Міністерство освіти і науки України
Одеський національний технологічний університет
Кафедра комп'ютерної інженерії

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО КВАЛІФІКАЦІЙНОЇ РОБОТИ

на тему *Побудова та тренінг нейронної мережі*
(назва кваліфікаційної роботи згідно наказу ОНТУ)

Здобувача *Галана М. М.*
(прізвище, ініціали)

2 ск-2 курсу *КІ-543* групи

Керівники: *проф. Артеменко С. В.*
(посада, прізвище та ініціали)
асист. Орел А. С.
(посада, прізвище та ініціали)

Консультанти: *ст. викл. Богданов О. О.*
(посада, прізвище та ініціали)
доц. Нєнов О. Л.
(посада, прізвище та ініціали)

Кваліфікаційна робота допускається до захисту

Рішення кафедри від *05.06* 2024 р., протокол № *8*

Завідувач кафедри комп. інженерії _____ Сергій АРТЕМЕНКО
(назва кафедри) (підпис) (Ім'я ПРІЗВИЩЕ)

Одеса - 2024 рік

ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерної інженерії, програмування та кіберзахисту*
Кафедра *комп'ютерної інженерії*
Ступінь вищої освіти *бакалавр*
Спеціальність *123 «Комп'ютерна інженерія»*
Освітня програма *Мережеві технології та інтернет речей*

ЗАТВЕРДЖУЮ
Зав. кафедри *комп'ютерної інженерії*
Сергій АРТЕМЕНКО
« *30* » *серпня* 2023 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА

Галана Миколи Миколайовича

1. Тема роботи *Побудова та тренінг нейронної мережі*

Затверджена наказом університету від « *30* » *серпня* 2023 р., наказ № *442-03*

2 Термін здачі здобувачем закінченої роботи *28 травня 2024 р.*

3. Вихідні дані роботи

Тип нейронної мережі: класична тришарова,
тип навчання: з вчителем,
призначення: розпізнавання рукописних символів.

4. Перелік питань, які потрібно розробити

1. Вступ. 2. Передпроектний аналіз. 3. Моделювання нейронної мережі.
4. Програмна реалізація, тренування і тестування нейронної мережі.
5. Техніко-економічне обґрунтування. 6. Охорона праці. 7. Загальні висновки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Слайд 1. Характеристика кваліфікаційної роботи. Слайд 2. Будова і принцип роботи нейрону. Слайд 3. Структура нейронної мережі. Слайд 4. Пряме розповсюдження сигналу в мережі. Слайд 6. Зворотне розповсюдження помилки. Слайд 7. Навчання через оновлення вагових коефіцієнтів. Слайд 8. Засоби реалізації. Слайд 9. Програмний клас нейронної мережі. Слайд 10. Тренувальні дані. Слайд 11. Тестувальні дані. Слайд 12. Результат тестування. Слайд 13. Залежність ефективності від коефіцієнту навчання. Слайд 13. Залежність ефективності від кількості епох навчання. Слайд 14. Техніко-економічні показники. Слайд 15. Висновки.

6. Консультанти по роботі, із зазначенням розділів роботи, що стосуються їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Економіка	*Phd, ст. викл. Богданов О.О.*		
Охорона праці	*к.т.н., доц. Нєнов О.Л.*		
Нормоконтроль	*Орел А.С.*		

7. Дата видачі завдання *30.08.2023*

Керівники *Сергій АРТЕМЕНКО*

Андрій ОРЕЛ

Завдання прийняв до виконання *Микола ГАЛАН*

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	*Дослідженні предметної області нейронних мереж*	*26.10.2023*	
2.	*Огляд засобів побудови нейронних мереж*	*30.11.2023*	
3.	*Моделювання структури нейронної мережі*	*28.01.2023*	
4.	*Моделювання процесів розповсюдження сигналів*	*15.02.2024*	
5.	*Програмна реалізація нейронної мережі*	*27.03.2024*	
6.	*Дослідження реалізованої нейронної мережі*	*15.04.2024*	
7.	*Підготовка техніко-економічної частини*	*29.04.2024*	
8.	*Підготовка розділу охорони праці*	*27.05.2024*	
9.	*Оформлення пояснювальної записки*	*27.05.2024*	
10.	*Оформлення графічної частини та лістингу*	*29.05.2024*	

Здобувач-дипломник _____ *Микола ГАЛАН*

Керівники роботи _____ *Сергій АРТЕМЕНКО*

_____ *Андрій ОРЕЛ*

Несу відповідальність за ідентичність електронного та друкованого варіантів кваліфікаційної роботи, даю згоду на обробку персональних даних та не заперечую проти розміщення кваліфікаційної роботи на офіційних web-ресурсах ОНТУ.

Підтверджую, що в кваліфікаційній роботі відсутні порушення норм академічної доброчесності.

Здобувач-дипломник *Микола ГАЛАН* _____

АНОТАЦІЯ

Робота присвячена моделюванню і дослідженню нейронної мережі і тестування її для вирішення задачі розпізнавання рукописних символів.

В першому розділі розглянуті основні поняття в області нейронних мереж, принципи їх функціонування і принципи моделювання нейронних мереж. Також виконаний порівняльний аналіз інструментів побудови нейронних мереж, на основі якого було обрано загальний підхід до подальшого проектування і реалізації нейронної мережі.

В другому розділі виконано математичне моделювання тришарової нейронної мережі, призначеної для вирішення задач розпізнавання, проаналізовані питання прямого і зворотного розповсюдження сигналів в мережі, оновлення вагових коефіцієнтів на основі даних про помилки і особливості підготовки вхідних даних для передавання на вхід нейронної мережі.

В третьому розділі обгрунтовано засіб реалізації нейронної мережі, виконано програмну реалізацію нейронної мережі у вигляді класу на мові програмування Python і здійснено її навчання розпізнаванню рукописним цифрам на основі бази даних MNIST. Також виконано дослідження шляхів поліпшення результатів розпізнавання і підбору оптимальних параметрів мережі. На основі двох підготовлених наборів з власноруч написаних цифр протестовано навчену мережу на предмет ефективності їх розпізнавання.

В роботі також досліджено суміжні питання економічного обґрунтування дослідження та охорони праці користувача.

Ключові слова: *нейронна мережа, Python, тренування, градієнтний спуск.*

ABSTRACT

The work is devoted to the modeling and research of a neural network and testing it to solve the problem of recognizing handwritten characters.

In the first chapter, the main concepts in the field of neural networks, the principles of their operation and the principles of modeling neural networks are considered. A comparative analysis of tools for building neural networks was also performed, on the basis of which a general approach to the further design and implementation of a neural network was chosen.

In the second chapter, mathematical modeling of a three-layer neural network designed to solve recognition problems is performed, issues of direct and reverse propagation of signals in the network are analyzed, weighting coefficients are updated based on error data and features of input data preparation for transmission to the input of the neural network.

In the third section, the neural network implementation tool is substantiated, the neural network is implemented in the form of a class in the Python programming language, and it is taught to recognize handwritten digits based on the MNIST database. A study of ways to improve recognition results and selection of optimal network parameters was also carried out. Based on two prepared sets of handwritten numbers, the trained network was tested for their recognition efficiency.

Related issues of economic justification of research and user labor protection are also investigated in the work.

Keywords: *neural network*, *Python*, *training*, *gradient descent*.

ЗМІСТ

стор.

ВСТУП

Штучні нейронні мережі сьогодні стають де-факто основним інструментом в сфері штучного інтелекту та машинного навчання завдяки своїм досить широким можливостям та здатності до вирішення різноманітних завдань, які були і залишаються досить складними для традиційних комп'ютерних засобів. Постійний розвиток технологій обробки даних, збільшення обчислювальної потужності та обсягів і доступності даних і, головне, останні дослідження і практичні результати в галузі штучного інтелекту призвели сьогодні до вибухоподібного розвитку сервісів на основі штучних нейронних мереж. Нейронні мережі ефективно використовуються сьогодні у багатьох сферах, включаючи дизайн, програмування, медицину, фінанси, автомобільну промисловість, рекламу та багатьох інших. На даний час можливості великих штучних нейронних мереж в поєднанні з постійно зростаючими потребами користувачів і компаній у вирішенні «інтелектуальних» завдань роблять їх незамінним інструментом для інновацій та прогресу у багатьох галузях.

Хоча сьогодні існує багато великих нейронних мереж для генерації тексту, зображень, аудіо і відео, які є загально відомими і продовжують стрімко розвиватися, створення нових невеликих нейронних мереж також може бути доцільним у певних випадках. Невеликі нейронні мережі можуть бути корисними для вирішення конкретних завдань, які не потребують великої обчислювальної потужності або масштабного обсягу даних. Вони можуть бути більш ефективними та оптимізованими для певних конкретних вимог, а також мають потенціал для використання в ресурсозберігаючих або вбудованих системах. Це обумовлює актуальність даної роботи.

При цьому слід зауважити, що при розробці невеликих мереж існують і певні ризики, оскільки недостатня потужність або недостатні дані можуть призвести до низької ефективності моделі. Враховуючи це, рішення щодо того, чи доцільно створювати конкретну невелику нейронну мережу,

залежить від контексту задачі та ресурсів, доступних для розробки та навчання моделі.

Об’єктом дослідження в даній роботі є процес побудови, навчання і використання штучних нейронних мереж.

Предметом дослідження є принципи і методи побудови, навчання і використання штучних нейронних мереж певного призначення.

Метою даної роботи є підвищення ефективності способів побудови, навчання і використання штучних нейронних мереж певного призначення.

Для здійснення поставленої мети в роботі вирішуються такі завдання:

- аналіз базових принципів функціонування і моделювання нейронних мереж;
- аналіз сучасних підходів та інструментальних засобів побудови нейронних мереж;
- моделювання нейронної мережі і основних процесів її функціонування;
- постановка задачі і вибір засобу реалізації нейронної мережі;
- програмна реалізація нейронної мережі;
- навчання нейронної мережі виконанню поставленої задачі;
- прикладне дослідження з використанням реалізованої нейромережі з метою перевірки її працездатності і виявлення ключових параметрів її роботи;
- техніко-економічний аналіз і обґрунтування розробки;
- дослідження питань охорони праці користувача-дослідника.

Наукова новизна роботи полягає в отриманні залежностей ефективності роботи нейронної мережі від основних показників, а також виявленні областей їх оптимальних значень при заданих умовах.

РОЗДІЛ 1
СУЧАСНИЙ СТАН ГАЛУЗІ ШТУЧНИХ НЕЙРОННИХ МЕРЕЖ

1.1 Основні поняття і принципи функціонування нейронних мереж

Штучна нейронна мережа – це математична модель, а також її програмне або апаратне втілення, побудована за принципом організації біологічних нейронних мереж – мереж нервових клітин живого організму [6].

Класичні комп'ютери в основі своїй – це, по суті своїй, обчислювачі, які швидко виконують арифметичні операції. Це дозволяє їм успішно вирішувати різноманітні завдання, пов'язані з числами, такі як підсумовування числових даних, аналіз даних для визначення продажів, обчислення податків, побудова графіків тощо. Перегляд відео або прослуховування музики через Інтернет представляють для комп'ютерів дещо більшу, але все ж не принципову складність. Так, реконструкція відеокадру теж відбувається за допомогою арифметичних операцій. Хоча складність алгоритмів обробки мультимедійних потоків може вражати, це, звісно, ще не інтелект: комп'ютер просто слідує інструкціям.

Існує багато задач, які є набагато більш складними для класичних комп'ютерів. Наприклад, люди (і багато з тварин) можуть досить швидко і точно розпізнавати об'єкти навколо і, зокрема, на зображеннях. Під час аналізу зображень наш мозок опрацьовує величезний обсяг інформації. У порівнянні з цим, комп'ютерам важко справлятися з подібним завданням, оскільки вони не мають людського інтелекту. В роботі над створенням штучного інтелекту приділяється значна увага саме таким завданням, оскільки комп'ютери працюють швидше і без втоми. Тому метою багатьох наукових досліджень штучного інтелекту є пошук принципів або алгоритмів, які базуються на нових підходах до вирішення подібних «інтелектуальних» задач.

У комп’ютерних системах є канали введення та виведення інформації, між якими відбуваються обчислення з даними. У випадку нейронних мереж це не так. Якщо точні принципи функціонування якоїсь системи не є відомими, можна спробувати отримати уявлення про її роботу, використовуючи модель з регульованими параметрами. Якщо б, наприклад, ми не знали, як арифметично перетворити кілометри на милі, то могли б використати для цієї мети лінійну функцію як модель з регульованим нахилом. Систему, яка вирішує подібну задачу, іноді називають прогнозною машиною, оскільки вона отримує вхідні дані та робить певний прогноз щодо того, якими мають бути вихідні дані. Можна поліпшувати прогноз, налаштовуючи внутрішній параметр (або декілька параметрів) системи на основі величини помилки, яку можна визначити, порівнюючи прогноз з відомим точним значенням. Налаштування параметрів на основі порівняння результатів моделі з точними результатами у відомих прикладах є досить ефективним способом покращення таких моделей.

Одним з типів задач, які вирішуються за допомогою прогнозних машин, є класифікація об’єктів. Багато з таких задач передбачають побудову лінійного класифікатора, тобто лінійної залежності, яка дозволяє відділити одну підмножину об’єктів від іншої і, таким чином, розділити їх на класи. Параметрами лінійного класифікатора є зсуви і нахил відповідної прямої відносно осей координат. Різні нахили прямої, зокрема, призводитимуть до різних помилок (неточностей) у використанні даних варіантів лінійного класифікатора. Щоб знайти співвідношення між вихідною помилкою класифікатора та параметром регульованого нахилу достатньо мати базові математичні знання. Знаючи це співвідношення, можна визначити величину зміни нахилу, необхідну для усунення вихідної помилки.

Проте, недолік прямолінійного підходу до налаштування параметрів полягає в тому, що модель оновлюється лише для досягнення найкращого результату лише для останнього навчального прикладу, тоді як всі попередні приклади не беруться до уваги. Це також зменшує ефективність використання системи в майбутніх випадках. Одним зі способів усунення

цього недоліку є зменшення величини оновлень за допомогою коефіцієнта швидкості навчання, щоб жоден окремий навчальний приклад не домінував у процесі навчання.

Також навчальні приклади, взяті з реальної практики, можуть бути спотворені шумом або містити помилки. Згладжування оновлень сприяє обмеженню впливу таких хибних прикладів.

Слід зазначити, що простий лінійний класифікатор не може відокремити області даних належним чином, якщо ці дані не управляються простим лінійним процесом. Один з таких прикладів – дані, які керуються логічною функцією виключного «АБО». В деяких випадках ця проблема може бути вирішена використанням набору лінійних класифікаторів.

1.2 Будова нейрону

Довгий час мозок тварин ставив вчених в глухий кут, оскільки навіть у маленьких істот він часто проявляє такі можливості, які є недоступними для цифрових комп'ютерів з великою кількістю електронних обчислювальних елементів і значним об'ємом пам'яті, які працюють на частотах, недосяжних для живого мозку. Тоді вчені зосередили увагу на архітектурних відмінностях. У традиційних комп'ютерах дані обробляються послідовно, за чітко встановленими правилами без неоднозначності та нечіткості. З іншого боку, поступово ставало зрозуміло, що мозок тварин, незважаючи на видиму повільність його робочих ритмів у порівнянні з комп'ютерами, обробляє сигнали паралельно, і що невизначеність є суттєвою відмінною рисою його діяльності.

На рисунку 1.1 показано будову базової структурно-функціональної одиниці біологічного мозку – нейрона.

Усі різновиди нейронів передають електричні сигнали від одного кінця нейрона до іншого – від дендритів через аксони до терміналей. Потім ці сигнали передаються від одного нейрона до іншого. Завдяки цьому механізму істота спроможна сприймати світло, звук, дотик, тепло та інше.

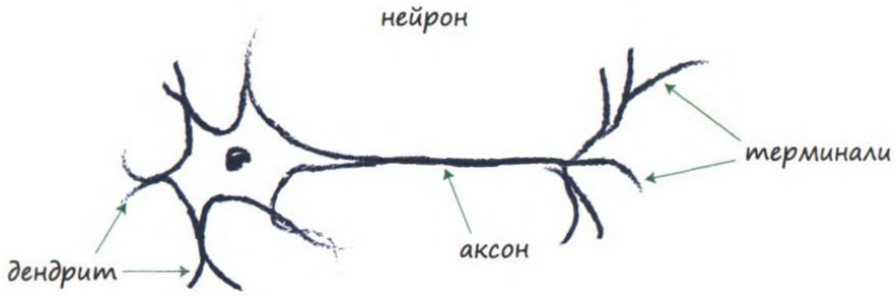

Рис. 1.1 – Будова біологічного нейрону

Сигнали від спеціалізованих рецепторних нейронів передаються через нервову систему до мозку, який в основному також складається з нейронів.

Мозок людини містить близько 100 мільярдів нейронів. Мозок дрозофіли складається з приблизно 100 тисяч нейронів, а мозок нематоди має лише 302 нейрони, проте вони здатні вирішувати досить складні завдання, такі як пошук їжі, уникнення небезпеки тощо. Складні механізми роботи мозку (такі, як, наприклад, наявність свідомості) ще залишаються загадкою, але на сьогодні вже досить багато відомо про нейрони, щоб запропонувати різні способи вирішення завдань.

Нейрон отримує вхідний електричний сигнал і генерує інший, вихідний сигнал. Це дещо нагадує моделі класифікатора, які отримують дані, виконують обчислення і видають результат. Але нейрони не є простими лінійними функціями: вони пригнічують вхідний сигнал, поки він не перевищить певне порогове значення для генерації вихідного сигналу. Таким чином, нейрони пропускають лише сильні сигнали, які несуть корисну інформацію, а не слабкий шум.

Функція, яка отримує вхідний сигнал, але генерує вихідний з урахуванням порогового значення, називається функцією активації. Існує багато таких функцій, наприклад, ступенева: для слабких вхідних значень вона виводить нуль, а якщо вхідний поріг перевищений, – генерує максимальний рівень сигналу (в біології такий нейрон називається збудженим).

Для створення штучних нейронних мереж часто замість ступеневої функції застосовують *S*-подібну функцію, яка називається сигмоїдою або логістичною кривою (рис. 1.2). Вона описується наступною формулою:

$$y = \frac{1}{1+e^{-x}}. \qquad (1.1)$$

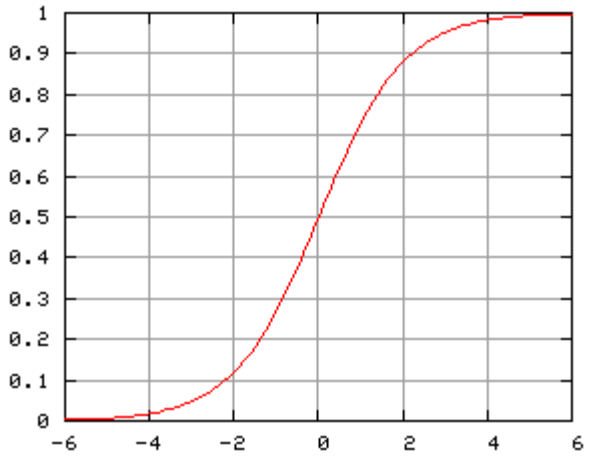

Рис. 1.2 – Графік сигмоїди (логістичної кривої)

Як видно з графіку, базова сигмоїда перетинає вісь *y* при значенні 0,5, а усі її можливі значення знаходяться у межах від 0 до 1.

Іноді для побудови нейронних мереж використовуються інші функції активації, проте сигмоїда доволі проста і дуже популярна, тому є хорошим вибором у більшості випадків. Крім того, наявний досвід показує, що виконувати розрахунки з сигмоїдою набагато простіше, ніж з будь-якою іншою *S*-подібною функцією [19].

Розглянемо принципи моделювання штучного нейрона (рис. 1.3).

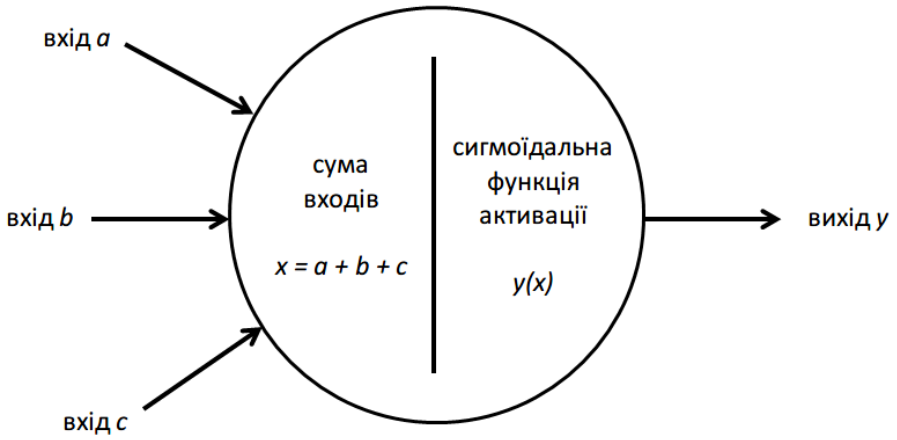

Рис. 1.3 – Принципова схема моделі нейрону

Нейрон має декілька входів. Ми комбінуємо їх і отримуємо суму, яка є вхідним значенням для сигмоїди. Якщо комбінований сигнал слабкий, сигмоїда пригнічує вихідний сигнал. Якщо сума велика, функція збуджує нейрон. Нейрон може збудитися навіть якщо кожен сигнал слабкий, але разом вони перевищують поріг.

Електричні сигнали сприймаються дендритами, де вони комбінуються для формування більш сильного сигналу. Якщо цей сигнал перевищує поріг, нейрон збуджується, і сигнал передається через аксон до терміналів, звідки він надходить на дендрити наступного нейрона. Зв’язані таким чином нейрони схематично зображені на рис. 1.4. Як видно на рисунку, кожен нейрон приймає вхідний сигнал від декількох нейронів, які знаходяться перед ним, і, у свою чергу, також передає сигнал багатьом іншим у випадку збудження.

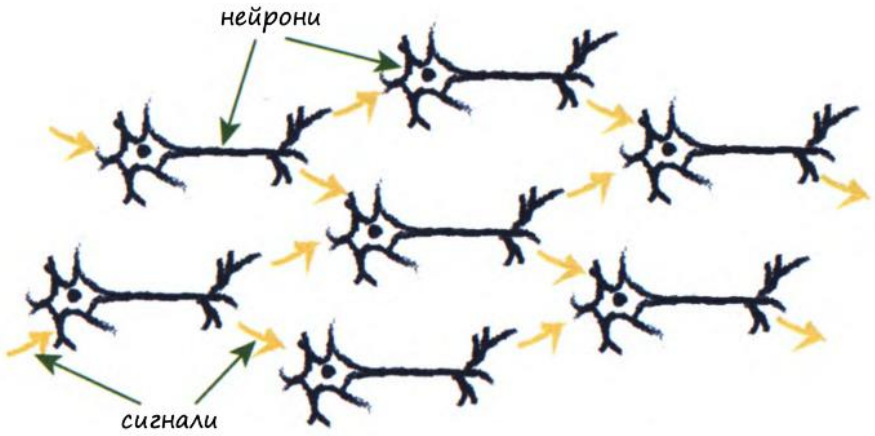

Рис. 1.4 – Зв’язані нейрони

1.3 Принцип моделювання нейронної мережі

Одним із способів відтворення поведінки живих нейронів у штучній моделі є створення багатошарових нейронних структур, в яких кожен нейрон з’єднаний з кожним з нейронів у попередньому та наступному шарах. Цю ідею ілюструє рис. 1.5, який показує структуру тришарової нейронної мережі. Середній шар в такій мережі називається прихованим.

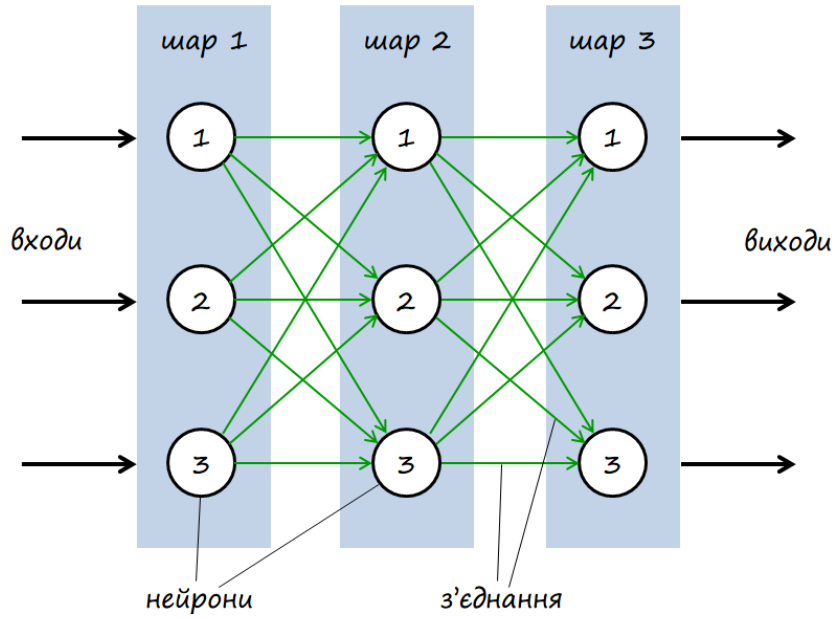

Рис. 1.5 – Тришарова структура нейронної мережі

Для моделювання процесу навчання мережі треба визначити частину цієї структури, яка має бути відповідальною за навчання. Найбільш очевидною величиною для регулювання є сила зв'язку між вузлами. В межах вузла можна також додатково контролювати суму вхідних значень або форму сигмоїди. На практиці часто обмежуються більш простим підходом, оскільки він успішно працює, дозволяючи вирішувати поставлені завдання.

Для регулювання сили зв'язку кожному з'єднанню між нейронами i та j призначається певна вага $w_{i,j}$. Низький ваговий коефіцієнт послаблюватиме сигнал, високий – посилюватиме його.

Треба зазначити, що кожен вузол шару не зобов'язаний бути зв'язаним з кожним з вузлів попереднього і наступного шарів. Взагалі, шари можна з'єднувати між собою будь-яким можливим способом. В даній роботі не розглядаються інші можливі способи з тієї причини, що завдяки однорідності описаної схеми повного взаємного з'єднання нейронів закодувати її у вигляді комп'ютерних інструкцій набагато простіше, ніж будь-яку іншу схему. Крім того, наявність більшої кількості зв'язків, ніж обов'язковий мінімум, який може знадобитися для вирішення певної задачі,

не заподіє ніякої шкоди. Якщо додаткові зв’язки не потрібні, то процес навчання ослабить їх вплив. Як тільки мережа навчиться покращувати свої вихідні значення за допомогою уточнення вагових коефіцієнтів зв’язків всередині мережі, деякі ваги обнуляться (або стануть близькими до нуля) і, таким чином, ці зв’язки не матимуть впливу на мережу, оскільки їх сигнали не будуть передаватися.

1.4 Порівняльний аналіз інструментів побудови нейронних мереж

У процесі розробки виникла необхідність проаналізувати способи і технології вирішення задачі розпізнавання рукописного тексту за допомогою нейронної мережі, а також обрати конкретний підхід для подальшого проектування.

Програмне забезпечення для нейромереж – це основний набір інструментів для створення, навчання та розгортання штучних нейронних мереж. Поєднання алгоритмів машинного навчання з адаптованістю мов сценаріїв робить його зручним інструментом для фахівців в області даних, моделей машинного навчання та дослідників.

Для роботи в галузі науки про дані, особливо при вирішенні таких задач як прогнозування складних закономірностей, комп’ютерний зір, розпізнавання рукописного тексту, адаптивний інтелектуальний аналіз даних тощо часто потрібні певні інструменти, які дозволяють будувати потрібні нейронні мережі і навчати їх.

В даній роботі були проаналізовані декілька з таких інструментів, здатних вирішувати широкий спектр завдань. Їх призначення, а також переваги і недоліки зведені в табл. 1.1.

Метою цього дослідження є вибір конкретного підходу та засобів щодо подальшої розробки нейронної мережі та її тренування для досягнення оптимальних результатів.

Таблиця 1.1

Порівняльна характеристика інструментів побудови нейронних мереж

Назва	Призначення	Переваги	Недоліки
Chainer	Створення графів динамічних обчислень	• динамічний підхід до архітектури "визначення за запуском"; • велика бібліотека зумовлених шарів та функцій; • ефективні обчислення на графічному процесорі з підтримкою CUDA	• більш крута крива навчання для початківців; • слабка підтримка спільноти; • доступно небагато сторонніх розширень
SuperLearner	Реалізація методологій ансамблевого навчання	• пакет з відкритим вихідним кодом, доступний безкоштовно; • комплексні ансамблеві методології; • інтеграція із різними алгоритмами на основі мови R; • зручний API	• обмежено середовищем програмування R; • може бути ресурсомістким з великими наборами даних; • потрібне добре розуміння ансамблевих методологій задля досягнення оптимальних результатів
NVIDIA Deep Learning AMI	Прискорення графічного процесора, інтегрованого в AWS.	• попередньо встановлено багато фреймворків глибокого навчання; • оптимізовано для графічних процесорів; • тісна інтеграція із основними сервісами AWS.	• обмежено екосистемою AWS; • може виявитися дорогим для тривалих ресурсомістких завдань

Продовження таблиці 1.1

Назва	Призначення	Переваги	Недоліки
Swift AI	Бізнес-аналітика на основі даних.	• широкі можливості прогнозної і бізнес-аналітики; • інтеграція з основними рішеннями зберігання даних	• крута крива навчання для новачків; • щорічне виставлення рахунків не для всіх є оптимальним варіантом
NVIDIA DIGITS	Інтерактивна візуалізація глибокого навчання	• сумісність із основними платформами глибокого навчання; • інтуїтивно зрозумілий інтерфейс управління експериментами	• переважно адаптований для користувачів графічних процесорів NVIDIA; • для оптимальної продуктивності може знадобитися спеціальне обладнання
Keras	Глибоке навчання з підтримкою модульності та швидкого експериментування	• зручний API; • модульна конструкція; • інтеграція з основними бекендами глибокого навчання	• крута крива навчання для новачків у глибокому навчанні; • невелика продуктивність для великомасштабних моделей; • залежність від бекенда для розширених функцій
Neuton AutoML	Автоматизована побудова та вибір моделей	• автоматизоване проектування функцій та попередня обробка; • інтеграція із основними платформами даних; • оптимізований вибір моделі	• високі ціни; • потрібне розуміння машинного навчання повного використання; • обмежена підтримка нішевих або спеціалізованих моделей

Продовження таблиці 1.1

Назва	Призначення	Переваги	Недоліки
Caffe	Модульність при глибокому навчанні	• висока модульність для різноманітних архітектур нейромереж; • сумісність з численними графічними процесорами; • інтеграція з Python та MATLAB	• складність для новачків; • обмежена підтримка спільноти; • деякі функції потребують ручного налаштування для оптимальної продуктивності
Synaptic.js	Нейромережі у середовищах JavaScript	• пропонує ряд архітектур, що навчаються; • плавна інтеграція з node.js та іншими JS-фреймворками	• потрібне глибоке розуміння нейромереж; • обмежена документація; • надлишковість для простих задач
Microsoft Cognitive Toolkit	Інструменти глибокого навчання від Microsoft, що масштабуються	• оптимізовано до роботи з кількома графічними процесорами; • підтримка великих ресурсів Microsoft; • інтеграція Azure для масштабування та розгортання	• складніше для новачків; • потенційна залежність від екосистеми Microsoft
Google Cloud Deep Learning Containers	Програми глибокого навчання, інтегровані до Google Cloud	• передвстановлені фреймворки та бібліотеки; • забезпечує узгодженість середовища на протязі всього робочого процесу; • інтеграція із сервісами Google Cloud	• найкраще підходить лише для тих, хто працює в екосистемі Google Cloud; • менша гнучкість у порівнянні з середовищами користувача; • витрати варіюються залежно від використання хмари

Проведений аналіз інструментів побудови нейронних мереж показав, що усі ці інструменти є досить потужними, але усі вони спрямовані на вирішення конкретних задач. Серед них, на жаль, не виявлено задачі розпізнавання рукописного тексту. Таким чином, на даному етапі оптимальним представляється побудова (програмування) власної нейронної мережі.

Висновки до першого розділу

В першому розділі були проаналізовані основні поняття в області нейронних мереж, зокрема, принципи їх функціонування, основи будови нейрону, принципи моделювання нейронних мереж. Це дозволило перейти до моделювання нейронної мережі із заданими принципами функціонування, призначеної для вирішення задачі розпізнавання рукописних символів.

Також був виконаний порівняльний аналіз інструментів побудови нейронних мереж, на основі якого було обрано загальний підхід до подальшого проектування і реалізації нейронної мережі.

РОЗДІЛ 2
МОДЕЛЮВАННЯ НЕЙРОННОЇ МЕРЕЖІ І ПРОЦЕСІВ ЇЇ ФУНКЦІОНУВАННЯ

2.1 Розповсюдження сигналів по нейронній мережі

Для моделювання процесу розповсюдження сигналів по нейронній мережі будь-якого розміру розглянемо спочатку її фрагмент – елементарну структуру з 4 нейронів, розподілених по двох шарах. Початкові значення вагових коефіцієнтів задамо випадковим чином, оскільки в нас немає апріорної інформації про різницю зв'язків. По мірі тренування ці значення будуть прилаштовуватися під конкретну задачу.

Перший шар будемо вважати вхідним: до його нейронів функція активації не застосовуватиметься. В другому шарі треба для кожного нейрону визначати комбінований вхідний сигнал x. Ця комбінація утворюється з необроблених вхідних сигналів зв'язаних вузлів попереднього шару (i), згладжених ваговими коефіцієнтами зв'язків w_i:

$$x = \sum_i \left(i \cdot w_i \right). \tag{2.1}$$

Для структури 2 x 2 це проілюстровано на рисунку 2.1.

Після визначення вхідного сигналу кожного нейрону його треба пропустити через відповідну функцію активації (1.1).

Комп'ютерна реалізація розрахунків за формулами (1.1) і (2.1) для нейронних мереж навіть невеликих розмірів представляє певну алгоритмічну складність. Цю реалізацію можна спростити, використовуючи математичний апарат матриць, який реалізований і є доступним в більшості сучасних мов програмування. Матриці дозволять сформулювати завдання у більш простій і компактній формі та забезпечать їх швидке та ефективне виконання.

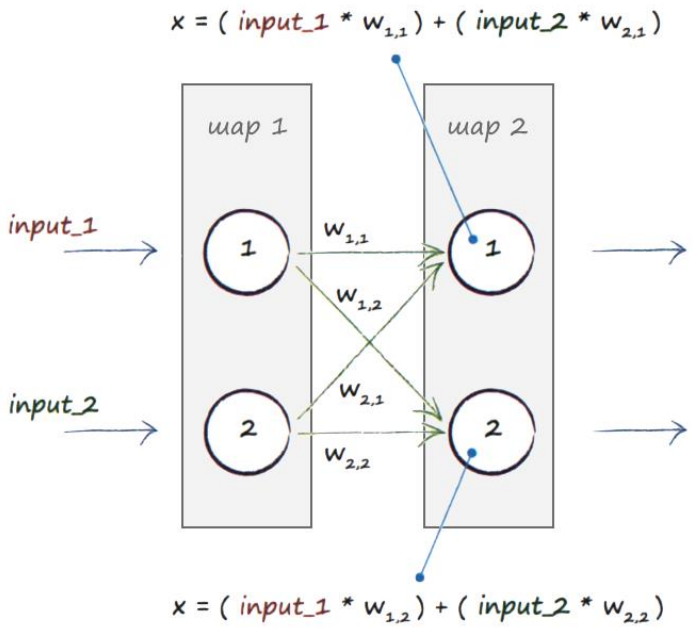

Рис. 2.1 – Приклад розрахунку вхідних сигналів
для нейронів другого шару нейронної мережі 2 x 2

Формулу (2.1) можна розглядати як формулу отримання елементу результуючої матриці при скалярному перемноженні матриць вхідних сигналів і вагових коефіцієнтів відповідних зв'язків, по яких вони надходять. Відповідно, комбінований сигнал на вході *i*-го нейрона у будь-якому шарі можна розглядати як елемент матриці **X**, отриманої в результаті перемноження матриці вагових коефіцієнтів **W** і матриці вхідних сигналів **I** даного шару:

$$\mathbf{X} = \mathbf{W} \cdot \mathbf{I}. \qquad (2.2)$$

Слід зазначити, що матриця **W** має бути транспонованою відносно структури мережі, оскільки рядки її елементів мають відповідати стовпцям нейронів у шарі.

В розгорнутому вигляді матриця **X** для нейронної мережі розміром, наприклад, 2 x 2 обчислюватиметься по правилах скалярного множення матриць за таким виразом:

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} w_{1,1} & w_{2,1} \\ w_{1,2} & w_{2,2} \end{pmatrix} \cdot \begin{pmatrix} input\_1 \\ input\_2 \end{pmatrix} = \begin{pmatrix} (input\_1 \cdot w_{1,1}) + (input\_2 \cdot w_{2,1}) \\ (input\_1 \cdot w_{1,2}) + (input\_2 \cdot w_{2,2}) \end{pmatrix}.$$

Якщо зіставити цей вираз з рис. 1.6, видно, що в результаті множення матриць ми отримуємо саме потрібні нам значення *x* комбінованих входів для нейронів другого шару.

Після перемноження за формулою (2.2) для отримання матриці **O** вихідних сигналів шару треба застосувати функцію активації *S* до кожного елемента матриці **X**:

$$\mathbf{O} = S(\mathbf{X}). \tag{2.3}$$

Для тришарової мережі принцип залишається тим самим, але процедура отримання вихідних значень виконується двічі: перший раз для другого (прихованого) шару і другий раз – для третього (вихідного) шару. На даному етапі, коли нейромережа ще не є навченою, використовуються випадкові вагові коефіцієнти як для вхідних зв'язків прихованого шару, так і для вихідного шару. Таким чином відбувається розповсюдження сигналів нейронної мережі, тобто визначається величина вихідних сигналів при заданих величинах вхідних сигналів.

Наступний крок полягає у порівнянні вихідних сигналів нейронної мережі з даними тренувального прикладу для визначення помилки. Відома величина цієї помилки дасть можливість покращити вихідні результати шляхом зміни параметрів мережі. Щоб значно зменшити помилки і уточнити результати, етапи розповсюдження сигналів мають виконуватися багаторазово.

2.2 Зворотне розповсюдження помилок у нейронної мережі

Як було зазначено вище, для навчання нейронної мережі необхідно оновлювати вагові коефіцієнти вхідних зв'язків для кожного нейрону на

основі помилок – різниць між тренувальними даними и вихідними даними нейронної мережі:

$$e = t - o, \tag{2.4}$$

де t – тренувальні дані;

o – вихідне значення.

Якщо на вихідний вузол надходить сигнал тільки від одного вузла, все просто. Розглянемо, як використати помилку вихідного сигналу при хоча б двох вхідних вузлах.

Використання всієї помилки для оновлення одного вагового коефіцієнта не представляється розумним, оскільки помилка, як правило, враховує внески всіх зв'язків, а не лише одного. Взагалі, існує певна ймовірність, що конкретна помилка відповідає лише одному зв'язку. Але якщо в процесі коригуваня буде змінено ваговий коефіцієнт, який вже має «правильне» значення, і, таким чином, його буде погіршено, то він все одно має покращитися під час наступних ітерацій навчання.

Один з можливих – і доволі грубих – способів вирішення вказаної проблеми – це розподілити помилку між всіма вузлами порівну. Але більш доцільно більшу частину помилки приписувати внескам тих зв'язків, які мають більшу вагу, оскільки вони завдяки цьому більше впливають на величину помилки.

Таким чином, вагові коефіцієнти використовуються у двох цілях. По-перше, вони враховуються при розрахунку розповсюдження сигналів нейронної мережі від вхідного шару до вихідного. По-друге, вони використовуються для розповсюдження помилки у зворотному напрямку – від вихідного шару вглиб мережі. Цей метод називається зворотним розповсюдженням помилки (зворотним зв'язком) у процесі навчання нейронної мережі.

Для прикладу розглянемо процес зворотного розповсюдження помилки в мережі 2 x 2. Як видно на рис. 2.2, помилки, в загальному випадку,

виникають на обох вузлах і визначаються у відповідності до виразу (2.4) як $e_1 = t_1 - o_1$ та $e_2 = t_2 - o_2$.

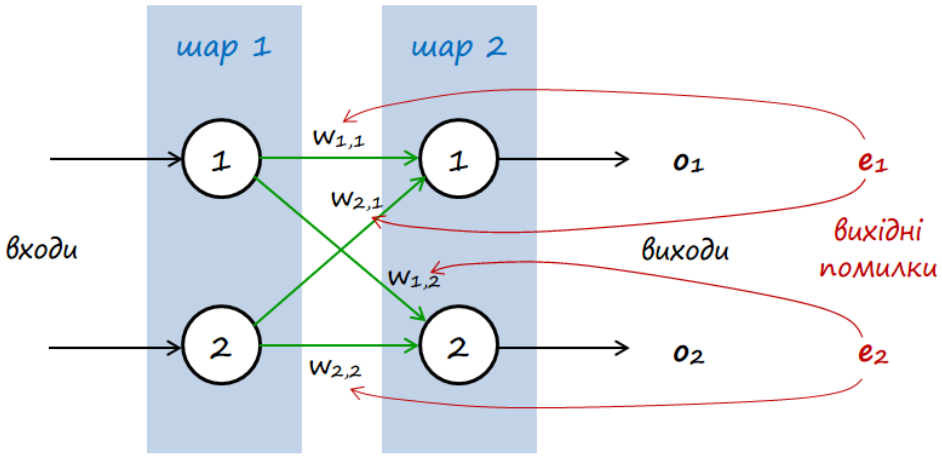

Рис. 2.2 – Зворотне розповсюдження помилки в мережі 2 x 2

Для корекції вагів внутрішніх зв'язків слід розподілити помилку кожного вихідного вузла між пов'язаними з ними вхідними вузлами пропорційно ваговим коефіцієнтам відповідних зв'язків. Цей розподіл виконується незалежно для першого і для другого вузлів другого шару, оскільки вхідні зв'язки цих вузлів не залежать один від одного.

У відповідності до цього підходу, при розподілі помилки e_1 між вузлами її частка, яка використовується для оновлення коефіцієнту $w_{1,1}$, визначається виразом $e_{1,1} = e_1 \cdot \frac{w_{1,1}}{w_{1,1} + w_{2,1}}$, а частка, яка використовується для оновлення коефіцієнту $w_{2,1}$ – виразом $e_{2,1} = e_1 \cdot \frac{w_{2,1}}{w_{1,1} + w_{2,1}}$. Аналогічним чином розраховуються частки помилки e_2 для коригування коефіцієнтів $w_{1,2}$ і $w_{2,2}$:

$$e_{1,2} = e_2 \cdot \frac{w_{1,2}}{w_{1,2} + w_{2,2}}, \quad e_{2,2} = e_2 \cdot \frac{w_{2,2}}{w_{1,2} + w_{2,2}}.$$

Розглянемо тепер мережу з трьома шарами по два вузли в кожному – 2 x 3.

Розподіл вихідної помилки кожного вузла вихідного шару між зв'язками з проміжним прихованим шаром відбувається за розглянутою вище процедурою. Наприклад, для зв'язків першого вузла вихідного шару позначимо ці помилки як $e_{п1,1}$ та $e_{п2,1}$. Вони визначатимуться як:

$$e_{п1,1} = e_1 \cdot \frac{w_{п1,1}}{w_{п1,1} + w_{п2,1}}, \quad e_{п2,1} = e_1 \cdot \frac{w_{п2,1}}{w_{п1,1} + w_{п2,1}}.$$

Помилки для зв'язків другого вузла вихідного шару, відповідно, визначатимуться як:

$$e_{п1,2} = e_2 \cdot \frac{w_{п1,2}}{w_{п1,2} + w_{п2,2}}, \quad e_{п2,2} = e_2 \cdot \frac{w_{п2,2}}{w_{п1,2} + w_{п2,2}}.$$

Узагальнюючи ці вирази для довільної кількості n вузлів у кожному шарі мережі, можна записати вираз для усіх помилок зв'язків i-го вузла вихідного шару; для j-го зв'язку:

$$e_{пj,i} = e_i \cdot \frac{w_{пj,i}}{\sum_{k=1}^{n} w_{пk,i}}, \; j = 1..n. \tag{2.5}$$

Далі треба визначитися з тим, як визначити помилку для кожного з вузлів проміжного шару для подальшого зворотного розповсюдження. На відміну від вихідного шару, необхідні для цього цільові (тренувальні) вихідні значення для проміжних вузлів відсутні.

Принцип вирішення описаної проблеми розглянемо знов на мережі 2 x 3. З кожним із двох зв'язків, які виходять з будь-якого вузла проміжного шару, пов'язана певна помилка. Зокрема, для першого вузла проміжного шару, зокрема, це будуть помилки $e_{п1,1}$ та $e_{п1,2}$. Якщо об'єднати помилки цих двох зв'язків, можна отримати загальну помилку для цього проміжного вузла:

$$e_{п1} = e_{п1,1} + e_{п1,2} = e_1 \cdot \frac{w_{п1,1}}{w_{п1,1} + w_{п2,1}} + e_2 \cdot \frac{w_{п1,2}}{w_{п1,2} + w_{п2,2}}.$$

Узагальнюючи, можна записати вираз для об'єднаної помилки *j*-го вузла проміжного шару в мережі 2 x 3:

$$e_{\text{пj}} = e_{\text{пj},1} + e_{\text{пj},2},$$

а також в мережі n x 3 з довільною кількістю вузлів в кожному шарі:

$$e_{\text{пj}} = \sum_{i=1}^{n} e_{\text{пj},i} = \sum_{i=1}^{n} \left(e_i \cdot \frac{w_{\text{пj},i}}{\sum_{k=1}^{n} w_{\text{пj},k}} \right), \; j = 1..n, \tag{2.6}$$

де $e_{\text{пj},i}$ – помилка зв'язку *j*-*i* між *j*-м вузлом проміжного шару та *i*-м вузлом вихідного шару.

Таким самим способом можна об'єднувати помилки і для вузлів вхідного шару.

Записи множин значень в мережі можна також здійснювати з використанням апарату матричної алгебри. Це дозволяє не лише скоротити довжину відповідних виразів, а й забезпечити більш високу ефективність комп'ютерних розрахунків, оскільки комп'ютери можуть використовувати повторюваний шаблон обчислень для прискорення виконання відповідних операцій.

В мережі з двома вузлами в кожному шарі матриця помилок вихідного шару запишеться у вигляді:

$$\mathbf{E}_{\text{вих}} = \begin{pmatrix} e_1 \\ e_2 \end{pmatrix}.$$

Матриця помилок проміжного прихованого шару запишеться у вигляді:

$$\mathbf{E}_{\text{пр}} = \mathbf{W}^{\text{T}}_{\text{пр,вих}} \cdot \mathbf{E}_{\text{вих}} = \begin{pmatrix} \frac{w_{1,1}}{w_{1,1} + w_{2,1}} & \frac{w_{1,2}}{w_{1,2} + w_{2,2}} \\ \frac{w_{2,1}}{w_{2,1} + w_{1,1}} & \frac{w_{2,2}}{w_{2,2} + w_{1,2}} \end{pmatrix} \cdot \begin{pmatrix} e_1 \\ e_2 \end{pmatrix}, \tag{2.7}$$

де $\mathbf{W}^{\mathrm{T}}_{\text{пр,вих}}$ – транспонована матриця вагових коефіцієнтів зв'язків між прихованим і вихідним шарами.

Як видно з останнього виразу, при перемноженні матриць відбувається множення вихідних помилок на пов'язані з ними ваги w_{ij}. Чим більшою є вага, тим більша частка помилки передається назад у прихований шар.

У дробах, що є елементами матриці вагів, нижня частина відіграє роль нормуючого множника. Якщо знехтувати цим нормуванням, можна втратити лише масштабування помилок, що передаються механізмом зворотного зв'язку. Таким чином, вираз $e_1 \cdot \frac{w_{1,1}}{w_{1,1} + w_{2,1}}$ спроститься до $e_1 \cdot w_{1,1}$. Після спрощення отримаємо наступне рівняння:

$$\mathbf{E}_{\text{пр}} = \begin{pmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \end{pmatrix} \cdot \begin{pmatrix} e_1 \\ e_2 \end{pmatrix}. \qquad (2.8)$$

Дослідження показують, що моделі без нормування вагових коефіцієнтів працюють так само ефективно, як і з нормуванням [19]. Навіть у тих випадках, коли у зворотному напрямку розповсюджуються надто великі або надто малі помилки, мережа сама виправить це при виконанні наступних ітерацій навчання. Головне, що при зворотному поширенні помилок враховуються вагові коефіцієнти зв'язків, завдяки чому розподіляється відповідальність за помилки, що виникають.

2.3 Оновлення вагових коефіцієнтів

У попередньому підрозділі було показано, як виконати зворотне розповсюдження помилок до кожного шару мережі. Далі потрібно змінити ваги зв'язків для покращення відповіді нейронної мережі.

З розглянутих принципів побудови нейрону слідує, що сигнали вузлів сумуються з урахуванням їх ваги, а потім до них застосовується сигмоїда.

Механізм оновлення ваги для зв'язків між цими вузлами представляє певну проблему. Використання алгебри для прямого обчислення ваги утруднене через громіздкість викладень: існує занадто багато комбінацій ваг та функцій, які залежать від інших функцій, що потрібно комбінувати при аналізі сигналу по мережі. Навіть у невеликій нейронній мережі з трьома шарами по три нейрони в кожному регулювання ваг між першим вхідним вузлом і другим прихованим так, щоб сигнал на виході третього вузла збільшився на 0,5, виявляється досить складним завданням.

Для прикладу, наведений нижче вираз представляє вихідний сигнал вузла *k* вихідного шару як функцію вхідних сигналів і вагових коефіцієнтів зв'язків для простої нейронної мережі з трьома шарами по три вузли (3 x 3):

$$o_k = \frac{1}{1 + e^{-\sum_{j=1}^{3}\left(w_{j,k} \cdot x_j\right)}}, \tag{2.9}$$

де $w_{j,k}$ – ваговий коефіцієнт для зв'язку, що з'єднує вузол *j* прихованого шару з вихідним вузлом *k*;

x_j – вихідний сигнал вузла *j* прихованого шару:

$$x_j = \frac{1}{1 + e^{-\sum_{i=1}^{3}\left(w_{i,j} \cdot x_i\right)}}, \tag{2.10}$$

де x_i – вхідний сигнал на вузлі *i*;

$w_{i,j}$ – ваговий коефіцієнт для зв'язку, що з'єднує вхідний вузол *i* з вузлом *j* прихованого шару.

Можна також просто перебирати випадкові вагові коефіцієнти, поки не буде отримано бажаний результат. Це так званий метод грубої сили – він іноді використовується, особливо коли задача досить важка для аналітичного моделювання. Але якщо кожен ваговий коефіцієнт може мати 1000 можливих значень у діапазоні від –1 до +1, тоді для нейронної мережі з

трьома шарами по три вузла довелося б перевірити 18 тисяч можливих варіантів. У більш типовій нейронній мережі з 500 вузлами в кожному шарі довелося б перевірити вже 500 мільйонів різних комбінацій вагових коефіцієнтів.

На практиці нерідко виникають й інші проблеми. Даних для тренування може бути недостатньо для ефективного навчання мережі. У тренувальних даних можуть бути помилки, тому припущення про їхню достовірність ставиться під сумнів. Кількість шарів або вузлів у самій мережі може бути недостатньою для вірного моделювання рішення завдання. Внаслідок цього підхід до вирішення задачі ефективного навчання мережі має враховувати вказані обмеження. Знайдене рішення може не бути ідеальним з математичної точки зору, але важливо, щоб воно було ефективним і дозволило отримати найкращі результати у вказаних умовах.

Одним з класичних методів пошуку результату в таких умовах є метод градієнтного спуску. Він застосовується для пошуку мінімуму деякої складної функції, яку неможливо описати аналітичними методами. Цей метод не забезпечує отримання абсолютно точного результату, але покроково дозволяє покращувати точність наближення до необхідної величини. В якості аналізованої функції в даному випадку виступає помилка нейронної мережі, а результатом застосування методу є її мінімізація.

Необхідним поліпшенням методу градієнтного спуску є зміна розміру коригувальних кроків, щоб уникнути перестрибування через мінімум. Незмінні кроки зазвичай призводять до нескінченних стрибків навколо мінімуму. При зменшенні розміру кроку пропорційно величині градієнта ми робимо все менші кроки по мірі наближення до мінімуму. При цьому ми припускаємо, що чим ближче помилка до мінімуму, то меншим є нахил функції в точці (похідна). Для більшості гладких (безперервно диференційованих) функцій таке припущення цілком прийнятне. Воно не буде справедливим лише по відношенню до функцій з точками розриву.

Переваги методу градієнтного спуску особливо проявляються у випадку функцій від багатьох параметрів. Функція вихідного сигналу нейронної мережі, а разом із нею і функція помилки, залежить від множини вагових коефіцієнтів, які можуть обчислюватися сотнями. Даний метод також стійкий до наявності дефектних даних і не заведе далеко в неправильну сторону, якщо функція не описується ідеально або час від часу здійснюються неправильні кроки корекції.

Функція вихідного сигналу сама по собі не є функцією помилки. На перший погляд, її можна легко перетворити її на таку функцію, оскільки, як слідує з виразу (2.4), помилка – це різниця між цільовими тренувальними значеннями та фактичними вихідними значеннями. Але якщо взяти суму помилок по всіх вузлах, то вона часто є близькою до нуля. Це пояснюється тим, що позитивна та негативна помилки взаємно скорочуються. Таким чином, проста різниця значень $(t - o)$, навіть якщо їхнє взаємне скорочення є неповним, не годиться для використання в якості загального показника того, наскільки добре навчена мережа.

Інший підхід передбачає використання абсолютної величини різниці $|t - o|$. Перевагою даного підходу є те, що в цьому випадку помилки не скорочуються. Проте, він має і недолік: похідна при цьому не є безперервною функцією поблизу мінімуму, що ускладнює використання методу градієнтного спуску. При наближенні до мінімуму нахил, а разом з ним і величина кроку зміни змінної не зменшується, а це означає ризик перескоку. Як наслідок, результат корекції постійно «стрибатиме» навколо мінімуму.

Третій варіант полягає в тому, щоб використовувати як міру помилки квадрат різниці: $(t - o)^2$. Існує декілька причин, з яких цей варіант кращий за другий, включаючи наступні:

- він спрощує обчислення, за допомогою яких визначається величина нахилу для методу градієнтного спуску;

- функція помилки є безперервно гладкою, що забезпечує нормальну роботу методу градієнтного спуску через відсутність провалів та стрибків значень функції;
- у міру наближення до мінімуму градієнт зменшується, що означає зниження ризику перескоку через мінімум, якщо використовується зменшення величини кроків.

Можливі й інші варіанти: в якості функції помилки можна використати будь-яку складну функцію. Одні з них можуть взагалі не працювати, інші – добре працювати для певного кола завдань, а треті – працювати добре майже у всіх випадках, але при цьому їхня надмірна складність може призводити до невиправданих витрат обчислювальних ресурсів.

Щоб скористатися методом градієнтного спуску, потрібно визначити нахил функції помилки по відношенню до вагових коефіцієнтів засобами диференціального обчислення. Зміну помилки E при зміні ваги $w_{j,k}$ представляє вираз:

$$\frac{\partial E}{\partial w_{j,k}} = \frac{\partial}{\partial w_{j,k}} \sum_n (t_n - o_n)^2 . \qquad (2.11)$$

Вираз (2.11) можна спростити, оскільки вихідний сигнал o_k на вузлі k залежить лише від тих зв'язків, які безпосередньо з ним з'єднані. Відповідно, можна видалити із цієї суми всі сигнали окрім того, який пов'язаний з вагою $w_{j,k}$, тобто o_k:

$$\frac{\partial E}{\partial w_{j,k}} = \frac{\partial}{\partial w_{j,k}} (t_k - o_k)^2 . \qquad (2.12)$$

Член t_k – константа, яка не змінюється при зміні $w_{j,k}$, тобто не є функцією $w_{j,k}$. У результаті залишається лише член o_k, який залежить від $w_{j,k}$: вагові коефіцієнти впливають на розповсюдження у прямому напрямі сигналів, які потім перетворюються на вихідні сигнали o_k.

Щоб розбити задачу диференціювання (2.12) на більш прості частини, можна скористатися ланцюговим правилом (правилом диференціювання складних функцій):

$$\frac{\partial E}{\partial w_{j,k}} = \frac{\partial E}{\partial o_k} \cdot \frac{\partial o_k}{\partial w_{j,k}}. \qquad (2.13)$$

Оскільки $E = (t_k - o_k)^2$, для першої частини треба взяти похідну від квадратичної функції: $-2\,(t_k - o_k)$.

В другій частині o_k – це вихідний сигнал вузла k, який є результатом застосування сигмоїди до сигналів, що потрапляють на даний вузол:

$$o_k = S\left(\sum_j w_{j,k} \cdot o_j\right), \qquad (2.14)$$

де o_j – вихідні сигнали з попереднього (прихованого) шару;

S – функція сигмоїди (1.1), яка диференціюється так:

$$\frac{\partial}{\partial x} S(x) = S(x) \cdot (1 - S(x)). \qquad (2.15)$$

Об'єднавши вирази (2.13), (2.14) і (2.15), отримаємо вираз:

$$\frac{\partial E}{\partial w_{j,k}} = -2\,(t_k - o_k) \cdot S\left(\sum_j w_{j,k} \cdot o_j\right) \cdot \left(1 - S\left(\sum_j w_{j,k} \cdot o_j\right)\right) \cdot \frac{\partial}{\partial w_{j,k}} S\left(\sum_j w_{j,k} \cdot o_j\right) =$$

$$= -2\,(t_k - o_k) \cdot S\left(\sum_j w_{j,k} \cdot o_j\right) \cdot \left(1 - S\left(\sum_j w_{j,k} \cdot o_j\right)\right) \cdot o_j.$$

Нарешті, можна позбутися множника 2, оскільки нас цікавить лише напрямок градієнту функції помилки. Таким чином, кінцевий вираз для зміни ваги $w_{j,k}$ прийме вигляд:

$$\frac{\partial E}{\partial w_{j,k}} = -\,(t_k - o_k) \cdot S\left(\sum_j w_{j,k} \cdot o_j\right) \cdot \left(1 - S\left(\sum_j w_{j,k} \cdot o_j\right)\right) \cdot o_j. \qquad (2.16)$$

Отриманий вираз (2.16) дозволяє виконувати уточнення вагових коефіцієнтів зв'язків між прихованим та вихідним шарами в нейронній мережі з трьома шарами.

Тепер потрібно знайти похідну аналогічної функції помилки для коефіцієнтів зв'язків між вхідним та прихованим шарами. При цьому треба врахувати такі відмінності:

1. Перша частина виразу для похідної, яка раніше була вихідною помилкою (цільове значення мінус фактичне значення), тепер стає рекомбінованою вихідною помилкою прихованих вузлів, яка розраховується відповідно до механізму зворотного розповсюдження помилок. Позначимо її e_j.

2. Друга частина виразу, що включає сигмоїду, залишається тією ж, але вираз із сумою, що передається функції, тепер відноситься до попередніх шарів, і тому підсумовування здійснюється за всіма вхідними сигналами прихованого шару, згладженими вагами зв'язків, що ведуть до прихованого вузла j. Позначимо цю суму як i_j.

3. Остання частина виразу має сенс вихідних сигналів o_i вузлів першого шару, і в даному випадку ці сигнали є вхідними.

Отже, вираз для обчислення градієнту функції помилки по вагових коефіцієнтах зв'язків між вхідним та прихованим шарами, матиме наступний вигляд:

$$\frac{\partial E}{\partial w_{i,j}} = -(e_j) \cdot S\left(\sum_i w_{i,j} \cdot o_i\right) \cdot \left(1 - S\left(\sum_i w_{i,j} \cdot o_i\right)\right) \cdot o_i. \qquad (2.17)$$

При коригуванні вагових коефіцієнтів за допомогою виразів (2.16) і (2.17) слід враховувати, що напрям зміни коефіцієнтів є протилежний напрямку градієнта. Крім того, зміни параметрів мають згладжуватися за допомогою коефіцієнта навчання, який можна налаштовувати з урахуванням особливостей конкретного завдання. Це відповідає виразу:

$$w'_{j,k} = w_{j,k} - \alpha \cdot \frac{\partial E}{\partial w_{j,k}}, \qquad (2.18)$$

де $w'_{j,k}$ – оновлений ваговий коефіцієнт зв'язку між вузлами j та k;

$w_{j,k}$ – попереднє (поточне) значення цього коефіцієнту;

α – коефіцієнт навчання – множник, який згладжує величину змін, щоб запобігти перескакуванням через мінімум помилки;

У виразі (2.16) компонент $\alpha \cdot \frac{\partial E}{\partial w_{j,k}}$ – це поправка, величина якої пропорційна похідній від функції помилки. Оскільки ця поправка віднімається, вага зменшуватиметься при додатній похідній і збільшуватиметься при від'ємній.

Вираз застосовується до вагових коефіцієнтів зв'язків не тільки між прихованим та вихідним, а й між вхідним та прихованим шарами. Ці два випадки відрізняються градієнтами функції помилки, вирази яких наведені вище.

На основі виразу (2.18) можна записати матрицю змін вагових коефіцієнтів:

$$\begin{pmatrix} \Delta w_{1,1} & \Delta w_{2,1} & \Delta w_{j,1} & \dots \\ \Delta w_{1,2} & \Delta w_{2,2} & \Delta w_{j,2} & \dots \\ \Delta w_{1,k} & \Delta w_{2,k} & \Delta w_{j,k} & \dots \\ \dots & \dots & \dots & \dots \end{pmatrix} = \begin{pmatrix} E_1 \cdot S_1(1-S_1) \\ E_2 \cdot S_2(1-S_2) \\ E_k \cdot S_k(1-S_k) \\ \dots \end{pmatrix} \cdot \begin{pmatrix} o_1 & o_2 & o_j & \dots \end{pmatrix}. \qquad (2.19)$$

У виразі (2.19) опущений коефіцієнт навчання α, оскільки це константа, і вона не впливає на те, як організовується матричне множення.

Матриця змін ваг містить значення поправок до вагових коефіцієнтів w_{jk} для зв'язків між вузлом j одного шару та вузлом k наступного шару. Видно, що в першій частині виразу праворуч від знака рівності використовуються значення наступного шару (вузол k), а в другій – з попереднього

шару (вузол j). Друга частина – це транспонована матриця сигналів o_j на виході попереднього шару.

Використовуючи символічний запис матриць, приведемо формулу (2.19) до виду, більш пристосованого для реалізації в програмному коді мовою, що забезпечує ефективну роботу з матрицями:

$$\Delta \mathbf{W}_{j,k} = \alpha \cdot \mathbf{E}_k \cdot \mathbf{O}_k (1 - \mathbf{O}_k) \cdot \mathbf{O}_j^{\mathrm{T}}. \qquad (2.20)$$

Як видно, вираз (2.18) сприймається і, відповідно, реалізується простіше, ніж попередні. Сигмоїди тут приховані у матрицях $\mathbf{O}_k$ вихідних сигналів вузлів.

2.4 Підготовка даних

У цьому підрозділі розглянуті рекомендації щодо того, як краще підготувати тренувальні дані та випадкові початкові значення вагових коефіцієнтів і спланувати вихідні значення для ефективного процесу навчання.

1. Вхідні та вихідні дані, а також початкові значення вагових коефіцієнтів мають бути узгоджені зі структурою мережі та специфікою конкретного завдання.

2. Поширеною проблемою є насичення мережі – ситуація, коли великі значення сигналів, часто обумовлені великими значеннями вагових коефіцієнтів, призводять до сигналів, що потрапляють в область значень градієнта функції активації, близьких до нуля. Результатом цього є зниження здатності мережі вчитися на найкращих значеннях вагових коефіцієнтів.

3. Іншу проблему становлять нульові значення сигналів або вагових коефіцієнтів. Ці значення також повністю позбавляють мережу можливості вчитися на найкращих значеннях вагових коефіцієнтів.

4. Значення вагових коефіцієнтів внутрішніх зв’язків мають бути випадковими та невеликими, але не нульовими. Як варіант, можна спробувати використати більш складні правила, що включають, наприклад, зменшення значень вагових коефіцієнтів зі збільшенням кількості зв’язків, що ведуть до вузла.

5. Вхідні сигнали повинні масштабуватись до невеликих, але ненульових значень. Зазвичай оптимальним є діапазон значень від 0,01 до 0,99 та від –1.0 до +1,0 залежно від цього, який краще відповідає специфіці завдання.

6. Вихідні сигнали повинні знаходитися в межах діапазону, який здатна забезпечити функція активації. Значення, менші або рівні 0 і більші або рівні 1, не сумісні з логістичною сигмоїдою. Установка тренувальних цільових значень за межами допустимого діапазону призведе до ще більших значень ваг і, зрештою, до насичення мережі. Непоганим діапазоном є діапазон значень від 0,01 до 0,99.

Висновки до другого розділу

В другому розділі кваліфікаційної роботи було виконано математичне моделювання тришарової нейронної мережі, призначеної для вирішення задач розпізнавання. Зокрема, були проаналізовані такі питання:

- пряме і зворотне розповсюдження сигналів;
- оновлення вагових коефіцієнтів на основі даних про помилки;
- підготування даних для розпізнавання.

При цьому були висвітлені такі концептуальні положення:

1. Помилка нейронної мережі є функцією ваги її внутрішніх зв’язків.

2. Поліпшення нейронної мережі означає зменшення цієї помилки шляхом зміни зазначених ваг.

3. Безпосередній підбір цих ваг пов’язаний зі значними труднощами. Метод градієнтного спуску полягає у ітеративному поліпшенні вагових коефіцієнтів шляхом зменшення функції помилки невеликими кроками. Кожен крок здійснюється у напрямку якнайшвидшого спуску з поточної позиції до заданого мінімуму.

4. Градієнт помилки розраховується за допомогою апарату диференціального обчислення.

Виконана робота дозволяє перейти до програмної реалізації нейронної мережі і її подальшого дослідження.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ, ТРЕНУВАННЯ І ТЕСТУВАННЯ НЕЙРОННОЇ МЕРЕЖІ

3.1 Вибір засобів реалізації нейронної мережі

Існує багато мов програмування, які широко використовуються для побудови та тренування нейронних мереж. Проаналізуємо далі найбільш популярні з них.

Найпопулярнішою мовою програмування для побудови нейронних мереж є Python. Код на Python, як правило, досить простий і читабельний. Крім того, для Python існує широкий набір бібліотек і фреймворків для машинного навчання. Найбільш популярними бібліотеками для нейронних мереж у Python є:

- TensorFlow: потужний фреймворк від Google для машинного навчання, який підтримує як глибоке, так і класичне машинне навчання;
- PyTorch: фреймворк від Facebook, відомий своєю динамічною обчислювальною графікою та зручністю використання;
- Keras: високоабстрактний API для нейронних мереж, який працює поверх TensorFlow або Theano;
- scikit-learn: бібліотека для класичного машинного навчання, яка пропонує базові можливості для нейронних мереж.

R – мова програмування, яка часто використовується в статистиці та аналізі даних. Вона також має бібліотеки для побудови нейронних мереж, такі як:

- nnet: пакет для простих нейронних мереж;
- keras: інтерфейс для Keras у R;

- tensorflow: інтерфейс для TensorFlow у R.

Java також використовується для машинного навчання та побудови нейронних мереж. При використанні Java доступні бібліотеки:

- Deeplearning4j: досить розвинений фреймворк для глибокого навчання, який працює на JVM;
- DL4J: інтерфейс для Deeplearning4j.

C++ – ще одна дуже відома мова програмування, яка часто використовується для розробки високопродуктивних застосунків. Для нейронних мереж на C++ використовують, зокрема, такі бібліотеки:

- Caffe: фреймворк глибокого навчання, відомий своєю швидкістю;
- Torch: наукова обчислювальна структура для машинного навчання, написана на C++ з інтерфейсом Lua.

JavaScript дозволяє запускати нейронні мережі безпосередньо в браузері. Основні бібліотеки JavaScript – це:

- TensorFlow.js: версія TensorFlow для JavaScript;
- Brain.js: легкий фреймворк для нейронних мереж у JavaScript.

Досить широко використовується в академічних та наукових колах для аналізу даних і машинного навчання пакет MATLAB. Він має вбудовані функції для побудови нейронних мереж через свій засіб Deep Learning Toolbox.

Julia – відносно молода мова програмування, яка поєднує в собі високу продуктивність та простоту у використанні. Вона набирає популярності у сфері машинного навчання завдяки бібліотекам, таким як Flux.jl.

Мова Scala, особливо з бібліотекою Breeze, використовується для розробки розподілених систем машинного навчання. Бібліотека Spark MLlib також підтримує машинне навчання і може використовуватися разом із Scala.

Звісно, кожна з вказаних мов має свої переваги та недоліки, і вибір мови часто залежить від конкретних вимог проекту.

Для подальшої розробки була обрана мова Python. Окрім простоти і читабельності коду, а також наявності різноманітних бібліотек Python має декілька додаткових переваг:

1. Одна з найбільших спільнот розробників у світі. Це означає, що майже для будь-якої проблеми або питання можливо швидко знайти допомогу в документації, на форумах або на спеціалізованих сайтах, таких як Stack Overflow.
2. Можливість інтеграції з іншими мовами та системами, такими як C++, Java та R, що дозволяє використовувати його разом з іншими інструментами та бібліотеками, якщо це необхідно.
3. Наявність платформ для експериментів та обчислень, таких як Google Colab та Jupyter Notebooks, що дозволяє легко експериментувати з кодом, ділитися ним та виконувати обчислення в хмарі.
4. Підтримка розподілених обчислень і роботи з великими обсягами даних через бібліотеки, такі як Dask і Apache Spark (через PySpark).
5. Підтримка інструментів для розробки та тестування, таких як pytest для написання тестів, профайлери для оптимізації коду та інструменти для безперервної інтеграції.
6. Крос-платформність: Python є крос-платформною мовою, що дозволяє писати код, який працюватиме на різних операційних системах без змін.

Всі ці переваги роблять Python оптимальним вибором для розробки нейронних мереж та машинного навчання, який пропонує баланс між простотою використання та потужністю інструментів.

Також треба було визначитися з середовищем розробки. Серед багатьох варіантів було прийнято рішення використовувати інтерактивну оболонку IPython у пакеті Anaconda із редактором коду Jupyter Notebook. Ця

оболонка надає розширені функції та інструменти для більш ефективного написання коду, його тестування та виконання. Важливими особливостями IPython є:

- інтерактивна консоль, яка дозволяє виконувати Python-код у режимі реального часу (корисно для швидкого тестування та відлагодження коду);
- магічні команди (magic commands) – спеціальні команди, які спрощують виконання типових завдань, таких як завантаження даних, вимірювання часу виконання коду, керування середовищем тощо;
- зберігання та завантаження сесій, що полегшує відновлення роботи після перерви;
- підтримка інтерактивної роботи з даними завдяки інтеграції з бібліотеками для обробки та аналізу даних (Pandas, NumPy, Matplotlib);
- інтеграція з Jupyter Notebook, який додає можливість створення та обміну інтерактивними документами з кодом, текстом, графіками та іншими елементами, які часто використовуються в наукових дослідженнях, освіті та аналізі даних;
- автодоповнення та підсвічування синтаксису, що полегшує написання коду та робить його більш читабельним;
- можливість розширення через написання власних магічних команд і модулів, що дозволяє адаптувати середовище під конкретні потреби дослідника.

Таким чином, IPython – це розвинений інструмент для інтерактивної роботи з Python, який значно розширює можливості стандартної оболонки Python.

Що стосується бібліотек розширень, вони спрощують і прискорюють практичну роботу з нейронними мережами, проте деякі задачі дослідження параметрів нейронних мереж та деяких аспектів їх роботи стають при цьому надто складними, якщо взагалі можливими. Тому оберемо підхід до

реалізації на основі класичного програмування без використання спеціалізованих бібліотек.

3.2 Програмна реалізація нейронної мережі на Python

3.2.1 Каркас класу нейронної мережі

Нейронна мережа в нашому програмному проекті буде представлятися класом з трьома основними методами:

- ініціалізація – завдання кількості вхідних, прихованих та вихідних вузлів;
- тренування – уточнення вагових коефіцієнтів у процесі обробки тренувальних прикладів, наданих для навчання мережі;
- опитування – отримання значень сигналів із вихідних вузлів після надання значень вхідних сигналів.

Скелетний код класу:

```
class neuralNetwork:
# ініціалізація нейронної мережі
def __init__():
      pass
# тренування нейронної мережі
def train():
      pass
# опитування нейронної мережі
def query():
      pass
```

3.2.2 Ініціалізація мережі

В методі ініціалізації потрібно задати кількість вузлів вхідного, прихованого та вихідного шарів. Ці дані визначають конфігурацію та розмір нейронної мережі. Замість того, щоб жорстко задавати їх у коді, ми передбачимо встановлення відповідних значень у вигляді параметрів під час створення об’єкта нейронної мережі. Завдяки цьому можна буде легко

створювати нові нейронні мережі різного розміру. Також під час створення нової нейронної мережі можна встановлювати коефіцієнт навчання α – позначимо у програмі змінну, яку ми налаштовуємо в головній програмі як *learning_rate*, а відповідний параметр функції – як *learningrate*. Функція ініціалізації *init()* у такому випадку виглядатиме так:

```
# ініціалізувати нейронну мережу
def __init__ (self, inputnodes, hiddennodes, outputnodes, learningrate):
    # задати кількість вузлів у вхідному, прихованому та вихідному шарі
    self.inodes = inputnodes
    self.hnodes = hiddennodes
    self.onodes = outputnodes
    # коефіцієнт навчання
    self.lr = learningrate
    pass
```

Наступний код створює об’єкт невеликої мережі з трьома вузлами у кожному шарі та коефіцієнтом навчання, рівним 0,3.

```
# кількість вхідних, прихованих та вихідних вузлів
input_nodes = 3
hidden_nodes = 3
output_nodes = 3
# Коефіцієнт навчання дорівнює 0,3
learning_rate = 0.3
# створити екземпляр нейронної мережі
n = neuralNetwork.(input_nodes, hidden_nodes, output_nodes, learning_rate)
```

3.2.3 Створення матриць зв’язків між шарами мережі

Наступний крок – створення мережі, що складається з вузлів та зв’язків. Найважливіша частина цієї мережі – вагові коефіцієнти зв’язків (ваги). Вони використовуються для розрахунку поширення сигналів у прямому напрямку, а також зворотного поширення помилок, і саме вагові коефіцієнти будуть уточнюватися при спробах покращити характеристики мережі.

Створимо такі матриці:

- матрицю $\mathbf{W}_{\text{вхідний\_прихований}}$ ваг для зв’язків між вхідним та прихованим шарами, розмірністю *hidden_nodes* × *input_nodes*;

– матрицю $\mathbf{W}_{\text{прихований\_вихідний}}$ для зв’язків між прихованим та вихідним шарами, розмірністю *output_nodes* × *hidden_nodes*.

Початкові значення вагових коефіцієнтів повинні бути невеликими та вибиратися випадковим чином.

Нижченаведений код за допомогою функції пакету *numpy* створює дві матриці вагових коефіцієнтів, використовуючи значення змінних *self.inodes*, *self.hnodes* і *self.onodes* для завдання відповідних розмірів кожної з них:

```
import numpy
# Матриці вагових коефіцієнтів зв'язків wih (між вхідним та прихованим)
# шарами) і who (між прихованим та вихідним шарами).
# Вагові коефіцієнти зв'язків між вузлом і та вузлом j наступного шару
# позначені як wij:
# wll w21
# wl2 w22 і т.д.
self.wih = (numpy.random.rand(self.hnodes, self.inodes) - 0.5)
self.who = (numpy.random.rand(self.onodes, self.hnodes) - 0.5)
```

Таким чином, цей невеликий код реалізує саму основу нейронної мережі – матриці зв’язків між її вузлами.

Поліпшений варіант ініціалізації вагових коефіцієнтів передбачає, що вагові коефіцієнти вибиратимуться з нормального розподілу з центром у нулі та зі стандартним відхиленням, величина якого обернено пропорційна кореню квадратному з кількості вхідних зв’язків на вузол.

За допомогою функції *numpy.random.normal()* можна отримати вибірку з нормального розподілу. Її параметрами є центр розподілу, стандартне відхилення та розмір масиву *numpy*, оскільки нам потрібна матриця випадкових чисел, а не одиночне число.

Оновлений код ініціалізації вагових коефіцієнтів:

```
self.wih = numpy.random.normal(0.0, pow(self.hnodes, -0.5),
    (self.hnodes, self.inodes))
self.who = numpy.random.normal(0.0, pow(self.onodes, -0.5),
    (self.onodes, self.hnodes))
```

Центр нормального розподілу встановлено в 0,0. Стандартне відхилення обчислюється за кількістю вузлів у наступному шарі за допомогою

функції *pow(self.hnodes, –0.5)*, яка підносить кількість вузлів до степеня –0,5, тобто бере квадратний корінь. Останній параметр визначає конфігурацію масиву *numpy*.

3.2.4 Опитування мережі

Функція *query()* (метод класу) приймає в якості аргументу вхідні дані нейронної мережі та повертає її вихідні дані. Для цього потрібно передати сигнали від вузлів вхідного шару через прихований шар до вузлів вихідного шару, щоб отримати вихідні дані. При цьому при поширенні сигналів треба згладжувати їх, використовуючи вагові коефіцієнти зв'язків між відповідними вузлами, а також застосовувати сигмоїду для зменшення вихідних сигналів вузлів.

У разі великої кількості вузлів написання для кожного з них коду на Python, що здійснює згладжування вагових коефіцієнтів, підсумовування сигналів і застосування до них сигмоїди, виявилося б надто складним. Але завдяки простій та компактній матричній формі і тому, що Python вміє ефективно працювати з матрицями, можна радикально спростити цей процес:

```
hidden_inputs = numpy.dot(self.wih, inputs)
```

Даний рядок коду Python об'єднує всі вхідні сигнали з відповідними вагами для отримання матриці згладжених комбінованих сигналів у кожному вузлі прихованого шару. Важливо, що цей код не залежить від кількості вузлів у шарах мережі.

Для отримання вихідних сигналів прихованого шару треба застосувати до кожного з них сигмоїду. Бібліотека *scipy* у Python містить набір спеціальних функцій, у тому числі функцію сигмоїди, яка називається *expit()*.

Оскільки в майбутньому може знадобитися поекспериментувати з функцією активації, настроївши її параметри або повністю замінивши іншою функцією, краще буде визначити її один раз в об'єкті нейронної мережі під час його ініціалізації. Після цього ми зможемо неодноразово посилатися на неї, так само, як на функцію *query()*. Така організація програми означає, що у разі внесення змін доведеться зробити це лише в одному місці, а не скрізь у коді, де використовується функція активації.

Нижче наведено код, що визначає функцію активації, який використовується у розділі ініціалізації нейронної мережі.

```
import scipy.special
# використання сигмоїди як функції активації
self.activation_function = lambda x: scipy.special.expit(x)
```

Цей код створює функцію з використанням так званого лямбда-виразу. Замість звичного визначення функції у формі *def ім'я()* тут використовується ключове слово *lambda*, яке дозволяє створювати функції «на місці». В даному випадку функція приймає аргумент *x* і повертає *scipy.special.expit()*, тобто сигмоїду. Функції, створювані з допомогою лямбда-виразів, є анонімними, але цій функції ми надали ім'я *self.activation_function()*. Кожного разу, коли потрібно буде використовувати функцію активації, її можна буде викликати через це ім'я.

Отже, повертаючись до нашого завдання, ми застосовуємо функцію активації до згладжених комбінованих вхідних сигналів, що надходять на приховані вузли. Відповідний код:

```
# розрахувати вихідні сигнали для прихованого шару
hidden_outputs = self.activation_function(hidden_inputs)
```

Таким чином, сигнали, що виходять із прихованого шару, описуються матрицею *hidden_outputs*.

Поширення сигналу від прихованого шару до вихідного нічим принципово не відрізняється від попереднього випадку, тому спосіб розрахунку залишається тим самим, а отже, і код буде аналогічним до попереднього.

Нижче наведено підсумковий фрагмент коду, що поєднує розрахунки сигналів прихованого та вихідного шарів.

```
# розрахувати вхідні сигнали для прихованого шару
hidden_inputs = numpy.dot(self.wih, inputs)
# розрахувати вихідні сигнали для прихованого шару
hidden_outputs = self.activation_function(hidden_inputs)
# розрахувати вхідні сигнали для вихідного шару
final_inputs = numpy.dot(self.who, hidden_outpute)
розрахувати вихідні сигнали для вихідного шару
final_outputs = self.activation_function(final_inputs)
```

3.2.5 Тренування мережі

Задачу тренування мережі можна поділити на дві частини:

1. Розрахунок вихідних сигналів для заданого тренувального прикладу. Це нічим не відрізняється від того, що ми можемо робити за допомогою функції *query()*.

2. Порівняння розрахованих вихідних сигналів з бажаною відповіддю та оновлення вагових коефіцієнтів зв'язків між вузлами з урахуванням знайдених відмінностей.

Запишемо готову першу частину тренування нейронної мережі:

```
def train(self, inputs_list, targets_list):
    # Перетворити список вхідних значень на двомірний масив
    inputs = numpy.array(inputs_list, ndmin=2).T
    targets = numpy.array(targets_list, ndmin=2).T
    # розрахувати вхідні сигнали для прихованого шару
    hidden_inputs = numpy.dot(self.win, inputs)
    # розрахувати вихідні сигнали для прихованого шару
    hidden_outputs = self.activation_function(hidden_inputs)
    # розрахувати вхідні сигнали для вихідного шару
    final_inputs = numpy.dot(self.who, hidden_outputs)
    розрахувати вихідні сигнали для вихідного шару
    final_outputs = self.activation_function(final_inputs)
    pass
```

Цей код майже збігається з кодом функції *query()*, оскільки процес передачі сигналу від вхідного шару до вихідного залишається тим самим. Єдиною відмінністю є введення додаткового параметра *targetslist*, що передається при виклику функції, оскільки неможливо тренувати мережу

без надання їй тренувальних прикладів, які включають бажані або цільові значення.

Список *targets_list* перетворюється на масив так само, як список *input_list*:

```
targets = numpy.array(targets_list, ndmin=2).T
```

Тепер ми можемо перейти до вирішення основного завдання тренування мережі – уточнення ваги на основі розбіжності між розрахунковими і цільовими значеннями.

Насамперед, необхідно обчислити помилку, яка є різницею між бажаним цільовим вихідним значенням, наданим прикладом тренувань, і фактичним вихідним значенням. Вона є різницею між матрицями (*targets – final_outputs*), що розраховується поелементно. Відповідний код виглядає так:

```
# помилка = цільове значення - фактичне значення
output_errors = targets - final_outputs
```

Далі ми маємо розрахувати зворотне розповсюдження помилок для вузлів прихованого шару. Розподіл помилок між вузлами пропорційно ваговим коефіцієнтам зв'язків з подальшою рекомбінацією їх на кожному вузлі прихованого шару відповідають раніше записаному виразу (2.7). Код, що реалізує цей вираз:

```
# помилки прихованого шару - це помилки output_errors,
# розподілені пропорційно ваговим коефіцієнтам зв'язків
# і рекомбіновані на прихованих вузлах
hidden_errors = numpy.dot(self.who.T, output_errors)
```

Отже, ми отримали те, що потрібно для уточнення вагових коефіцієнтів у кожному шарі. Для вагів зв’язків між прихованим та вихідним шарами ми використовуємо змінну *output_errors*. Для ваг зв’язків між вхідним та прихованим шарами ми використовуємо щойно розраховану змінну *hidden_errors*.

Раніше нами було отримано вираз (2.20) для оновлення ваги зв’язку між вузлом *j* та вузлом *k* наступного шару в матричній формі. Реалізуємо цей вираз в код Python.

Спочатку запишемо код для оновлення ваги зв’язків між прихованим і вихідним шарами:

```
# оновити вагові коефіцієнти зв'язків між прихованим та вихідним шарами
self.who += self.lr * numpy.dot((output_errors * final_outputs *
    (1.0 - final_outputs)), numpy.transpose(hidden_outputs))
```

Тут *self.lr* це кефіцієнт навчання, який множиться на решту виразу. Матричне множення виконується за допомогою функції *numpy.dot()*. Наступні два елементи відносяться до помилки і сигмоїдів з наступного шару. В кінці – транспонована матриця вихідних сигналів попереднього шару.

Код для уточнення вагових коефіцієнтів зв’язків між вхідним та прихованим шарами працює за подібним принципом. Скористаємося симетрією виразів і перепишемо код, замінюючи в ньому імена змінних таким чином, щоб вони належали до попередніх шарів:

```
# оновити вагові коефіцієнти зв'язків між вхідним та прихованим шарами
self.win += self.lr * numpy.dot((hidden_errors * hidden_outputs *
    (1.0 - hidden_outputs)), numpy.transpose(inputs))
```

Повний код класу нейронної мережі наведений у Додатку Б.

3.3 Навчання нейронної мережі розпізнаванню рукописних цифр

3.3.1 Формування тестових тренувальних даних

Розпізнавання рукописного тексту – це досить складна для комп’ютера задача, яка характеризується певною нечіткістю і яку вже давно намагаються вирішити з більшим або меншим успіхом. Розмір проблеми розпізнавання рукописного тексту характеризує, зокрема, те, що навіть для людей ця задача іноді є складною. Буває складно розібрати окремі, відірвані

від контексту символи, які схожі один на один, наприклад, літера *ч* і цифра *4*, і написані при цьому нерозбірливим почерком. З появою і розповсюдженням нейронних мереж у вирішенні цієї задачі спостерігається значний прогрес.

Для подальшої роботи ми будемо використовувати колекцію зображень рукописних цифр *MNIST*, яка надається авторитетним дослідником нейронних мереж Яном Лекуном для безкоштовного загального доступу за адресою в Інтернеті http://yann.lecun.com/exdb/mnist/. Ця база часто використовуються дослідниками штучного інтелекту як популярний набор для тестування різних ідей та алгоритмів. Завдяки тому, що ця колекція відома і користується популярністю, різні програми можуть тестуватися різними дослідниками з використанням однакового тестового набору.

На вказаному вище сайті також наводяться відомості щодо успішності колишніх і нинішніх спроб коректного розпізнавання цих рукописних символів. Ці дані дають можливість перевірити, наскільки наші рішення конкурентоспроможні порівняно з рішеннями, які пропонуються іншими дослідниками.

Формат бази даних MNIST не дуже зручний для безпосередньої роботи. Тому ми скористаємося еквівалентними за змістом файлами у більш простому і досить універсальному форматі CSV, які були створені іншими спеціалістами (наприклад, http://pjreddie.com/projects/mnist-in-csv/). В цих файлах окремі значення представляють звичайний текст і розділені комами. На вказаному вище сайті надані такі два файли:

- http://www.pjreddie.com/media/files/mnist_train.csv – тренувальний набір;
- http://www.pjreddie.com/media/files/mnist_test.csv – тестовий набір.

Тренувальний набір містить 60 000 промаркованих екземплярів, що використовуються для тренування нейронної мережі. «Промарковані»

означає, що для кожного екземпляра вказана відповідна правильна відповідь.

Менший тестовий набір, що включає 10 000 екземплярів, використовується для перевірки правильності роботи ідей або алгоритмів. Він також містить коректні маркери, що дозволяють побачити, чи здатна нейронна мережа дати правильну відповідь.

Використання незалежних один від одного наборів тренувальних та тестових даних гарантує, що з тестовими даними нейронна мережа раніше не стикалася. В іншому випадку можна було б просто запам’ятати тренувальні дані, щоб отримати найкращі результати.

На рис. 3.1 показано частину тестового набору MNIST, завантаженого в текстовий редактор.

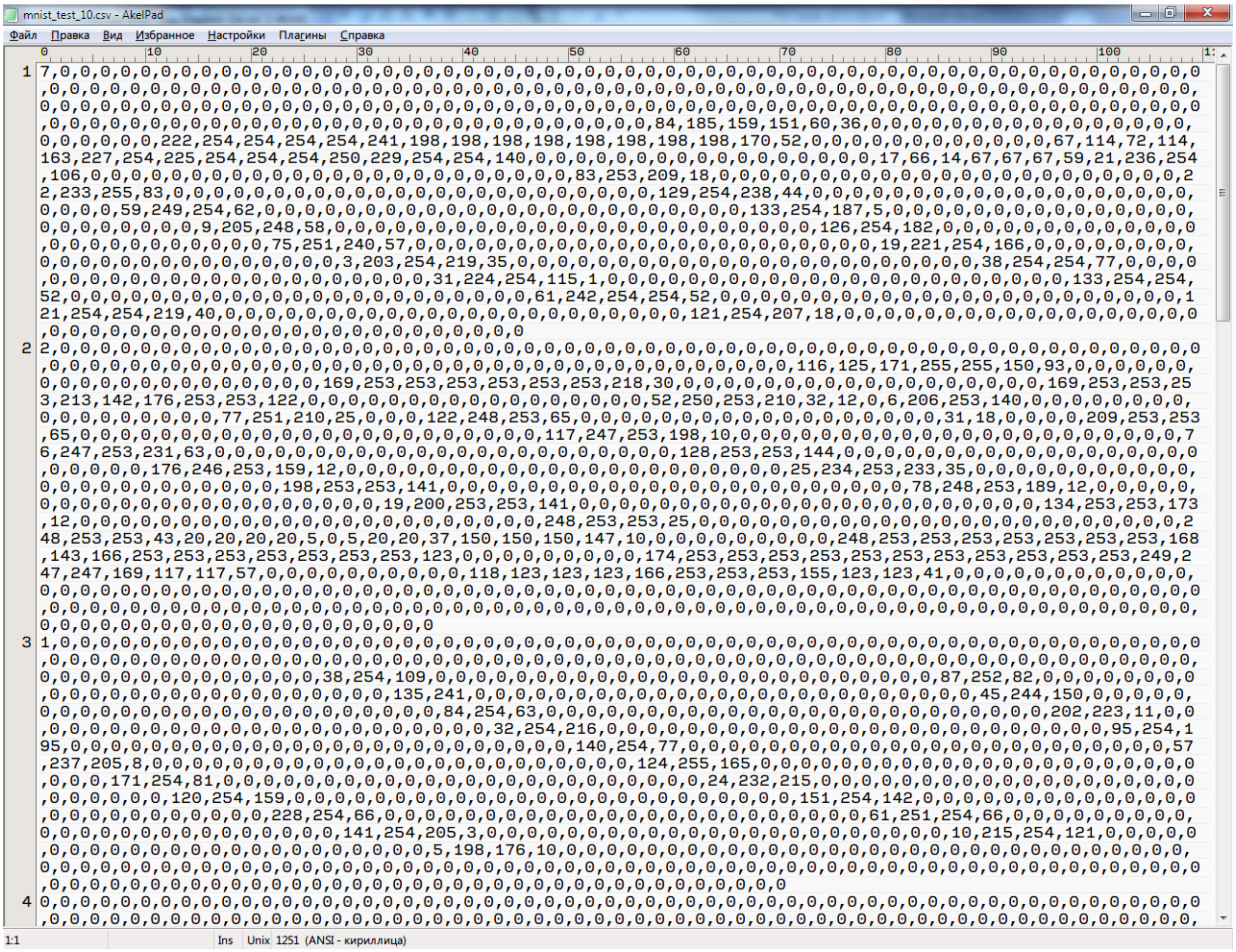

Рис. 3.1 – Фрагмент тестового набору MNIST у форматі CSV

У вікні текстового редактора відображаються рядки тексту, які містять числа, розділені комами. Кожен з цих рядків тексту займає декілька рядків на екрані. Текстовий редактор відображає на полі зліва реальні номери рядків.

Перше значення в рядку тексту – це маркер, тобто, фактична цифра, яку повинен представляти даний рукописний екземпляр. Іншими словами, це відповідь, правильному отриманню якої має навчитися нейронна мережа.

Наступні значення, розділені комами, – це значення пікселів рукописної цифри. Піксельний масив має розмірність 28x28, тому за кожним маркером слідують 784 пікселі. Щоб побачити, яким чином список із 784 значень формує зображення рукописної цифри, необхідно буде вивести цю цифру у графічному вигляді.

Для більш зручного тестування програми було підготовлено невелику підмножину набору даних з бази MNIST також у форматі CSV:

- *mnist_test_10.csv* – 10 записів із тестового набору даних MNIST;
- *mnist_train_100.csv* – 100 записів із тренувального набору даних MNIST.

Коли алгоритм буде відпрацьовано, можна буде повернутися до повного набору даних.

Перш ніж з цими даними можна буде щось зробити, наприклад, побудувати графік або навчити з їх допомогою нейронну мережу, необхідно забезпечити доступ до них з коду мовою Python.

Наступний код відкриває файл зі 100 записами тренувального набору і отримує їх вміст:

```
data_file = open("mnist_dataset/mnist_train_100.csv", 'r')
data_list = data_file.readlines()
data_file.close()
```

Відкриває файл функція *open(),* першим параметром якій передається ім’я файлу. Другий параметр є необов’язковим і повідомляє Python, як треба

працювати з файлом. Літера "*r*" означає, що треба відкрити файл лише для читання, а не для запису. Тим самим ми запобігаємо будь-якій ненавмисній зміні або видаленню файлу.

Функція *open()* створює дескриптор, який відіграє роль посилання на файл, що відкривається, і ми присвоюємо його змінної *data_file*. Коли файл відкритий, будь-які наступні дії з ним, наприклад, читання, здійснюються через цей дескриптор.

Функція *readlines()*, асоційовану з дескриптором файлу *data_file*, використовується для читання всіх рядків вмісту файлу в змінну *data_list*. Ця змінна містить список, кожен елемент якого є рядком, який представляє рядок файлу. Це зручно, оскільки ми можемо переходити до будь-якого рядка файлу так само, як і до конкретних записів у списку. Так, *datalist[0]* – це перший запис, а *data_list[9]* – десятий тощо.

Особливістю функції *readlines()* є те, що вона зчитує весь файл в оперативну пам’ять, тому вона не дуже підходить для великих наборів даних. Однак наші файли невеликі, і використання функції *readlines()* дозволить значно спростити подальший код.

Останній рядок закриває файл, звільняючи системні ресурси після використання.

Виконання наведеного вище коду призводить до того, що список *data_list* заповнюється даними. Щоб побачити візуальне відображення цих даних, скористаємося функцією *imshow()*. Але перед цим перетворимо список чисел в масив:

```
import numpy
import matplotlib.pyplot
%matplotlib inline

all_values = data_list[0].split(',')
image_array = numpy.asfarray(all_values[1:]).reshape((28,28))
matplotlib.pyplot.imshow(image_array, cmap='Greys', interpolation='None')
```

У першому рядку з нижніх трьох перший запис *data_list[0]* розбивається на окремі значення за допомогою функції *split()* з використанням коми

в якості роздільника. Результат розміщується у змінній *all_values*.

В наступному рядку запис списку *all_values[1:]* свідчить про те, що беруться всі елементи списку крім першого. Тим самим ігнорується перше значення, що грає роль маркера, і береться лише решта 784 елементів. Функція *numpy.asfarray()* перетворює текстові рядки на реальні числа і створює масив цих чисел. Останній фрагмент інструкції – *.reshape ((28, 28))* – забезпечує, щоб список був сформований у вигляді квадратної матриці розміром 28x28. Результуючий масив такої ж розмірності отримує назву *image_array*.

Третій рядок просто виводить на екран масив *image_array* за допомогою функції *imshow()*. При цьому використовується колірна палітра відтінків сірого для кращої помітності рукописних символів.

Результат роботи коду показано на рис. 3.2.

```
In [5]: import numpy
        import matplotlib.pyplot
        %matplotlib inline

        all_values = data_list[0].split(',')
        image_array = numpy.asfarray(all_values[1:]).reshape((28,28))
        matplotlib.pyplot.imshow(image_array, cmap='Greys', interpolation='None')

Out[5]: <matplotlib.image.AxesImage at 0x8114748>
```

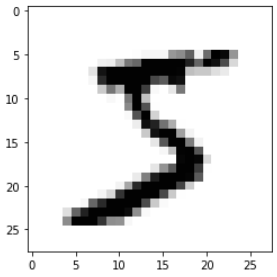

Рис. 3.2 – Відображення першого елементу тестового набору

3.3.2 Підготовка тренувальних даних MNIST

Як зазначалося вище, нейронні мережі працюють краще, якщо вхідні та вихідні дані конфігуруються таким чином, щоб вони залишалися в діапазоні значень, оптимальному для функцій активації вузлів нейронної мережі. Відповідно, перше, що необхідно зробити, – це перевести значення

колірних кодів з діапазону значень 0…255 у діапазон, що охоплює значення від 0,01 до 1,0. Нижня межа діапазону 0,01 дозволить уникнути згаданих раніше проблем з нульовими вхідними значеннями, оскільки вони можуть штучно блокувати оновлення ваги. Щодо верхньої межі допустимого діапазону, – немає потреби уникати значень 1,0 для вхідних сигналів.

Ділення вхідних значень, що змінюються в діапазоні 0…255, на 255 призведе їх до діапазону 0…1,0. Подальше множення цих значень на коефіцієнт 0,99 призведе їх до діапазону 0,0…0,99. щоб вмістити їх в бажаний діапазон 0,01…1,0, треба інкрементувати їх на 0,01. Всі ці дії реалізує наступний код Python:

```
scaled_input = (numpy.asfarray(all_values[1:]) / 255.0 * 0.99) + 0.01
print(scaled_input)
```

Виведення результату підтверджує, що дані дійсно належать до діапазону значень від 0,01 до 1,00 (рис. 3.3).

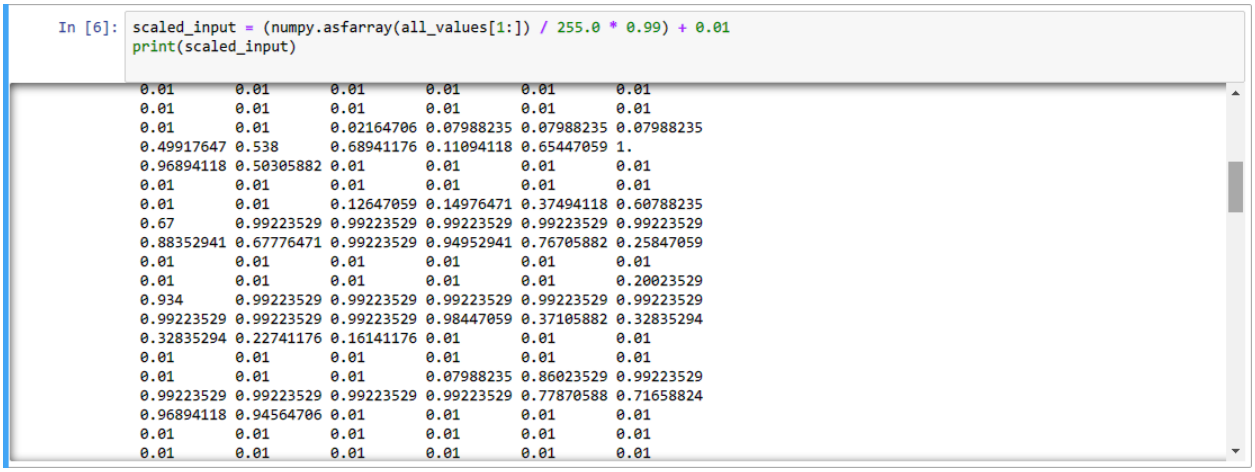

Рис. 3.3 – Приведення значень кольорів пікселів до діапазону 0,01…1

Отже, ми здійснили підготовку даних MNIST шляхом їх масштабування та зсуву і тепер можемо подавати їх на вхід нашої нейронної мережі як для її тренування, так і з метою опитування.

На цьому етапі також потрібно вирішити, як використовувати і як формувати вихідні значення нейронної мережі. Раніше було показано, що

вихідні значення повинні вкладатися в діапазон значень, що забезпечується функцією активації. Логістична функція, яка використовується, охоплює діапазон чисел (0, 1). Таким чином, необхідно масштабувати вихідні значення у процесі тренування мережі.

Підсумковим результатом, який ми взагалі хочемо отримати від нейронної мережі, є маркер, який присвоюється зображенню символу за результатом його класифікації. Таким маркером може бути одне з десяти чисел в діапазоні від 0 до 9. Це означає, що вихідний шар мережі повинен мати 10 вузлів, по одному на кожну можливу відповідь. Якщо відповіддю є «0», то активізуватися повинен перший вузол, тоді як решта вузлів має залишатися пасивними. Якщо відповіддю є «9», то повинен активізуватися останній вузол вихідного шару при пасивних інших вузлах.

Відповідно, треба створити цільовий масив, в якому усі елементи, крім одного, матимуть малі значення, а один елемент – велике значення. Наприклад, для маркера «9» цільовий масив може виглядати так: [0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.99].

Код, що створює цільовий масив:

```
# кількість вихідних вузлів - 10 (приклад)
onodes = 10
targets = numpy.zeros(onodes) + 0.01
targets[int(all_values[0])] = 0.99
```

Другий рядок коду за допомогою функції *numpy.zeros()* створює масив, заповнений нулями. В даному випадку створюється одновимірний масив, розмір *onodes* якого дорівнює кількості вузлів в кінцевому вихідному шарі. Проблема нулів, яка висвітлювалася вище, усувається шляхом додавання 0,01 до кожного елемента масиву.

Наступний рядок вибирає перший елемент запису з набору даних MNIST, що є цільовим маркером тренувального набору, і перетворює його на ціле число. Як тільки перетворення виконано, отриманий цільовий маркер використовується для встановлення значення відповідного елемента

масиву рівним 0,99.

Приклад виконання цього коду для першого елементу тренувального набору («5») показано на рис. 3.4.

```
In [5]: import numpy
        import matplotlib.pyplot
        %matplotlib inline
        data_file = open("mnist_dataset/mnist_train_100.csv", 'r')
        data_list = data_file.readlines()
        data_file.close()
        all_values = data_list[0].split(',')
        image_array = numpy.asfarray(all_values[1:]).reshape((28,28))
        # кількість вихідних вузлів - 10 (приклад)
        onodes = 10
        targets = numpy.zeros(onodes) + 0.01
        targets[int(all_values[0])] = 0.99
        print(targets)

        [0.01 0.01 0.01 0.01 0.01 0.99 0.01 0.01 0.01 0.01]
```

Рис. 3.4 – Приклад формування цільового масиву для маркеру «5»

На даному етапі наш код виглядає так:

```
# Код для створення тришарової нейронної мережі
# і її навчання за допомогою набору даних MNIST
import numpy
# scipy.special для сигмоїдної функції expit()
import scipy.special
# бібліотека для побудови масивів
import matplotlib.pyplot

# визначення класу нейронної мережі
class neuralNetwork:


    # ініціалізація нейронної мережі
    def __init__(self, inputnodes, hiddennodes, outputnodes, learningrate):
        # встановити кількість вузлів у вхідному, прихованому, вихідному шарі
        self.inodes = inputnodes
        self.hnodes = hiddennodes
        self.onodes = outputnodes

        # матриці ваг зв'язків, wih та who
        # ваги зв'язків між вузлами i та j наступного шару - w_i_j
        # w11 w21
        # w12 w22 і т.д.
        self.wih = numpy.random.normal(0.0, pow(self.inodes, -0.5),
           (self.hnodes, self.inodes))
        self.who = numpy.random.normal(0.0, pow(self.hnodes, -0.5),
           (self.onodes, self.hnodes))

        # коеф. навчання
        self.lr = learningrate

        # використання сигмоїди в якості функції активації
        self.activation_function = lambda x: scipy.special.expit(x)

        pass
```

```
    # тренування нейронної мережі
    def train(self, inputs_list, targets_list):
        # конвертація списку вхідних даних у 2d масив
        inputs = numpy.array(inputs_list, ndmin=2).T
        targets = numpy.array(targets_list, ndmin=2).T

        # обчислення вхідних сигналів прихованого шару
        hidden_inputs = numpy.dot(self.wih, inputs)
        # обчислення сигналів, що виходять із прихованого шару
        hidden_outputs = self.activation_function(hidden_inputs)

        # обчислення сигналів, що надходять у кінцевий вихідний рівень
        final_inputs = numpy.dot(self.who, hidden_outputs)
        # обчислення сигналів, що виходять із кінцевого вихідного рівня
        final_outputs = self.activation_function(final_inputs)

        # помилками вихідного рівня є (target - actual)
        output_errors = targets - final_outputs
        # помилки прихованого шару — це вихідні помилки output_errors,
        # роздпоілені пропорційно вагам зв'язків
        # і повторно об'єднані в прихованих вузлах
        hidden_errors = numpy.dot(self.who.T, output_errors)

        # оновити ваги для зв'язків між прихованим та вихідним шарами
        self.who += self.lr * numpy.dot((output_errors * final_outputs * (1.0
- final_outputs)), numpy.transpose(hidden_outputs))

        # оновити ваги для зв'язків між вхідним і прихованим шарами
        self.wih += self.lr * numpy.dot((hidden_errors * hidden_outputs * (1.0
- hidden_outputs)), numpy.transpose(inputs))

        pass


    # опиитування нейронної мережі
    def query(self, inputs_list):
        # перетворення списку вхідних даних у 2d масив
        inputs = numpy.array(inputs_list, ndmin=2).T

        # обчислення вхідних сигналів у прихованому шарі
        hidden_inputs = numpy.dot(self.wih, inputs)
        # обчислення вихідних сигналів для прихованого шару
        hidden_outputs = self.activation_function(hidden_inputs)

        # обчислення вхідних сигналів для вихідного шару
        final_inputs = numpy.dot(self.who, hidden_outputs)
        # обчислення вихідних сигналів для вихідного шару
        final_outputs = self.activation_function(final_inputs)

        return final_outputs

# кількість вхідних, прихованих та вихідних вузлів
input_nodes = 784
hidden_nodes = 200
output_nodes = 10

# коеф. навчання
learning_rate = 0.1
```

```
# створення екземпляру нейронної мережі
n = neuralNetwork(input_nodes,hidden_nodes,output_nodes, learning_rate)

# завантажити у список тестовий набір даних CSV-файлу набору MNIST
training_data_file = open("mnist_dataset/mnist_train.csv", 'r')
training_data_list = training_data_file.readlines()
training_data_file.close()

# тренування нейронної мережі

# перебрати усі записи у тренувальному наборі даних
for record in training_data_list:
    # отримання записів, розділених комою (',')
    all_values = record.split(',')
    # масштабування і зсув вхідних значень
    inputs = (numpy.asfarray(all_values[1:]) / 255.0 * 0.99) + 0.01
    # створення цільових вихідних значень
    # (усі = 0.01, за виключенням бажаного маркерного значення, яке = 0.99)
    targets = numpy.zeros(output_nodes) + 0.01

    # all_values[0] – цільове маркерне значення для даного запису
    targets[int(all_values[0])] = 0.99
    n.train(inputs, targets)
    pass
```

На початку цього коду ми імпортуємо графічну бібліотеку, додаємо код для завдання розмірів вхідного, прихованого та вихідного шарів, зчитуємо малий тренувальний набір даних MNIST, а потім тренуємо нейронну мережу з використанням цих записів.

784 вхідних вузла відповідають кількості пікселів, з яких складається зображення рукописної цифри. Кількість прихованих вузлів (100) вибрана меншою, ніж 784, з тих міркувань, що нейронна мережа повинна знаходити у вхідних значеннях такі особливості або шаблони, які можна виразити у більш короткій формі, ніж ці значення. Тому, задаючи кількість вузлів меншою, ніж кількість вхідних значень, ми змусимо мережу намагатися знаходити ключові особливості шляхом узагальнення інформації. У той же час, якщо вибрати кількість прихованих вузлів занадто малою, буде обмежено можливості мережі щодо визначення достатньої кількості відмітних ознак або шаблонів у зображенні. Тим самим ми позбавили б мережу можливості виносити власні судження щодо даних MNIST. З урахуванням того, що вихідний шар повинен забезпечувати виведення 10

маркерів, а отже, повинен мати десять вузлів, вибір проміжного значення 100 для кількості вузлів прихованого шару представляється збалансованим.

У зв'язку з цим можна зробити важливий висновок: немає ідеального загального методу вибору кількості прихованих вузлів, як і ідеального методу вибору кількості прихованих шарів. Найкращим підходом є проведення експериментів доти, доки не буде отримана конфігурація мережі, оптимальна для завдання, яке вирішується.

3.3.3 Тестування нейронної мережі

Після тренуванням мережі на невеликій підмножині зі ста записів треба перевірити, як вона працює на тестовому наборі даних. Насамперед необхідно отримати тестові записи. Відповідний код схожий на той, який був написаний для отримання тренувальних даних:

```
# завантажити до списку тестовий набір даних CSV-файлу набору MNIST
test_data_file = open("mnist_dataset/mnist_test_10.csv", 'r')
test_data_list = test_data_file.readlines()
test_data_file.close()
```

Розпакуємо ці дані так само, як і попередні, оскільки вони мають аналогічну структуру.

Перш ніж створювати цикл для перебору всіх тестових записів, перевіримо, що станеться, якщо ми вручну виконаємо одиночний тест.

```
# отримати перший тестовий запис
all_values = test_data_list[0].split(,)
# вивести маркер
print(all_values[0])
```

Нижче на рис. 3.5 наведено результати опитування вже навченої нейронної мережі, виконаного з використанням першого запису тестового набору даних. Як видно, в якості маркеру першого запису тестового набору мережа визначила символ «7». Графічне відображення піксельних значень підтверджує, що рукописною цифрою дійсно є цифра «7».

```
In [10]: # завантажити до списку тестовий набір даних CSV-файлу набору MNIST
         test_data_file = open("mnist_dataset/mnist_test_10.csv", 'r')
         test_data_list = test_data_file.readlines()
         test_data_file.close()

In [11]: # отримати перший тестовий запис
         all_values = test_data_list[0].split(',')
         # вивести маркер
         print(all_values[0])

         7

In [12]: image_array = numpy.asfarray(all_values[1:]).reshape((28,28))
         matplotlib.pyplot.imshow(image_array, cmap='Greys', interpolation='None')

Out[12]: <matplotlib.image.AxesImage at 0x59c6148>
```

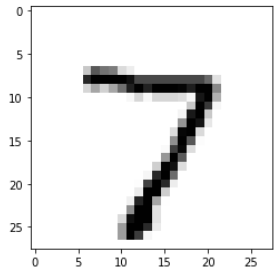

Рис. 3.5 – Результат розпізнавання першого елементу тестового набору даних

В результаті опитування навченої мережі ми отримуємо список чисел, які є вихідними значеннями кожного вихідного вузла:

```
array([[0.12853206],
       [0.1348404 ],
       [0.06462845],
       [0.13178473],
       [0.12749312],
       [0.05451636],
       [0.09853635],
       [0.30977808],
       [0.11925045],
       [0.11447658]])
```

Видно, що одне з вихідних значень, а саме 0,30977808, перевищує інші, і цьому значенню відповідає маркер «7». Це восьмий елемент списку, оскільки першому елементу відповідає маркер «0».

Таким чином, ми вже навчили нашу нейронну мережу так, що вона спромоглася визначити цифру, надану їй у вигляді зображення. Важливо, що мережа раніше не стикалася з цим зображенням, оскільки воно не входило в тренувальний набір даних. Отже, нейронна мережа виявилася спроможною коректно класифікувати незнайомий їй цифровий символ.

Варто також відзначити, що цей результат було отримано при тому, що для навчання мережі була використана лише невелика частина повного набору тренувальних даних. Повний набір включає 60 тисяч записів, а ми поки що використали лише 100 з них.

Наступний код дозволяє перевірити, наскільки добре нейронна мережа справляється з рештою набору даних.

```
# тестування нейронної мережі
# журнал оцінок роботи мережі, спочатку порожній
scorecard = []
# перебрати всі записи в тестовому наборі даних
for record in test_data_list:
    # отримати список значень із запису, розділених комами (',')
    all_values = record.split(',')
    # правильна відповідь - перше значення
    correct_label = int(all_values[0])
    print(correct_label, "істинний маркер")
    # масштабувати та зсунути вхідні значення
    inputs = (numpy.asfarray(all_values[1:]) / 255.0 * 0.99) + 0.01
    # опитування мережі
    outputs = n.query(inputs)
    # індекс найбільшого значення є маркерним значенням
    label = numpy.argmax(outputs)
    print(label, "відповідь мережі")
    # приєднати оцінку відповіді мережі до кінця списку
    if (label == correct_label):
        # у разі правильної відповіді мережі приєднати до списку значення 1
        scorecard.append(1)
    else:
        # у разі неправильної відповіді мережі приєднати до списку значення 0
        scorecard.append(0)
    pass
pass
```

Цей код дає такий результат:

```
7 істинний маркер
7 відповідь мережі
2 істинний маркер
0 відповідь мережі
1 істинний маркер
1 відповідь мережі
0 істинний маркер
0 відповідь мережі
4 істинний маркер
4 відповідь мережі
1 істинний маркер
1 відповідь мережі
4 істинний маркер
7 відповідь мережі
9 істинний маркер
1 відповідь мережі
```

```
5 істинний маркер
0 відповідь мережі
9 істинний маркер
7 відповідь мережі
```

Як бачимо, є декілька розбіжностей.

Команда

```
print(scorecard)
```

виводить такий рядок результатів:

```
[1, 0, 1, 1, 1, 1, 0, 0, 0, 0]
```

Як видно, з десяти тестових записів правильно було розпізнано лише 5, тобто частка правильних результатів становила 50 %.

Доповнимо код фрагментом, який виводитиме відносну частку правильних відповідей у вигляді дробу:

```
# розрахувати показник ефективності у вигляді частки правильних відповідей
scorecard_array = numpy.asarray(scorecard)
print ("ефективність = ", scorecard_array.sum() / scorecard_array.size)
```

Ця частка розраховується як кількість всіх записів у журналі, що містять «1», поділене на загальну кількість записів (розмір журналу). Результат в даному випадку:

```
ефективність =  0.5
```

3.3.4 Тренування та тестування нейронної мережі з використанням повної бази даних

Далі проведемо тренування і тестування ефективності мережі з використанням повної бази даних. Для цього замінимо імена файлів, оскільки тепер вони повинні вказувати на повний набір тренувальних даних, що налічує 60 тисяч записів, і набір тестових даних, що налічує 10 тисяч записів. Раніше ці набори були збережені у файлах з іменами, відповідно, *mnist_dataset/mnist_train.csv* та *mnist_dataset/mnist_test.csv*.

Виконання модифікованої програми тренування зайняло близько 1 хвилини. Результат тестування ефективності:

```
ефективність =  0.9518
```

Якщо порівняти цей показник з аналогічними результатами еталонних тестів, доступними за адресою http://yann.lecun.com/exdb/mnist/, можна побачити, що в деяких випадках отриманий нами результат навіть кращий за еталонні.

Таким чином, перша спроба вже продемонструвала ефективність на рівні нейронної мережі професійних дослідників.

3.3.5 Поліпшення результатів шляхом налаштування коефіцієнта навчання

Отримання показника ефективності 95 % при тестуванні набору даних MNIST нашою нейронною мережею, – зовсім непоганий результат. Однак ми спробуємо покращити його.

Насамперед, ми налаштуємо коефіцієнт навчання α. Перед цим ми задали його значення $\alpha = 0{,}3$, не тестуючи інші значення. Спробуємо подвоїти це значення до величини 0,6 і подивимося, як це позначиться на можливості нейронної мережі вчитися.

Тестування показало, що виконання коду із таким значенням коефіцієнта навчання дає показник ефективності 0,9174. Цей результат гірший від попереднього. Очевидно, в даному випадку збільшення коефіцієнта навчання порушує монотонність процесу мінімізації помилок методом градієнтного спуску та супроводжується перескоками через мінімум.

Повторимо обчислення, встановивши коефіцієнт навчання $\alpha = 0{,}1$.

Цього разу показник ефективності покращився до 0,9532, що майже збігається з результатом еталонного тесту, який проводився з використан-

ням тисячі прихованих вузлів. Наш результат був отриманий з використанням набагато меншої кількості вузлів.

Спробуємо зменшити коефіцієнт навчання до 0,01.

Тестування показало, що це також призвело до зменшення показника ефективності мережі – до 0,9238. Очевидно, занадто малі значення коефіцієнта навчання знижують ефективність. Це є логічним, оскільки малі кроки зменшують швидкість градієнтного спуску.

Описані результати представлені на рис. 3.6 у вигляді графіка. Взагалі, для більшої достовірності слід було б провести такі експерименти багато разів, щоб виключити фактор випадковості та вплив невдалого вибору маршрутів градієнтного спуску. Але навіть у такому спрощеному вигляді проведений експеримент ілюструє загальну ідею про існування певного оптимального значення коефіцієнта навчання.

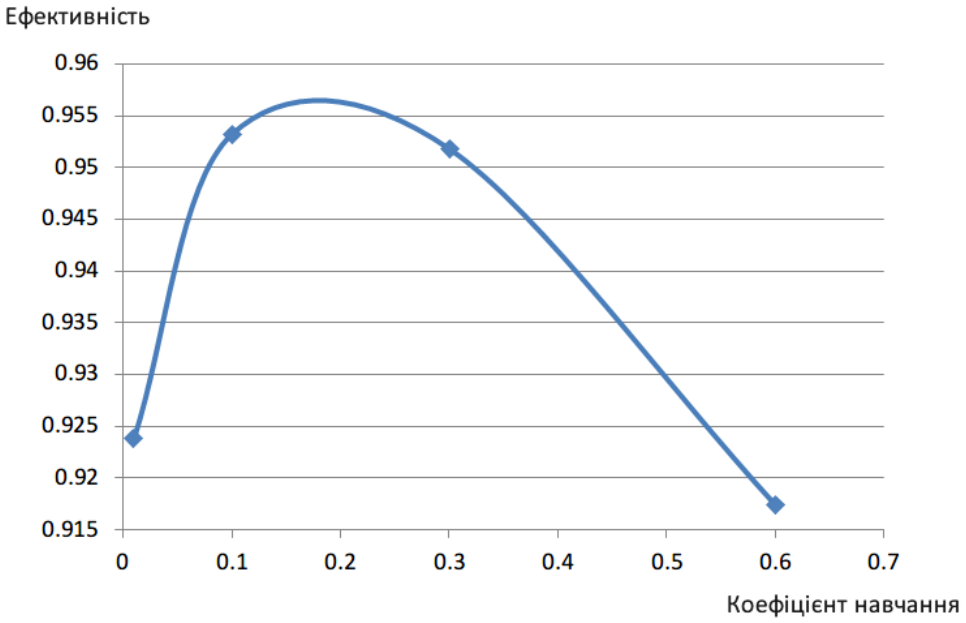

Рис. 3.6 – Залежність ефективності розпізнавання від значень коефіцієнту навчання

Вигляд графіка показує, що оптимальним може бути значення в інтервалі α від 0,1 до 0,3, тому випробуємо значення α = 0,2.

При даному значенні показник ефективності вийшов рівним 0,9558. Як видно, цей результат трохи кращий за ті, які був отриманий при

значеннях коефіцієнта навчання 0,1 та 0,3. Отже, зупинимося на значенні коефіцієнта навчання $\alpha = 0{,}2$, яке виявило себе як оптимальне серед протестованих для набору даних MNIST та нашої нейронної мережі.

3.3.6 Поліпшення результатів шляхом повторення тренувальних циклів

Для наступного удосконалення результатів здійснимо багаторазове повторення циклів тренування з тим самим набором даних. Для одного тренувального циклу використовується термін «епоха». Сеанс тренування із десяти епох означає десятикратний прогін всього тренувального набору даних. Завдяки цьому повторенню забезпечується більше маршрутів градієнтного спуску, які оптимізують вагові коефіцієнти.

Подивимося спочатку, що нам дадуть дві тренувальні епохи. Для цього треба змінити код, передбачивши додатковий цикл виконання коду тренування:

```
# тренування нейронної мережі

# змінна epochs вказує, скільки разів тренувальний набір даних
# використовується для тренування мережі
epochs = 2

for e in range(epochs):
    # перебрати всі записи в тренувальному наборі даних
    .
    .
    .
    pass
```

Результуючий показник ефективності для двох епох склав 0,9576, що дещо краще за показник для однієї епохи.

Аналогічно тому, як ми налаштовували коефіцієнт навчання, проведемо експеримент із використанням різної кількості епох і побудуємо графік залежності показника ефективності від цього фактора. Інтуїція підказує, що чим більше тренувань, тим вищою буде ефективність. Але можна також припустити, що занадто велика кількість тренувань може призвести і до

погіршення ефективності через так зване перенавчання мережі на тренувальних даних, що знижує ефективність при роботі з незнайомими даними.

У даному разі отримані результати, які показані на рис. 3.7.

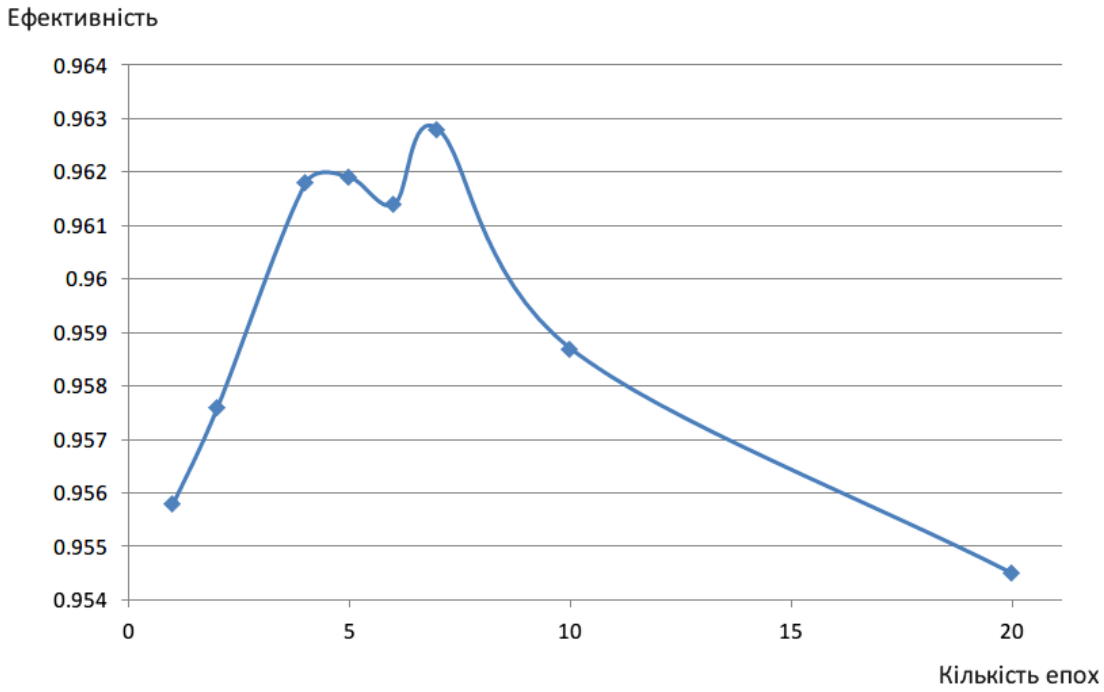

Рис. 3.7 – Залежність ефективності роботи нейронної мережі від кількості епох

Тепер пікове значення ефективності становить 0,9628, або 96,28 % при семи епохах. Як бачимо, результати виявилися не дуже передбачуваними. Оптимальна кількість епох при поточних вхідних даних – 5 або 7. При великих значеннях ефективність падає, що, скоріше за все, є наслідком перенавчання. Провал при 6 епохах, ймовірно, обумовлений невдалими параметрами циклу, які привели градієнтний спуск до помилкового мінімуму. За своєю суттю навчання нейронної мережі – це випадковий процес, який іноді може давати не дуже вдалі результати. Насправді, можна було б очікувати на дещо інші результати, якщо б ми провели декілька експериментів для кожної точки даних, щоб зменшити варіацію, викликану випадковими чинниками. Інше можливе пояснення отриманого результату –

це те, що коефіцієнт навчання виявився занадто великим для більшої кількості епох.

Повторимо експеримент, зменшивши коефіцієнт навчання α з 0,2 до 0,1. На наступному графіку (рис. 3.8) нові значення ефективності при $\alpha = 0,1$ представлені разом із попередніми результатами, щоб їх було зручно порівняти між собою.

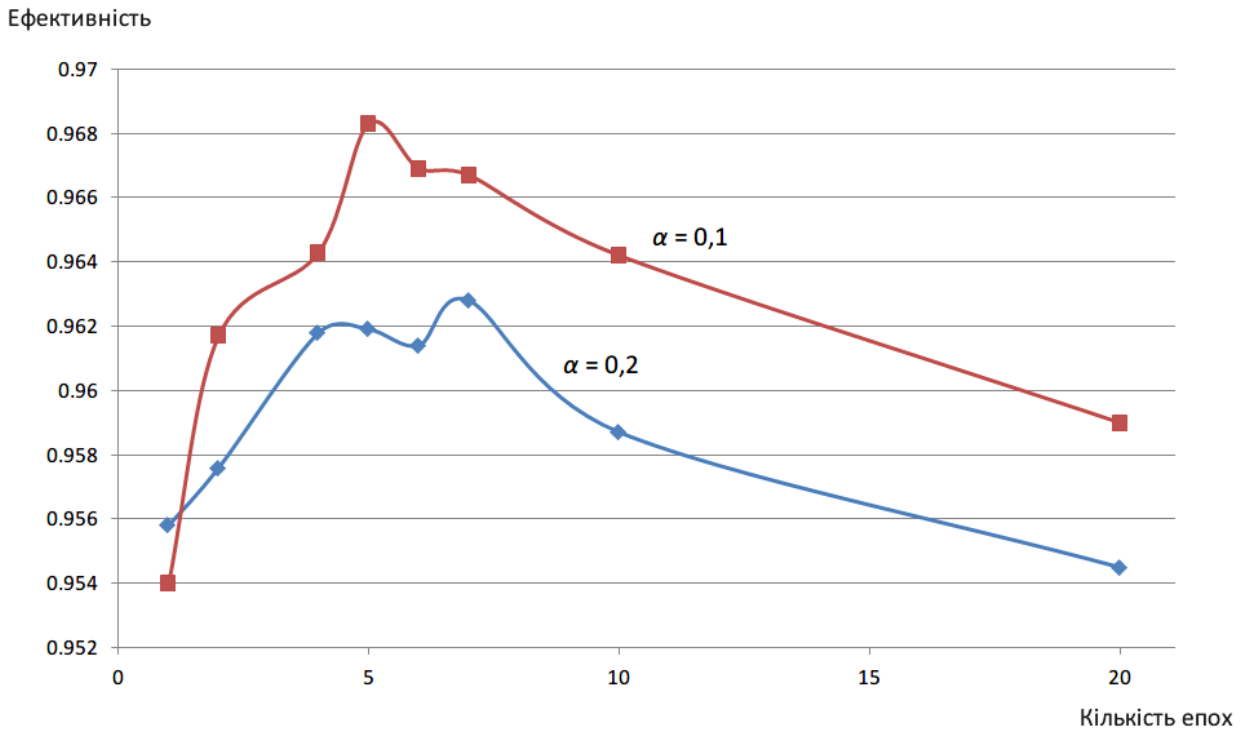

Рис. 3.8 – Залежність ефективності роботи нейронної мережі від кількості епох при різних значеннях коефіцієнту навчання α

Як видно з рис. 3.8, зменшення коефіцієнта навчання призвело до поліпшення ефективності при більшій кількості епох. Піковому значенню 0,9683 відповідає ймовірність помилок приблизно у 3 %, що можна порівняти з еталонними результатами, вказаними на сайті Яна Лекуна за адресою http://yann.lecun.com/exdb/mnist/.

Можна припустити, що якщо ми плануємо використовувати метод градієнтного спуску за значно більшої кількості епох, то зменшення коефіцієнта навчання, який забезпечує коротші кроки, загалом призведе

до вибору кращих маршрутів мінімізації помилок. Ймовірно, 5 епох – це оптимальна кількість для тестування нашої нейронної мережі з набором даних MNIST.

3.3.7 Зміна конфігурації мережі

Ще один із факторів, вплив яких варто дослідити – це конфігурація нейронної мережі. Спробуємо змінити кількість вузлів прихованого проміжного шару. Раніше ми встановили його рівним 100.

Прихований шар є тією частиною мережі, в якій саме відбувається навчання. Вузли вхідного шару приймають вхідні сигнали, а вихідні вузли видають відповідь мережі. Власне, прихований шар (або шари) має навчити мережу перетворювати вхідні сигнали у відповідь. Саме у ньому й відбувається навчання, яке ґрунтується на значеннях вагових коефіцієнтів до та після прихованого шару.

Можна очікувати, що при надто малій кількості прихованих вузлів (наприклад, 3) неможливо буде навчити мережу будь-чому, оскільки не можна вмістити всю вхідну інформацію в цих вузлах. Мережа в цьому випадку матиме обмежену здатністю до навчання.

Якщо, навпаки, створити 10 тисяч прихованих вузлів, то можна припустити, що здатності до навчання цілком вистачить для будь-яких задач. Але при цьому ми, скоріш за все, зіткнемося з труднощами навчання мережі, оскільки кількість маршрутів навчання буде непомірно великою. Не виключено, що для тренування такої мережі знадобилося б 10 тисяч епох.

Проведені експерименти щодо дослідження впливу розміру прихованого шару дозволили отримати графік, показаний на рис. 3.9. Як можна бачити, при невеликій кількості вузлів результати гірші, ніж при великих кількостях. Це збігається із зробленими раніше припущеннями. Однак примітно те, що навіть для лише п’яти прихованих вузлів показник ефективності склав 0,7. Тобто за такої малої кількості вузлів, у яких

відбувається власне навчання мережі, вона все одно дає 70 % правильних відповідей. До цього виконувалися тести з використанням ста прихованих вузлів. Зараз ми бачимо, що навіть десять прихованих вузлів забезпечують точність на рівні 0,9 (90 %).

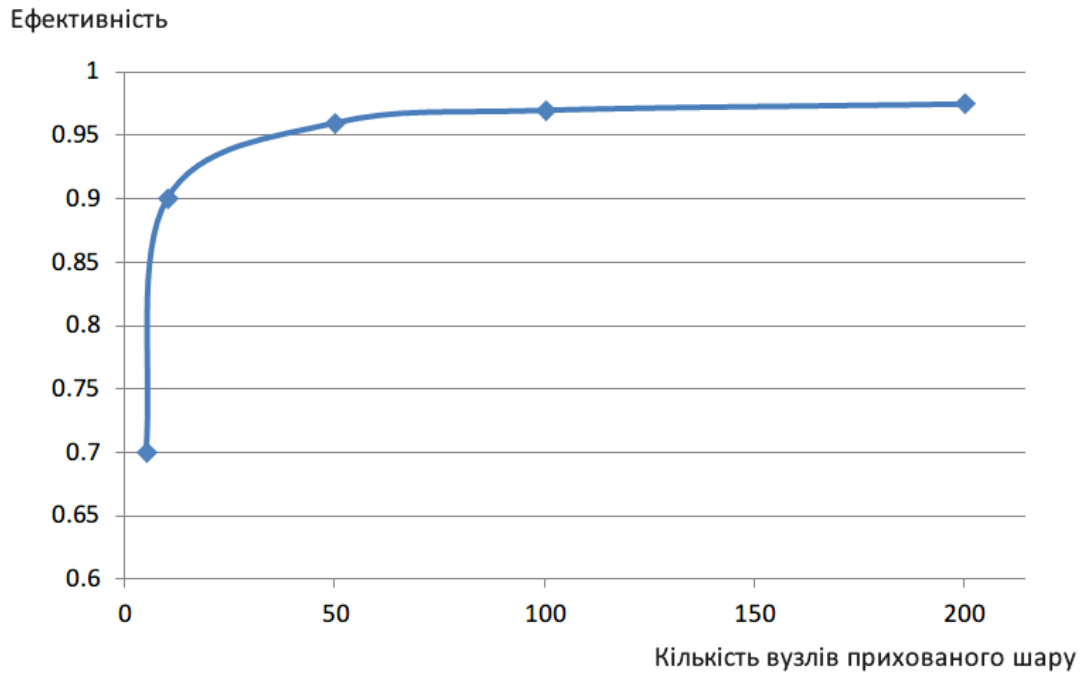

Рис. 3.9 – Залежність ефективності роботи нейронної мережі від кількості вузлів прихованого шару

Таким чином, можна зробити висновок, що нейронна мережа здатна давати хороші результати навіть при невеликій кількості прихованих вузлів, у яких відбувається навчання, що свідчить про її досить потужні можливості. У міру подальшого збільшення кількості прихованих вузлів результати продовжують покращуватись, проте вже не так різко. При цьому значно зростає час, що витрачається на тренування мережі, оскільки додавання кожного додаткового прихованого вузла призводить до збільшення кількості зв’язків вузлів прихованого шару з вузлами попереднього та наступного шарів, а разом з цим і обсягу обчислень. Тому очевидно, що необхідно досягати компромісу між кількістю прихованих вузлів та

допустимими витратами часу для конкретної конфігурації апаратної платформи моделювання нейронної мережі.

В результаті даного експерименту була також отримана найбільша точність роботи мережі: 0,97 при 200 прихованих вузлах. Це цілком на рівні результатів еталонних тестів, опублікованих на сайті Лекуна. Таким чином, межа точності 0,968, отримана раніше, була подолана за рахунок саме зміни конфігурації мережі.

3.4 Використання власної бази рукописних цифр

3.4.1 Формування зображень для розпізнавання

В даній частині роботи описано процес створення тестової бази даних з власноруч написаними цифрами (рис. 3.10). При цьому сирцевий матеріал не очищувався ретельно від різних видів перешкод, таких як шум, пошкодження, щоб не спрощувати максимально нейронній мережі роботу і подивитися, наскільки добре вона з ними впорається.

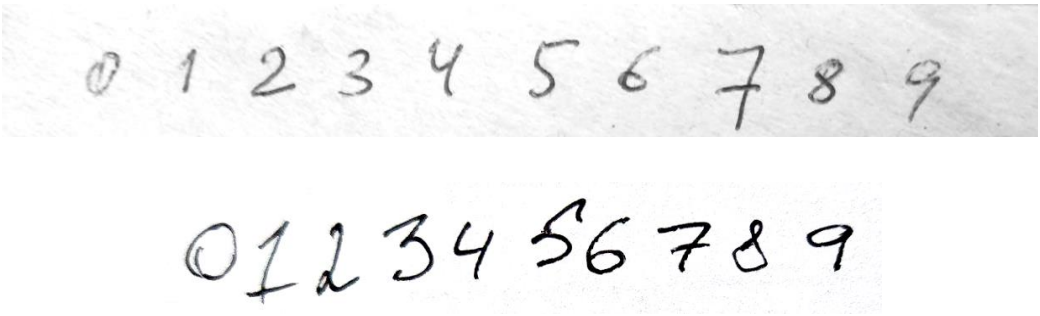

Рис. 3.10 – Два варіанти вхідного зображення з рукописними цифрами

Здатність людського мозку зберігати функціональність навіть після отримання ушкоджень дозволяє зробити припущення, що в нейронній мережі набуті знання розподіляються між декількома зв’язками, а тому в умовах пошкодження деяких зв’язків інші також можуть успішно

функціонувати. Це означає, крім того, що вони можуть функціонувати і у разі пошкодження або неповноти вхідного зображення.

Написані на листку цифри були зфотографовані і передані в цифровому вигляді в комп'ютер. Для подальшої обробки зображень використовувався безкоштовний графічний редактор Photofiltre.

Перш за все, були створені зменшені версії PNG-зображень шляхом масштабування їх до розміру 28x28 пікселів, щоб привести у відповідність до даних MNIST, що застосовувалися раніше. Файли окремих зображень були названі *2828_my_own_?.png* і поміщені в папку *my_own_images*.

3.4.2 Програмний доступ до зображень і їх розпізнавання

Для читання та декодування даних із поширених форматів зображень, включаючи PNG, існує спеціальна бібліотека Python – *imageio*. Наступний код Python показує приклад підготовки зображення для передачі нейронній мережі:

```
import imageio
img_array = imageio.imread(image_file_name, as_gray=True)
img_data = 255.0 - img_array.reshape(784)
img_data = (img_data / 255.0 * 0.99) + 0.01
```

Функція *imageio.imread()* забезпечує отримання даних з файлів зображень, таких як PNG або JPG. Параметр *as_gray=True* перетворює переводить колірні коди зображення у шкалу відтінків сірого, що нам і треба. Наступний рядок перетворює квадратний масив розмірністю 28x28 в довгий список значень, який потрібен для передачі даних нейронній мережі, як це робилося раніше. Новим тут є віднімання значень масиву з 255,0. Зазвичай коду 0 відповідає чорний колір, а коду 255 – білий. Але в наборі даних MNIST використовується зворотна схема, у зв'язку з чим необхідно інвертувати кольори для приведення їх у відповідність до угод, прийнятих у MNIST. Останній рядок виконує масштабування даних, переводячи їх у діапазон значень від 0,01 до 1,0.

Написану раніше програма потребує модифікації для тестування її з використанням набору даних, створеного на основі наших зображень.

Імпорт бібліотек:

```
# бібліотека для завантаження даних із файлів зображень PNG
import imageio
# glob допомагає вибрати кілька файлів за допомогою шаблонів
import glob
import scipy.misc
import numpy
# бібліотека для графічного відображення масивів
import matplotlib.pyplot
%matplotlib inline
```

Клас нейронної мережі залишається тим самим.

Задаємо параметри нейронної мережі і створюємо її:

```
# кількість вузлів у вхідному, прихованому і вихідному шарах
input_nodes = 784
hidden_nodes = 100
output_nodes = 10

# коефіцієнт навчання
learning_rate = 0.1

# створення об'єкту нейронної мережі
n = neuralNetwork(input_nodes,hidden_nodes,output_nodes, learning_rate)
```

Завантажуємо тренувальні дані з бази MNIST:

```
training_data_file = open("mnist_dataset/mnist_train.csv", 'r')
training_data_list = training_data_file.readlines()
training_data_file.close()
```

Тренуємо нейронну мережу на наборі даних MNIST циклом з 10 епох:

```
# кількість циклів тренування
epochs = 10

for e in range(epochs):
    # переглянути всі записи в наборі навчальних даних
    for record in training_data_list:
        # розділити записи по комах ','
        all_values = record.split(',')
        # scale and shift the inputs
        inputs = (numpy.asfarray(all_values[1:]) / 255.0 * 0.99) + 0.01
        # create the target output values (all 0.01, except the desired label
which is 0.99)
        targets = numpy.zeros(output_nodes) + 0.01
        # all_values[0] is the target label for this record
        targets[int(all_values[0])] = 0.99
```

```
        n.train(inputs, targets)
        pass
    pass
```

Тестуємо нейронну мережу на власному наборі даних:

```
# тестування на власному наборі зображень
our_own_dataset = []

# завантажити дані зображення png як тестовий набір даних
for image_file_name in glob.glob('my_own_images/2828_my_own_?.png'):

    # використання назви файлу для встановлення правильної мітки
    label = int(image_file_name[-5:-4])

    # завантаження даних зображення з файлів png в масив
    print ("завантаження ... ", image_file_name)
    img_array = imageio.imread(image_file_name, as_gray=True)

    # переформування з масиву 28x28 у список з 784 інвертованих значень
    img_data  = 255.0 - img_array.reshape(784)

    # масштабування даних зображення в діапазон від 0.01 до 1.0
    img_data = (img_data / 255.0 * 0.99) + 0.01
    print(" мінімальний масштабований елемент  = ", numpy.min(img_data))
    print(" максимальний масштабований елемент = ", numpy.max(img_data))

    # додавання даних мітки та зображення для тестування набору даних
    record = numpy.append(label,img_data)
    our_own_dataset.append(record)

    pass
```

Після виконання програми результат роботи тестувальної команди

```
завантаження ...  my_own_images\2828_my_own_0.png
 мінімальний масштабований елемент  =  0.01
 максимальний масштабований елемент =  0.8136471
завантаження ...  my_own_images\2828_my_own_1.png
 мінімальний масштабований елемент  =  0.01
 максимальний масштабований елемент =  0.658353
завантаження ...  my_own_images\2828_my_own_2.png
 мінімальний масштабований елемент  =  0.01
 максимальний масштабований елемент =  0.73988235
завантаження ...  my_own_images\2828_my_own_3.png
 мінімальний масштабований елемент  =  0.01
 максимальний масштабований елемент =  0.77482355
завантаження ...  my_own_images\2828_my_own_4.png
 мінімальний масштабований елемент  =  0.01
 максимальний масштабований елемент =  0.6932941
завантаження ...  my_own_images\2828_my_own_5.png
 мінімальний масштабований елемент  =  0.01
 максимальний масштабований елемент =  0.7515294
завантаження ...  my_own_images\2828_my_own_6.png
 мінімальний масштабований елемент  =  0.01
 максимальний масштабований елемент =  0.80588233
завантаження ...  my_own_images\2828_my_own_7.png
```

```
 мінімальний масштабований елемент  =  0.01
 максимальний масштабований елемент =  0.77482355
завантаження ...  my_own_images\2828_my_own_8.png
 мінімальний масштабований елемент  =  0.06047059
 максимальний масштабований елемент =  0.802
завантаження ...  my_own_images\2828_my_own_9.png
 мінімальний масштабований елемент  =  0.08376471
 максимальний масштабований елемент =  0.736
```

Нарешті, тестування нейронної мережі на основі власних зображень. Код для розпізнавання першого елементу – цифри «0»:

```
# запис для тестування
item = 0

# малювання зображення
matplotlib.pyplot.imshow(our_own_dataset[item][1:].reshape(28,28),
cmap='Greys', interpolation='None')

# правильна відповідь - перше значення
correct_label = our_own_dataset[item][0]
# дані - у наступних значеннях
inputs = our_own_dataset[item][1:]

# опитування мережі
outputs = n.query(inputs)
print (outputs)

# індекс найбільшого значення відповідає мітці
label = numpy.argmax(outputs)
print("відповідь мережі ", label)
# додати результат по співпадінню до списку
if (label == correct_label):
    print ("співпадає")
else:
    print ("не співпадає")
    pass
```

Аналогічні перевірки були проведені для всіх інших цифр.

Результат тестування на перших чотирьох зображеннях показаний на рис. 3.11. Як видно, нейронною мережею невірно розпізнався нуль внаслідок його складної форми – штрих всередині збив мережу з пантелику, так що вона дала відповідь «8».

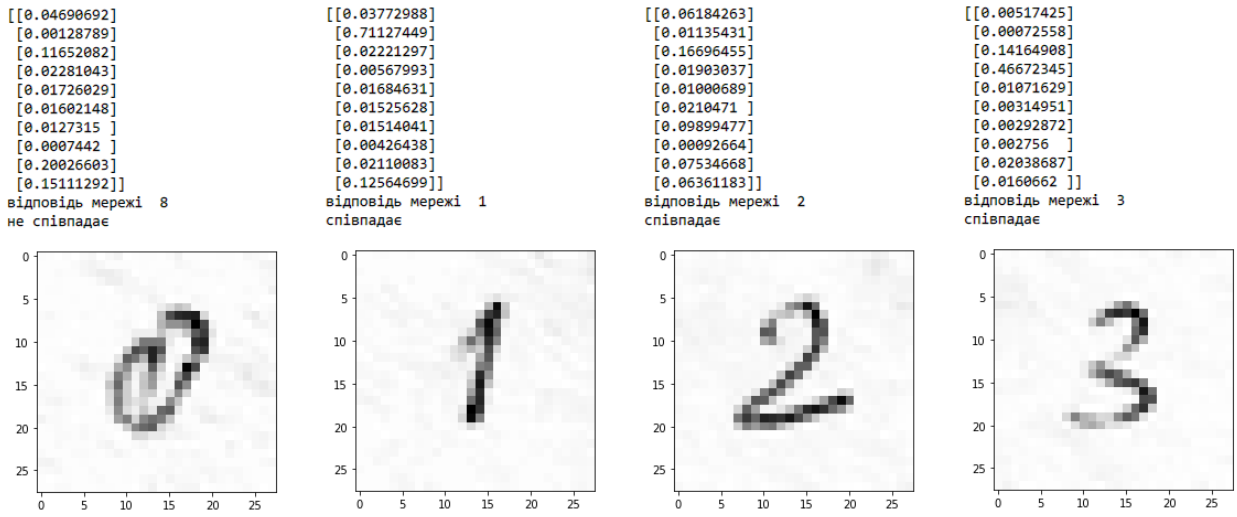

Рис. 3.11 – Результат тестування натренованої мережі
на першому варіанті рукописних цифр «0», «1», «2» та «3»

На рис. 3.12 показані результати розпізнавання символів «4», «5», «6» та «7». На цей раз мережа зробила помилку на цифрі «7». Очевидна причина в тому, що база MNIST створювалася на рукописних символах здебільшого американців, а в них не прийнято ставити горизонтальну риску у сімки.

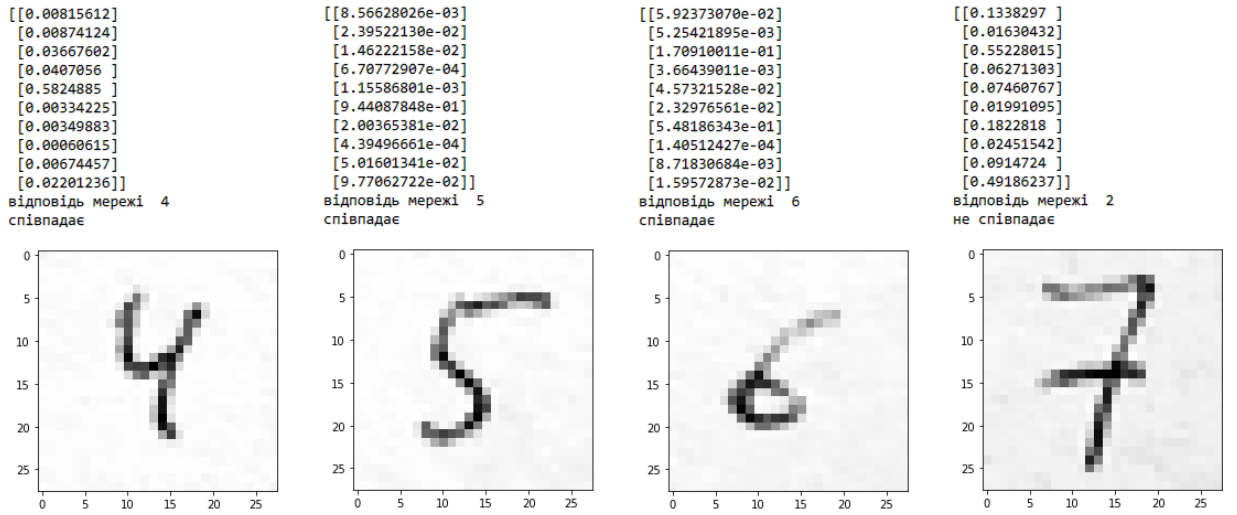

Рис. 3.12 – Результат тестування натренованої мережі
на першому варіанті рукописних цифр «4», «5», «6» та «7»

На рис. 3.13 показані результати розпізнавання символів «8» та «9». На цей раз мережа зробила помилку на цифрі «9», розпізнавши її як 8. Тут причина на перший погляд незрозуміла, бо на вигляд ця дев'ятка не здається схожою на 8. Але подальші експерименти виявили, що розпізнаванню заважає темний фон.

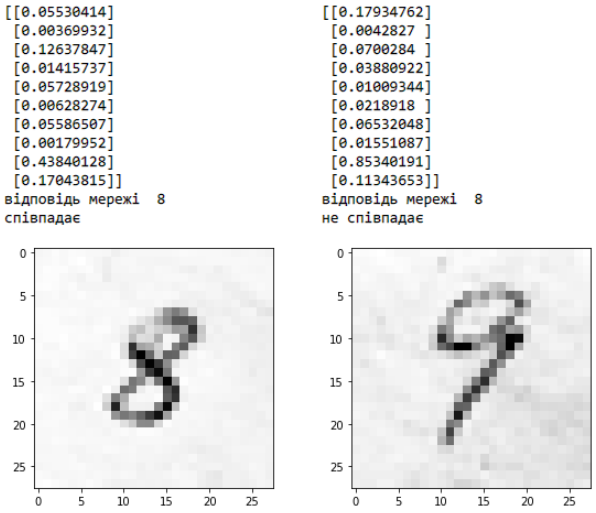

Рис. 3.13 – Результат тестування натренованої мережі
на першому варіанті рукописних цифр «8» та «9»

Загальний відсоток правильно розпізнаних першого варіанту власних цифр склав 7 / 10, тобто 70 %. Це не дуже багато, але навіть такі результати в деяких задачах є цілком прийнятними.

Після розпізнавання першого варіанту цифр експеримент було повторено для другого варіанту з однією зміною у структурі мережі – кількість прихованих вузлів була збільшена до 200. Це призвело до збільшення часу на навчання мережі приблизно утричі (33 хвилини).

Крім того, було піднято контраст зображення, щоб позбутися сірого фону, інакше усі цифри нейромережа розпізнавала як «8». Результат показаний на рис. 3.14, 3.15 та 3.16.

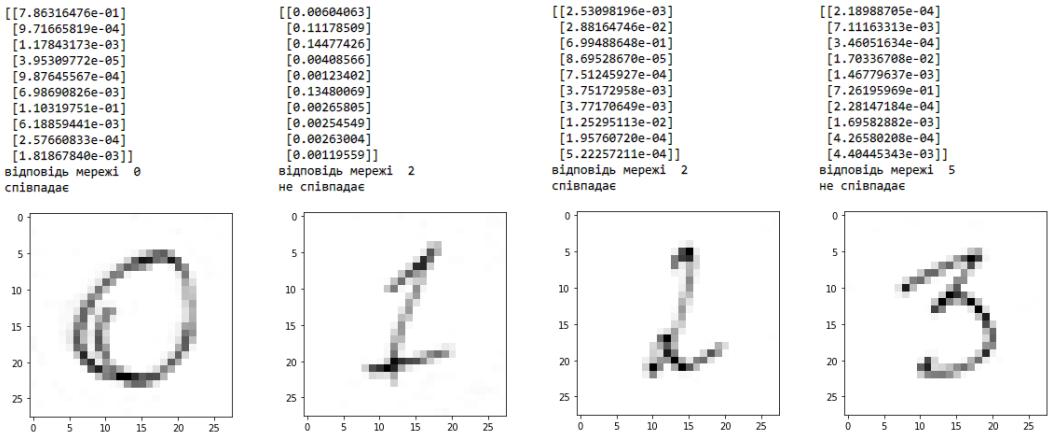

Рис. 3.14 – Результат розпізнавання другого варіанту рукописних цифр «0», «1», «2» та «3»

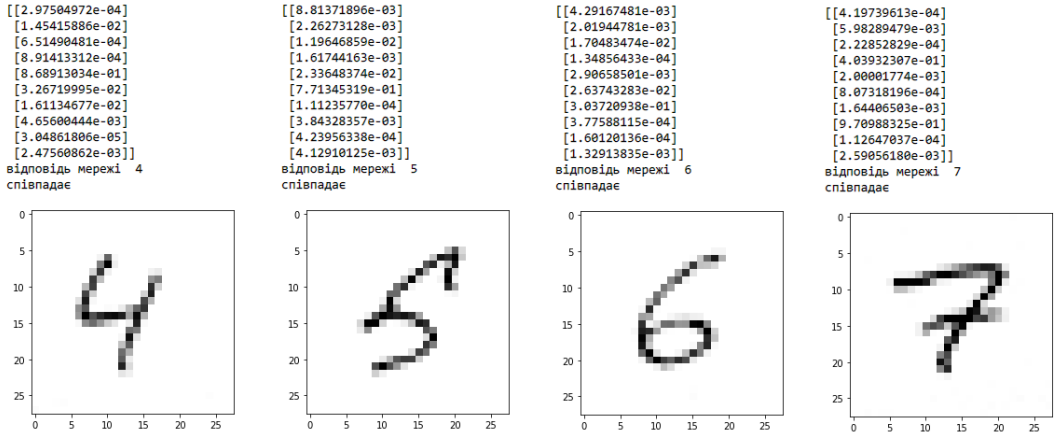

Рис. 3.15 – Результат розпізнавання другого варіанту рукописних цифр «4», «5», «6» та «7»

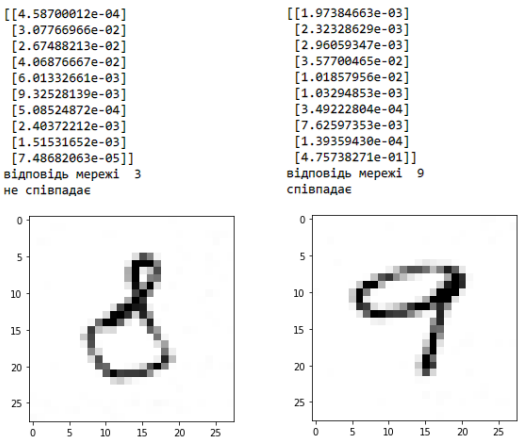

Рис. 3.16 – Результат розпізнавання другого варіанту рукописних цифр «8» та «9»

Як видно, у другому варіанті написання цифр нейронна мережа знов зробила 3 помилки, але на інших цифрах: «1» (очевидно, зіграла роль нижня зарубка), «3» і «8». Тобто, результат достовірності розпізнавання знову склав 70 %.

Для покращення результатів можна спробувати застосувати такі підходи:

- взяти більшу тренувальну базу, причому зібрану з текстів, написаних європейцями, в яких набагато більш розповсюджена практика ставити поперечну риску у рукописної цифри «7»;
- покращити якість вхідних зображень: краще очищувати їх від шуму та інших артефактів;
- продовжити експерименти з коефіцієнтом навчання, кількістю вузлів проміжного шару та кількістю епох навчання, щоб підібрати більш підходящі їх значення для даного випадку.

Висновки до третього розділу

В результаті виконання даної частини кваліфікаційної роботи було отримано такі результати:

1. Обрано засіб реалізації нейронної мережі в рамках підходу, прийнятого в першому розділі.

2. Виконано програмну реалізацію нейронної мережі у вигляді класу на мові програмування Python, включаючи методи ініціалізації мережі заданими параметрами, тренування і опитування.

3. Здійснено навчання нейронної мережі розпізнаванню рукописним цифрам за базі даних MNIST.

4. Виконано дослідження щодо шляхів поліпшення результатів розпізнавання і підбору оптимальних параметрів мережі та її тренування.

5. Підготовлено два набору з власноруч написаних цифр і протестовано навчену мережу стосовно можливості їх розпізнавання.

В результаті було отримано деякі важливі загальні результати щодо методів побудови і тренування нейронних мереж, а також покращення результатів їх роботи. Виявлено основні параметри, які можна регулювати для досягнення цієї мети, а саме:

- коефіцієнт навчання;
- кількість вузлів проміжного шару нейронної мережі;
- кількість епох навчання.

Отримані результати дозволять проводити в майбутньому подальші дослідження стосовно використання нейронної мережі на інших наборах даних та, взагалі, для вирішення інших задач.

РОЗДІЛ 4
ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ ПРОЕКТУ

4.1 Визначення етапів і розрахунок трудомісткості програмного продукту

У даній кваліфікаційній роботі проведено дослідження процесу побудови і навчання штучної нейронної мережі для вирішення задачі розпізнавання рукописного тексту.

Програмний продукт, що розробляється в даному проекті, відноситься до групи В – розробка, що має аналоги, 2 група складності – продукт, що реалізує обчислювальні алгоритми, група наведення вхідної та вихідної інформації – 11 та 21. Функціональне призначення продукту – розпізнавання графічних символів.

Етап розробки технічного завдання. На цьому етапі здійснюється розробка та узгодження технічного завдання, визначаються основні вимоги до програмного продукту, терміни роботи, кількість та склад необхідних вихідних даних. Робота проводиться спільно розроблювачем постановки задач та розроблювачем програмного забезпечення.

Трудомісткість розробки технічного завдання $T_{тз}$ розраховується за формулою:

$$T_{тз} = T^{тз}_{рпз} + T^{тз}_{роз} \quad (4.1)$$

де $T^{тз}_{рпз}$ – витрати часу розроблювача постановки задач на розробку технічного завдання, л.-дн.,

$$T^{тз}_{рпз} = t_{тз} \times K^{тз}_{рпз} \quad (4.2)$$

$T^{тз}_{роз}$ – витрати часу розроблювача програмного забезпечення на розробку технічного завдання, л.-дн.,

$$Т^{тз}_{роз} = t_{тз} \times К^{тз}_{роз} \quad (4.3)$$

$t_{тз}$ – норма часу на розробку технічного завдання на програмний продукт у залежності від функціонального значення і ступеня новизни розроблювального ПП, 63 л.-дн.

$К^{тз}_{рпз}$ – коефіцієнт, що враховує питому вагу трудомісткості робіт, виконуваних розроблювачем постановки задач на стадії ТЗ (у випадку самостійної роботи $К^{тз}_{рпз} = 1$);

$К^{тз}_{роз}$ – коефіцієнт, що враховує питому вагу трудомісткості робіт, виконуваних розроблювачем програмного забезпечення на стадії ТЗ; $К^{тз}_{роз} = 0,35$.

Таким чином, трудомісткість роботи з формування технічного завдання складає:

$$Т_{тз} = 63 \times 0,65 + 63 \times 0,35 = 63 \text{ л.-дн.}$$

Етап розробки ескізного проекту складається з розробки макету ПП, уточнення вимог до продукту, вибору зовнішніх показників ПП (дизайн робочого вікна, зовнішній вигляд робочих документів). Робота проводиться окремо розроблювачем постановки задач та розроблювачем програмного забезпечення.

Трудомісткість розробки ескізного проекту $Т_{еп}$ розраховується за формулою:

$$Т_{еп} = Т^{еп}_{рпз} + Т^{еп}_{роз} \quad (4.4)$$

де $Т^{еп}_{рпз}$ – витрати часу розроблювача постановки задач на розробку ескізного проекту, л.-дн.,

$$Т^{еп}_{рпз} = t_{еп} \times К^{еп}_{рпз} \quad (4.5)$$

$Т^{еп}_{роз}$ – витрати часу розроблювача програмного забезпечення на розробку ескізного проекту, л.-дн.,

$$T^{еп}_{роз} = t_{еп} \times K^{еп}_{роз} \quad (4.6)$$

$t_{еп}$ – норма часу на розробку ескізного проекту програмного продукту в залежності від функціонального значення і ступеня новизни розроблювального ПП, 120 л.-дн.

$K^{еп}_{рпз}$ – коефіцієнт, що враховує питому вагу трудомісткості робіт, виконуваних розроблювачем постановки задач на стадії ескізного проектування (у випадку самостійної роботи $K^{еп}_{рпз} = 1$);

$K^{еп}_{роз}$ – коефіцієнт, що враховує питому вагу трудомісткості робіт, виконуваних розроблювачем програмного забезпечення на стадії ескізного проектування (у випадку самостійної роботи $K^{еп}_{роз} = 1$).

Таким чином, трудомісткість розробки ескізного проекту складає:

$$T_{еп} = 120 \times 1 + 120 \times 1 = 240 \text{ л.-дн.}$$

Етап розробки технічного проекту містить у собі роботи, пов'язані зі збором інформації, її обробкою, систематизацією тощо.

Режим обробки інформації здійснюється за допомогою телекомунікаційних засобів. У роботі використовуються також існуючи бази даних у кількості двох одиниць.

Трудомісткість розробки технічного проекту визначається за формулою:

$$T_{тп} = (t^{т}_{рз} + t^{т}_{рп}) \times K_{вi} \times K_{рi} \quad (4.7)$$

де $t^{т}_{рз}$,$t^{т}_{рп}$ – норми часу, затрачуваного на розробку технічного проекту розроблювачем постановки задачі та розроблювачем програмного забезпечення відповідно; $t^{т}_{рз} = 80$, $t^{т}_{рп} = 20$.

$K_{вi}$ – коефіцієнт урахування виду використовуваної інформації, визначається з виразу:

$$К_{ві} = (К_п \times n_п + К_{нс} \times n_{нс} + К_б \times n_б)/n_п + n_{нс} + n_б, \quad (4.8)$$

де $К_п$, $К_{нс}$, $К_б$ – значення коефіцієнтів урахування виду використовуваної інформації для перемінної, нормативно–довідкової інформації і баз даних відповідно; $К_п = 1,7$; $К_{нс} = 1,2$; $К_б = 3,14$.

$n_п$, $n_{нс}$, $n_б$ – кількість наборів даних перемінної, нормативно–довідкової інформації і баз даних відповідно; $n_п = 3$, $n_{нс} = 3$, $n_б = 3$.

$К_{рі}$ – коефіцієнт урахування режиму обробки інформації. $К_{рі} = 1,6$.

Таким чином, трудомісткість розробки технічного проекту дорівнює:

$$Т_{тп} = (80 + 20) \times ((1,7*3 + 1,2* 3 + 3,14*3)/(3+3+3)) \times 1,6 = 322,13 \text{ л.-дн.}$$

Етап розробки робочого проекту складається з складання алгоритму ПП, визначення можливості та доцільності використання готових програмних модулів, вибору алгоритмічної мови програмування та безпосереднього формування ПП. У проекті використано алгоритмічна мова високого рівня, ступінь використання готових програмних модулів біля 30%.

Трудомісткість розробки робочого проекту визначається за формулою:

$$Т_{рп} = (t^{рп}_{рз} + t^{рп}_{рп}) \times К_к \times К_я \times К_г \times К_{са}, \quad (4.9)$$

де $t^{рп}_{рз}$, $t^{рп}_{рп}$ – норми часу, витраченого на розробку робочого проекту алгоритмічною мовою високого рівня розроблювачем постановки задачі та розроблювачем програмного забезпечення відповідно; $t^{рп}_{рз} = 30$, $t^{рп}_{рп} = 180$.

$К_к$ – коефіцієнт урахування складності контролю інформації; $К_к = 1,18$.

$К_я$ – коефіцієнт урахування рівня використовуваної алгоритмічної мови програмування; $К_я = 1$.

$К_г$ – коефіцієнт урахування ступеня використання готових програмних модулів; $К_г = 0,8$.

$К_{са}$ – коефіцієнт урахування виду використовуваної інформації і складності алгоритму ПП.

Значення цього коефіцієнта визначається за формулою:

$$К_{са} = (К'_{п} \times n_{п} + К'_{нс} \times n_{нс} + К'_{б} \times n_{б})/n_{п} + n_{нс} + n_{б}, \quad (4.10)$$

де $К'_{п}$, $К'_{нс}$, $К'_{б}$ – значення коефіцієнтів урахування складності алгоритму ПП і виду використовуваної інформації для перемінної, нормативно–довідкової інформації і баз даних відповідно;

$n_{п}$, $n_{нс}$, $n_{б}$ – кількість наборів даних перемінної, нормативно–довідкової інформації і баз даних відповідно.

Таким чином, трудомісткість розробки робочого проекту складає:

$$Т_{рп} = (30+180)\times 1{,}18\times 1\times 0{,}8\times((1{,}36\times 3+0{,}9\times 2+0{,}8\times 3)/(3+2+3)) = 312{,}99 \text{ л.-дн.}$$

Етап впровадження полягає у впровадженні готового ПП у виробництво, налагодження його на місці, усунення виявлених недоліків та погрішностей у роботі.

Трудомісткість робіт на стадії впровадження дорівнює:

$$Т_{в} = (t^{в}_{рз} + t^{в}_{рп}) \times К_{к} \times К_{я} \times К_{г}, \quad (4.11)$$

де $t^{в}_{рз}$ та $t^{в}_{рп}$ – норми часу, витраченого на виконання процедури впровадження розроблювачем постановки задачі та розроблювачем програмного забезпечення відповідно; $t^{в}_{рз} = 33$, $t^{в}_{рп} = 31$.

$К_{к}$ – коефіцієнт урахування складності контролю інформації; $К_{к} = 1{,}17$.

$К_{я}$ – коефіцієнт урахування рівня використовуваної алгоритмічної мови програмування; $К_{я} = 1$.

$К_{г}$ – коефіцієнт урахування ступеня використання готових програмних модулів. $К_{г} = 0{,}8$.

Таким чином, трудомісткість робіт на стадії впровадження складає:

$$Т_{в} = (33+31) \times 1{,}17\times 1\times 0{,}8 = 59{,}904 \text{ л.-дн.}$$

Загальна трудомісткість розробки програмного продукту визначається як сума величин трудомісткості виконання окремих стадій розробки ПП:

$$T_{пп}= T_{тз}+ T_{еп}+T_{тп}+T_{рп}+T_{в} =35+150+200,96+130,23+26,24=542,43 \text{ л.-дн.}$$

Визначення необхідної кількості працівників. Кількість працівників визначається в залежності від розрахованої трудомісткості за формулою:

$$Ш= T_{пп} / Н \times Г,$$

де $T_{пп}$ – загальна трудомісткість розробки ПП ;

Н – середня кількість робочих днів у місяці (20,6 дн.);

Г – тривалість робочого часу (8 годин).

Таким чином, для реалізації даного проекту необхідна наступна кількість працівників:

$$Ш= 824,08/20,6 \times 8=4,95 \approx 5 \text{ людей.}$$

4.2 Розрахунок вартості програмного продукту

Визначення основної заробітної плати проводиться на основі таблиці кількості виконавців та тарифних ставок (таблиця 4.1).

Таблиця 4.1

Розрахунок основної заробітної плати

Посада, науковий ступень, звання	Кількість працівників	Тарифна ставка, грн./міс.	Всього, грн./міс.
Відповідальний виконавець	1	15000	15000
Лаборант	1	12000	12000
Інженер 1 кат	1	10000	10000
Усього	3		37000

Основна зарплата виконавців ($З_о$) на місяць складає 37000 грн.

Визначимо вартість програмного продукту за допомогою таблиці 4.2.

Таблиця 4.2

Розрахунок вартості програмного продукту

Статті витрат	Формула для розрахунку елементів витрат ($В^i$)	Сума, грн.
Заробітна плата, у тому числі:	З	
основна заробітна плата	$З_о$ (розраховується за штатним розкладом)	37000
додаткова заробітна плата	$З_д=0,1\times З_о$	3700
Відрахування у загально державні цільові фонди (згідно з чинним законодавством), в т.ч.:	$В_{відрах}=В_{пф}+В_{вс}$	7937
ПДФО (податок на доходи фізичних осіб)	$В_{пф}=0,18\times(З_о+З_д)$	7326
військовий збір	$В_{вс}=0,015\times(З_о+З_д)$	611
Витрати на відрядження	$В_в=0,05\times З_о$	1850
Сировина та матеріали	$В_м=0,2\times З_о$	7400
Комерційні витрати	$В_к=0,2\times З_о$	7400
Накладні витрати, у тому числі послуги сторонніх організацій	$В_н=0,15\times З_о$	5550
Інші витрати	$В_{ін}=0,13\times З_о$	4810
Повні витрати	$В_п=\Sigma В$	34947

Тривалість усіх робіт з розробки ПП визначається за формулою:

$$Т_і=\frac{t_i+Q}{Нn_i}, \quad (4.3)$$

де t_i – трудомісткість i -ї роботи, л.-дн.;

Q – трудомісткість виконуваних додаткових робіт, л.-дн.;

Н – середня кількість робочих днів у місяці (20,6 дн.);

n_i – кількість виконавців, що виконують i-у роботу, чол.

Якщо припустити, що у середньому обсяг додаткових робіт складає 10–15%, то можна визначити середню тривалість робот наступним чином:

$$T_i = 542{,}43 \times 1{,}1/20{,}6 \times 5 \approx 6 \text{ місяців.}$$

Загальна вартість ПП визначається наступним чином:

$$B_з = B \times T_i = 20590 \times 6 = 123540 \text{ грн.}$$

Вартість ПП з урахуванням ПДВ становить

$$B_{ПДВ} = 123540 \times 1{,}2 = 148248 \text{ грн.}$$

Зведений розрахунок вартості розробки ПП наведено у таблиці 4.3.

Таблиця 4.3

Розрахунок трудомісткості та вартості ПП

№	Назва даних		Значення
1	Трудомісткість розробки технічного завдання, л.-дн.	($T_{тз}$)	63,00
2	Трудомісткість розробки ескізного проекту, л.-дн.	($T_{еп}$)	240,00
3	Трудомісткість розробки технічного проекту, л.-дн.	($T_{тп}$)	322,13
4	Трудомісткість розробки робочого проекту, л.-дн.	($T_{рп}$)	312,99
5	Трудомісткість впровадження розробленого ПП, л.-дн.	($T_в$)	59,904
6	Загальна трудомісткість, л.-дн.	($T_{пп}$)	542,43
7	Термін виконання роботи, міс.	(T_i)	6
8	Середня кількість робочих днів в одному місяці, дн.	(Н)	20,6
9	Кількісний склад виконавців, чол.	(Ш)	3
10	Основна зарплата виконавців на місяць, грн.	($З_{ср}$)	37000
11	Витрати на роботу в гривнях на місяць (без ПДВ), грн.	(В)	34947
12	Загальна вартість роботи без ПДВ, грн.		71947
13	Загальна вартість роботи з ПДВ (20%), грн.	($В_{ПДВ}$)	79142

Висновок до четвертого розділу

В результаті виконання даного розділу було отримано такі результати:

1. На основі проведеного аналізу інструментів побудови нейронних мереж було прийнято рішення про побудову власної нейронної мережі засобами середовища програмування Python.

2. Визначено етапи розробки програмного продукту і розрахована трудомісткість кожного виконання кожного етапу.

3. Визначена трудомісткість і вартість програмного продукту.

Ці результати дозволяють обґрунтовано робити висновки щодо доцільності реалізації даного проекту і планувати ефект від його реалізації і впровадження.

РОЗДІЛ 5
ОХОРОНА ПРАЦІ НА РОБОЧОМУ МІСЦІ

5.1 Загальні положення

Дана кваліфікаційна робота присвячена побудові, тренуванню і дослідженню нейронних мереж, призначених для вирішення задач розпізнавання. Оскільки дослідники багато годин працюють за персональним комп'ютером (ПК), вони мають слідувати правилам і рекомендаціям, що стосуються охорони праці з комп'ютером.

Продуктивність та характер діяльності людини прямо пов'язані з виконанням конкретних робіт та ефективністю праці. Остання визначається як впливом людського фактору, так і використанням засобів виробництва, а також умовами, пов'язаними з технологією та організацією праці. В сучасній промисловій сфері більшість працівників займається роботою, пов'язаною з використанням комп'ютерної техніки. Працюючи за комп'ютером, людина стикається з різноманітними факторами, такими як електромагнітні поля, інфрачервоне та іонізуюче випромінювання, шум, вібрації та статична електрика.

Робота за комп'ютером супроводжується нервовим навантаженням та потребує значної ментальної напруги для операторів. Вона також вимагає високого рівня зорової активності та значного фізичного навантаження на руки під час взаємодії з клавіатурою. Раціональна конструкція та розташування елементів робочого місця мають велике значення для забезпечення оптимальної робочої позиції під час праці за комп'ютером.

Під час роботи з комп'ютером важливо дотримуватись належного режиму праці та відпочинку. В іншому випадку працівники можуть відчувати незадоволення роботою, головний біль, роздратування, порушення сну, втому та відчувати біль в очах, спині, шиї та руках.

Обчислювальна техніка виділяє тепло, що може призвести до підвищення температури та зниження вологості у приміщенні.

Рівень шуму на робочому місці оператора не повинен перевищувати встановлені норми. Для зниження рівня шуму стіни та стеля в приміщенні, де знаходяться комп'ютери, повинні бути облицьовані матеріалами, що поглинають звук.

Оптичне випромінювання включає ультрафіолетове, видиме світло та інфрачервоне випромінювання.

Ультрафіолетове випромінювання впливає на шкіру та очі людини. Проте, аналіз досліджень робочих місць користувачів комп'ютерів показує, що у 86 % випадків ультрафіолетове випромінювання не було виявлено.

Світлове випромінювання, переважно, впливає на очі і може викликати втому та запалення райдужної оболонки. Однак ці симптоми швидко зникають і не спричиняють патологічних змін.

Електромагнітне випромінювання (ЕМВ) у радіочастотному діапазоні є основним джерелом ЕМВ в робочому середовищі, зокрема від монітора. Тому при обиранні місця для комп'ютера необхідно враховувати, що задня і бокові стінки можуть бути джерелом значно більшого ЕМВ, ніж сам екран.

Наукові дослідження свідчать про те, що радіочастотне випромінювання впливає на центральну нервову систему і є значним стрес-фактором.

Щоб зменшити вплив згаданих видів випромінювання, рекомендується використовувати монітори з низьким рівнем випромінювання, а також дотримуватись регламентованого режиму праці та відпочинку.

5.2 Способи зниження впливу шкідливих та небезпечних факторів при роботі з комп'ютером

Щодо зниження впливу шкідливих та небезпечних факторів під час роботи з комп'ютером, необхідно, зокрема, правильно розташовувати обладнання та електричні кабелі, щоб уникнути ризику ураження

електричним струмом. Інші заходи, пов’язані з електробезпекою, відповідають загальним вимогам щодо пожежної та електробезпеки.

Екран монітора повинен бути розташованим перпендикулярно до напрямку погляду. Якщо він нахилений, це може спричинити погіршення постави. Відстань між очима та екраном повинна трохи перевищувати звичайну відстань між очима та книгою. Якщо на моніторі, особливо на старих моделях, відсутній захисний екран, необхідно сидіти на відстані витягнутої руки від нього. Ще одним аспектом, що стосується зору, є створення неоднорідного поля зору. Це можна досягти, розмістивши на стінах плакати або картини з спокійними кольорами, наприклад, пейзажі або натюрморти.

Форма спинки крісла має відповідати формі спини. Висота крісла має бути така, щоб користувач не відчував тиску на стегна або куприк. Крісло бажано обладнати підлокітниками, а його розташування має бути зручним, щоб не доводилося напружуватись, щоб дістатися до клавіатури. Дуже важливими є регулярні переміщення та зміна положення тіла, а також перерви у роботі.

Під час напруженої роботи за комп’ютером рекомендується робити перерви тривалістю 15 хвилин кожну годину і займатися іншими справами. Кілька разів на годину корисно виконати серію легких вправ для розслаблення.

Якщо не дотримуватись заходів безпеки під час роботи за комп’ютером, можуть виникнути такі наслідки, як:

- проблеми з органами зору (спостерігаються у 60 % користувачів);
- захворювання серцево-судинної системи (20 % користувачів);
- захворювання шлунково-кишкового тракту (10 % користувачів);
- шкіряні проблеми (5 % користувачів);
- ризик виникнення різноманітних пухлин.

Якщо у приміщенні використовується більше одного комп’ютера, важливо враховувати, що на користувача може впливати випромінювання від інших комп’ютерів, зокрема з бокових, та задньої стінки сусідніх дисплеїв. Тому необхідно встановити спеціальні фільтри та забезпечити, щоб користувач розміщувався на відстані не менше одного метра від бічних і задніх стінок інших дисплеїв.

Отже, для запобігання негативним впливам важливо бути ознайомленим з небезпечними аспектами самого комп’ютера, правилами безпечної роботи з ним, засобами запобігання ризикам, особливо пов’язаним з відомими загрозами, такими як електричні ураження та пожежна небезпека.

5.3 Правила безпеки при роботі з комп’ютером

Перед початком роботи на комп’ютері користувач повинен переконатися у цілісності корпусу та компонентів комп’ютера, а також перевірити наявність заземлення, стан та цілісність живильних кабелів та їх правильне підключення. У разі виявлення несправностей вмикати комп’ютер та розпочинати роботу заборонено.

Під час роботи, після переконання у справності обладнання, можна увімкнути живлення комп’ютера і почати роботу, дотримуючись умов, зазначених у відповідному посібнику з експлуатації.

Заборонено:

- замінювати різні деталі або компоненти під час роботи комп’ютера;
- з’єднувати або від’єднувати вилки та розетки живильної мережі, які знаходяться під напругою;
- відкривати кришки, що закривають доступ до струмопровідних частин живильної мережі під час роботи обладнання;
- використовувати паяльник з не заземленим корпусом;
- замінювати запобіжники під напругою;
- залишати комп’ютер увімкненим без нагляду.

Після закінчення робочого дня необхідно:

- вимкнути живлення комп'ютера, натиснувши відповідну кнопку та вийнявши вилку живильного кабелю з розетки, дотримуючись інструкцій з експлуатації;
- прибрати робоче місце користувача комп'ютера, відклавши використане обладнання та матеріали на призначені для них місця;
- у разі виявлення дефектів під час роботи комп'ютера – повідомити відповідним посадовим особам та спеціалістам.

5.4 Пожежна профілактика

Для забезпечення пожежної безпеки потрібно, в першу чергу, визначити типи та кількість первинних засобів пожежогасіння. При цьому необхідно враховувати фізико-хімічні та пожежонебезпечні властивості горючих речовин, їх взаємодію з вогнегасниками, а також площу виробничих приміщень, установок та відкритих майданчиків.

Для загасання невеликих вогнищ пожеж, де горіння не може відбуватися без доступу повітря призначені азбестові полотна, грубошерстяні тканини та повсті розміром не менше 1 кв. м.

У пожежному стенді мають бути ємності для піску об'ємом не менше 0,1 куб. м. Конструкція ящика з піском має забезпечувати зручний доступ до нього і запобігати потраплянню опадів або проникненню вологи іншими маршрутами.

Комплектація технологічного обладнання вогнегасниками повинна відповідати вимогам технічних умов (паспортів) на це обладнання або відповідним правилам пожежної безпеки.

Необхідно вибирати тип вогнегасників і розраховувати їх необхідну кількість залежно від їх вогнегасної здатності, максимальної площі, класу

пожежі горючих речовин і матеріалів у закритих приміщеннях або на об’єктах згідно з ISO N 3941-77.

У замкнутих приміщеннях об’ємом до 50 куб. м для загасання пожеж можна використовувати портативні вогнегасники або додатково використовувати порошкові вогнегасники.

При виборі вогнегасника з врахуванням відповідної температурної межі використання необхідно враховувати кліматичні умови, в яких будуть експлуатуватися будівлі та споруди.

Вогнегасники, які були відправлені на перезарядку з підприємства, повинні бути замінені на відповідну кількість заряджених вогнегасників.

При захисті приміщень з ПК слід враховувати особливості взаємодії вогнегасних речовин з обладнанням, виробами, матеріалами та іншими елементами. Для цих приміщень рекомендується встановлювати вуглекислотні вогнегасники, враховуючи максимально допустиму концентрацію вогнегасної речовини.

Приміщення, які обладнані автоматичними стаціонарними установками пожежогасіння, повинні мати вогнегасники на 50 % від їх розрахункової кількості. Відстань від можливого вогнища пожежі до місця розташування вогнегасника не повинна перевищувати: 20 м для громадських будівель і споруд, 30 м для приміщень категорій А, Б і В, 40 м для приміщень категорії Г та 70 м для приміщень категорії Д.

Використання первинних засобів пожежогасіння для господарських та інших потреб, які не пов’язані з гасінням пожежі, заборонено.

З урахуванням типу будівлі (громадське приміщення) та можливого класу пожежі (Е), оскільки у приміщенні є багато комп’ютерів, визначаємо необхідну кількість вогнегасників – один порошковий вогнегасник об’ємом 5 літрів.

Висновки до п’ятого розділу

У даній частині дипломного проекту були розглянуті питання щодо гігієнічних норм організації і обладнання робочих місць користувачів персональних комп’ютерів, питання пожежної профілактики приміщень.

Отримані результати і висвітлені положення дозволяють обладнати приміщення для роботи користувачів обчислювальної техніки з дотриманням вимог безпеки їх праці.

ЗАГАЛЬНІ ВИСНОВКИ

Нейронні мережі, які вирішують задачі розпізнавання і класифікації образів, є дуже актуальними сьогодні. Вони використовуються у багатьох сферах, включаючи медицину, автомобільну промисловість, безпеку, роздрібну торгівлю та багато інших. Окремою задачею, яка залишається досі актуальною, є розпізнавання рукописного тексту. Саме для цього і була спроектована і реалізована нейронна мережі в цій роботі.

В роботі були отримані такі результати:

1. В першому розділі були проаналізовані основні поняття в області нейронних мереж, зокрема, принципи їх функціонування, основи будови нейрону, принципи моделювання нейронних мереж. Також був виконаний порівняльний аналіз інструментів побудови нейронних мереж, на основі якого було обрано загальний підхід до подальшого проектування і реалізації нейронної мережі.

В другому розділі кваліфікаційної роботи було виконано математичне моделювання тришарової нейронної мережі, призначеної для вирішення задач розпізнавання. Зокрема, були проаналізовані питання прямого і зворотного розповсюдження сигналів в мережі, оновлення вагових коефіцієнтів на основі даних про помилки і особливості підготовки вхідних даних для передавання на вхід нейронної мережі.

В третьому розділі кваліфікаційної роботи було обгрунтовано засіб реалізації нейронної мережі, виконано програмну реалізацію нейронної мережі у вигляді класу на мові програмування Python і здійснено її навчання розпізнаванню рукописним цифрам на основі бази даних MNIST. Також було виконано дослідження шляхів поліпшення результатів розпізнавання і підбору оптимальних параметрів мережі. Нарешті, було підготовлено набір з власноруч написаних цифр і протестовано навчену мережу на предмет можливості і ефективності їх розпізнавання.

В результаті було отримано деякі важливі загальні результати щодо методів побудови і тренування нейронних мереж, а також покращення результатів їх роботи. Виявлено основні параметри, які можна регулювати для досягнення цієї мети, а саме: коефіцієнт навчання, кількість вузлів проміжного шару нейронної мережі та кількість епох навчання. Отримані результати дозволять проводити в майбутньому більш грунтовні дослідження стосовно використання нейронної мережі на інших наборах даних та, взагалі, для вирішення інших задач.

Також виконано техніко-економічний аналіз розробки і розробку питань охорони праці користувача.

Напрямами подальшого розвитку роботи можуть стати підвищення точності розпізнавання, підбір більш вдалих комбінацій структурних параметрів самої мережі і параметрів її навчання, а також реалізація інших задач, в тому числі, розпізнавання, наприклад, суцільного рукописного тексту без розривів між символами слова.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Басюркіна Н. Й., Свистун Т. В. Методичні вказівки до оцінки науково-технічної ефективності розробки нової технології, нового обладнання та інших інновацій. Для студентів всіх спеціальностей СВО «бакалавр» і «магістр» денної і заочної форм навчання. – Одеса : ОНТУ, 2022. – 18 с.
2. Вігуржинська С. Ю., Колесник В. І. Дипломне проектування економічної частини проекту: Методичні вказівки . – Одеса : ОНАХТ, 2016. – 22 с.
3. Вісловух А. М. Охорона праці користувачів персональних комп’ютерів (ПК): Навчальний посібник. – К. : ІПК ДСЗУ, 2007. – 55 с.
4. Дмитрієнко В. Д., Основи нейрокомп’ютингу: навчально-методичний посібник до практичних занять [Текст] / В. Д. Дмитрієнко, О. Ю. Заковоротний, В. І. Носков, М. В. Мезенцев. – Х.: НТМТ, 2014. – 140 с.
5. Кононюк А. Ю. Нейроні мережі і генетичні алгоритми. Науково-практичне видання. – Київ : Корнійчук, 2008. – 446 с.
6. Любунь З. М. Основи теорії нейромереж. – Львів : Видавничий центр ЛНУ ім. Івана Франка, 2006. – 140 с.
7. Руденко О. Г. Штучні нейронні мережі: Навчальний посібник. [Текст] / О. Г. Руденко, Є. В. Бодянський. – Харків: ТОВ "Компанія СМІТ", 2006. – 404 с.
8. Субботін С. О. Нейронні мережі: теорія та практика. Навчальний посібник. – Житомир : Вид. О. О. Євенок, 2020. – 184 с.
9. Тимощук П. В. Штучні нейронні мережі. Навчальний посібник. – Львів : Видавництво Львівської Політехніки, 2011. – 444 с.
10. Троцько В. В. Методи штучного інтелекту – К. : Університет "КРОК", 2020. – 86 с.
11. Campbell Sam. Neural Networks for Beginners: Comprehensive Guide to Understanding the Power of Artificial Intelligence. – Independently published, 2023. – 80 p.

12. Dhaliwal R.S. Neural Networks / R.S. Dhaliwal, T. Lepage-Richer, L. Suchman. – Minneapolis: University of Minnesota Press, 2024. – 125 p.
13. Dingli Alexiei, Farrugia David. Neuro-Symbolic AI: Design transparent and trustworthy systems that understand the world as you do. – Packt Publishing, 2023. – 196 p.
14. Downing Keith L. Gradient Expectations: Structure, Origins, and Synthesis of Predictive Neural Networks. – MIT Press, 2023. – 224 p.
15. El-Baz A.S., Suri J.S. (eds.) State of the Art in Neural Networks and Their Applications: Volume 1 – Academic Press, 2021. – 310 p.
16. El-Baz A.S., Suri J.S. (eds.) State of the Art in Neural Networks and Their Applications: Volume 2. – Academic Press/Elsevier, 2023. – 328 p.
17. Ghosal I. The Future of Artificial Neural Networks / I. Ghosal, A. Mittal, H. Jain. – Nova Science Publishers, 2024. – 199 p.
18. Koyamada K. Analysis and Visualization of Discrete Data Using Neural Networks. – World Scientific Publishing, 2024. – 230 p.
19. Rashid T. Make Your Own Neural Network. – CreateSpace Independent Publishing Platform, 2016. – 222 p.
20. Yang S., Chen B. Neuromorphic Intelligence: Learning, Architectures and Large-Scale Systems. – Springer Cham, 2024. – 239 p.