction to neural networks / J. A. Anderson. – Cambridge : MIT Press, 1995 – 672 p.
5. Looney C. G. Pattern recognition using neural networks: theory and algorithms for engineers and scientists / C. G. Looney. – Oxford : Oxford University Press, 1997. – 480 p.
6. Pearlmutter B. A. An investigation of the gradient descent process in neural networks / B. A. Pearlmutter. – Pittsburgh : Carnegie Mellon University, 1996. – 140 p.
7. Chong E. K. P. Gradient Methods / E. K. P. Chong, S. H. Zak. An Introduction to Optimization. – Hoboken : Wiley, 2013. – 131-160 pp.
8. Chauvin Y. Backpropagation: theory, architectures, and applications / Y. Chauvin, D. E. Rumelhart. – Hillsdale, New Jersey : Psychology Press, 1995. – 576 p.
9. Buduma N. Fundamentals of deep learning / N. Buduma, N. Buduma, J. Papa. – Sebastopol : O'Reilly Media, Inc., 2022. – 388 p.
10. Chollet F. Deep learning with Python / F. Chollet. – New York : Manning Publications, 2021. – 504 p.
11. Python Deep Learning / I. Vasiliev, et al. – Birmingham : Packt Publishing Ltd, 2019. – 362 p.

12. Purkai N. Hands-On Neural Networks with Keras: Design and create neural networks using deep learning and artificial intelligence principles / N. Purkai. – Birmingham : Packt Publishing Ltd, 2019. – 431 p.
13. McCord-Nelson M. A practical guide to neural nets / M. McCord-Nelson, W. T. Illingworth. – Michigan : Addison-Wesley Longman Publishing Co., Inc., 1991. – 328 p.
14. Aghdam H. H. Guide to convolutional neural networks / H. H. Aghdam, J. H. Elnaz. – Cham : Springer, 2017. – 282 p.
15. Tyagi A. K. Recurrent Neural Networks: Concepts and Applications / A. K. Tyagi, A. Abraham. – Boca Raton : CRC Press, 2022. – 412 p.

Електронне навчальне видання

Харченко Василь Олегович

МОДЕЛЮВАННЯ НЕЙРОННИХ МЕРЕЖ

Навчальний посібник

Редакторка О. Ф. Дубровіна
Комп'ютерне верстання В. О. Харченка

Формат 60 × 84/16. Ум. друк. арк. 15,17. Обл.-вид. арк. 15,20.

Видавець і виготовлювач
Сумський державний університет,
вул. Римського-Корсакова, 2, м. Суми, 40007
Свідоцтво суб'єкта видавничої справи ДК № 3062 від 17.12.2007.