ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «НЕЙРОННА МЕРЕЖА ДЛЯ РОЗПІЗНАВАННЯ ЗОБРАЖЕНЬ З ВИКОРИСТАННЯМ МОВИ ПРОГРАМУВАННЯ PYTHON»

на здобуття освітнього ступеня магістра
зі спеціальності 126 Інформаційні системи та технології
освітньо-професійної програми Інформаційні системи та технології

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ *Артем ШТАНЬКО*

Виконав:
здобувач вищої освіти Артем ШТАНЬКО
група ІСДМ-63

Керівник: Оксана ТКАЛЕНКО
науковий ступінь, вчене звання к.т.н., доцент

Рецензент:
науковий ступінь, вчене звання _____
Ім'я, ПРІЗВИЩЕ

Київ – 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Інформаційних систем та технологій

Ступінь вищої освіти Магістр

Спеціальність Інформаційні системи та технології

Освітньо-професійна програма Інформаційні системи та технології

ЗАТВЕРДЖУЮ
Завідувач кафедрою ІСТ

_____Каміла СТОРЧАК
«____» _____20___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ СТУДЕНТУ
Штаньку Артему Олександровичу
(прізвище, ім'я, по батькові здобувача)

1.Тема кваліфікаційної роботи: «Нейронна мережа для розпізнавання зображень з використанням мови програмування Python»

керівник кваліфікаційної роботи Оксана ТКАЛЕНКО, к.т.н., доцент
(ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від «15» жовтня 2024 року № 320.

2. Строк подання кваліфікаційної роботи: 27 грудня 2024 року.

3. Вихідні дані до кваліфікаційної роботи: Методи http GET, POST, PUT, PATCH, DELETE;
Протоколи HTTP, HTTPS, SSH;
Фреймворки TensorFlow, Keras, PyTorch, Theano.
Науково-технічна література з питань, пов'язаних з наукою про дані.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Методологія по дослідженню даних CRISP-DM.

2. Дослідження особливостей машинного та глибокого навчання.
3. Розробка моделі нейронної мережі для ідентифікації зображень.

5. Перелік ілюстративного матеріалу: *презентація*
   1. Методологія по дослідженню даних CRISP-DM.
   2. Методи машинного навчання.
   3. Порівняльна характеристика ML та DL.
   4. Структура штучної нейронної мережі.
   5. Фреймворки TensorFlow, Keras, Pytorch.
   6. Функції активації нейромереж.
   7. Практична реалізація нейронної мережі.

6. Дата видачі завдання: 15 жовтня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	15.10 – 24.10.24	
2	Дослідження особливостей науки про дані	25.10 – 01.11.24	
3	Дослідження особливостей машинного та глибокого навчання	02.11 – 08.11.24	
4	Дослідження особливостей побудови нейронних мереж	09.11 – 14.11.24	
5	Вивчення особливостей фреймворків TensorFlow, Keras, PyTorch, Theano	15.11 – 20.11.24	
6	Практична реалізація нейронної мережі	21.11 – 09.12.24	
7	Оформлення роботи: вступ, висновки, реферат	10.12 – 18.12.24	
8	Розробка демонстраційних матеріалів	19.12 – 25.12.24	

Здобувач вищої освіти _____ Артем ШТАНЬКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник роботи
кваліфікаційної роботи _____ Оксана ТКАЛЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 77 стор., 31 рис., 9 табл., 20 джерел.

Мета роботи – розробка та реалізація нейронної мережі для розпізнавання зображень з використанням мови програмування Python.

Об'єкт дослідження – процес розпізнавання зображень за допомогою нейронних мереж.

Предмет дослідження – метод побудови нейронних мереж для розпізнавання зображень з використанням мови програмування Python.

Короткий зміст роботи: Досліджені методи та технології обробки даних в інформаційних системах, математичні інструменти для аналізу та обробки даних. Досліджені особливості машинного та глибокого навчання, особливості побудови нейронних мереж, найпопулярніші фреймворки для роботи з технологіями ML/DL: TensorFlow, Keras, PyTorch, Theano, визначені їх позитивні сторони та недоліки, надані рекомендації по використанню зазначених фреймворків. Розроблено нейронну мережу для розпізнавання зображень в інтерактивному середовищі Jupyter Notebook, яке є популярним інструментом серед дослідників, програмістів та аналітиків даних. Визначено роль штучного інтелекту в аналізі та обробці даних. Здійснено програмування нейронної мережі.

КЛЮЧОВІ СЛОВА: ДАНІ, НЕЙРОННА МЕРЕЖА, МОВА ПРОГРАМУВАННЯ PYTHON, ВІЗУАЛІЗАЦІЯ, ТЕХНОЛОГІЯ, АЛГОРИТМ, СОКЕТ, ПРОТОКОЛ, ІНТЕРАКТИВНЕ СЕРЕДОВИЩЕ.

ABSTRACT

Text part of the master`s qualification work: 77 pages, 31 pictures, 9 table, 20 sources.

The purpose of the work is to carry out a comprehensive analysis of data processing methods and technologies in information systems, as well as the development of a neural network for object recognition.

Object of research is the process of data processing in information systems.

Subject of research is methods and technologies of data processing in information systems.

Summary of the work: Methods and technologies of data processing in information systems, mathematical tools for data analysis and processing were studied. The features of machine and deep learning, the features of building neural networks, the most popular frameworks for working with ML/DL technologies: TensorFlow, Keras, PyTorch, Theano were investigated, their positive sides and disadvantages were determined, and recommendations were given for the use of these frameworks. A neural network was developed to recognize handwritten digits in the interactive Jupyter Notebook environment, which is a popular tool among researchers, programmers, and data analysts. The role of artificial intelligence in data analysis and processing was determined. Neural network programming was carried out.

KEYWORDS: DATA, NEURAL NETWORK, PYTHON PROGRAMMING LANGUAGE, VISUALIZATION, TECHNOLOGY, ALGORITHM, SOCKET, PROTOCOL, INTERACTIVE ENVIRONMENT.

ЗМІСТ

ВСТУП……..9
РОЗДІЛ 1. МЕТОДОЛОГІЯ ПО ДОСЛІДЖЕННЮ ДАНИХ CRISP-DM……....11
1.1 Визначення ролі CRISP-DM у розробці нейронних мереж.....11
1.2 Етап моделювання даних……....16
1.3 Визначення ролі датасетів у ML та DL…….21
РОЗДІЛ 2. ДОСЛІДЖЕННЯ ОСОБЛИВОСТЕЙ МАШИННОГО ТА ГЛИБОКОГО НАВЧАННЯ……...……27
2.1 ML/DL – цілі та принципи роботи…….27
2.2 Дослідження особливостей побудови нейронних мереж……...….36
2.3 Характеристика розповсюджених видів нейромереж.....40
2.4 Огляд мов програмування та популярних бібліотек для роботи з нейронними мережами……....47
РОЗДІЛ 3. РОЗРОБКА МОДЕЛІ НЕЙРОННОЇ МЕРЕЖІ ДЛЯ ІДЕНТИФІКАЦІЇ ЗОБРАЖЕНЬ……..53
3.1 Виконання імпорту та характеристика необхідних бібліотек…….....53
3.2 Визначення способу створення моделі нейронної мережі……..….55
3.3 Вибір функцій активації для створення нейронної мережі……59
3.4 Оцінка якості роботи моделі нейронної мережі……...64
3.5 Результати виконання моделювання в середовищі JN……....68
ВИСНОВКИ……...74
ПЕРЕЛІК ПОСИЛАНЬ…….....75
ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)……77

ВСТУП

Актуальність теми. З кожним роком зростає кількість застосувань технологій розпізнавання зображень у різних сферах: від медичної діагностики до автоматизованого контролю якості на виробництві та розпізнавання об'єктів в автономних транспортних засобах. Розробка ефективних моделей нейронних мереж для цих завдань стає надзвичайно важливою, оскільки вони дозволяють досягати високої точності та швидкості обробки зображень. Мова програмування Python є однією з найбільш популярних для розробки та впровадження таких технологій завдяки широкому набору бібліотек для роботи з даними та побудови моделей машинного навчання, таких як TensorFlow, Keras, PyTorch. Python дозволяє швидко реалізувати та протестувати алгоритми, що робить його ідеальним інструментом для розробки нейронних мереж для задач розпізнавання зображень. Розробка нейронної мережі для розпізнавання зображень з використанням Python є актуальним напрямом досліджень, що відкриває перспективи для подальшого розвитку технологій штучного інтелекту та їх застосування у реальних проєктах.

Метою роботи є розробка та реалізація нейронної мережі для розпізнавання зображень з використанням мови програмування Python.

Для досягнення поставленої мети необхідно *виконати наступні завдання*:

- дослідити основні типи нейронних мереж та їх застосування для розпізнавання зображень;
- дослідити методи попередньої обробки даних;
- визначити технології для обробки та аналізу даних;
- реалізувати нейронну мережу для розпізнавання цифр.

Об'єкт дослідження - процес розпізнавання зображень за допомогою нейронних мереж.

Предметом дослідження є метод побудови нейронних мереж для розпізнавання зображень з використанням мови програмування Python.

Наукова новизна отриманих результатів полягає у розробці та реалізації нейронної мережі для розпізнавання зображень за допомогою мови програмування Python, забезпечуючи стабільне і швидке навчання моделі, що є важливим для досягнення оптимальних результатів при обробці великих обсягів зображень.

Апробація результатів магістерської роботи:

1. Штанько А.О. «Аналіз технологій штучного інтелекту». Тези доповіді на II Всеукраїнській науково-технічній конференції «Технологічні горизонти: дослідження та застосування інформаційних технологій для технологічного прогресу України і світу». – Київ, 18 листопада 2024 року.

2. Штанько А.О. «Використання мови програмування Python для задач штучного інтелекту». Наукова стаття у загальногалузевому науково-виробничому журналі «Зв'язок», м.Київ - №5 (171), вересень-жовтень 2024 року, С.54-58.

1 МЕТОДОЛОГІЯ ПО ДОСЛІДЖЕННЮ ДАНИХ CRISP-DM

1.1 Визначення ролі CRISP-DM у розробці нейронних мереж

Методологія CRISP-DM (Cross-Industry Standard Process for Data Mining) є широко визнаним стандартом для виконання проєктів з аналізу даних та має важливе значення у розробці нейронних мереж. Досліджуючи дану методологію, було визначено роль CRISP-DM у розробці нейронних мереж. По-перше, методологія CRISP-DM забезпечує структурованість процесу розробки нейронної мережі - надає структурований підхід, що допомагає організувати та спланувати проєкт з розробки нейронної мережі. По-друге, забезпечує якість даних: розуміння та підготовка даних допомагають забезпечити високу якість вхідних даних, що є критично важливим для ефективності нейронних мереж. По-третє, дана методологія сприяє цілеспрямованості: завдяки бізнес-розумінню методологія допомагає орієнтувати розробку нейронної мережі на конкретні бізнес-цілі та задачі. В четверте, етапи моделювання та оцінки дозволяють експериментувати з різними архітектурами нейронних мереж та параметрами, а також оцінювати їхню продуктивність перед впровадженням. По-п’яте, CRISP-DM забезпечує моніторинг і підтриму, адже це забезпечить стабільну роботу нейронних мереж у реальному середовищі, дозволяючи вчасно реагувати на зміни та вдосконалювати модель.

Таким чином, CRISP-DM допомагає систематично та ефективно проходити всі етапи розробки нейронних мереж від початкового розуміння задачі до впровадження та підтримки моделі в реальному світі.

Методологія CRISP-DM визначає покроковий підхід до розробки, побудови та впровадження моделей для аналізу даних (рис.1.1).

CRISP-DM є стандартом у галузі аналізу даних і забезпечує основу для успішного проведення проєктів з використанням даних, допомагаючи командам структурувати свою роботу та досягати бажаних результатів. Дана методологія складається з шести етапів:

1. Бізнес-розуміння (Business Understanding);

2. Розуміння даних (Data Understanding);

3. Підготовка даних (Data Preparation);

4. Моделювання (Modeling);

5. Оцінка (Evaluation);

6. Розгортання (Deployment).

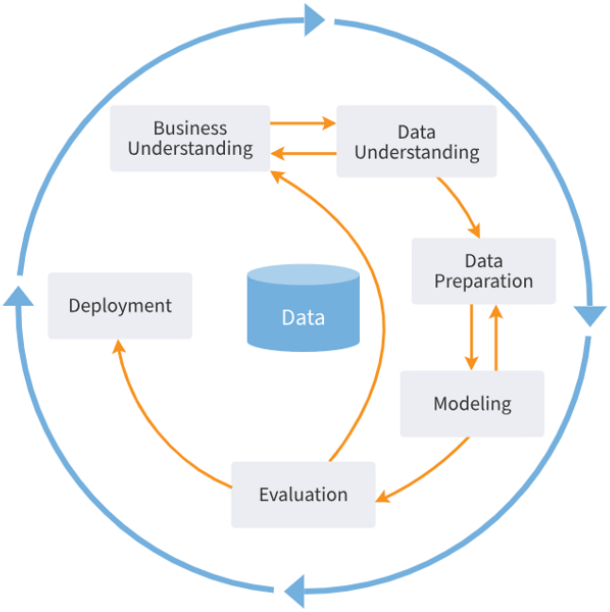


Рис. 1.1 CRISP-DM

Блок *Business Understanding* має наступне призначення: визначення бізнес-цілей; розуміння бізнес-вимог; формулювання завдань для аналізу даних.

Етап визначення бізнес-цілей полягає у розумінні бізнес-проблеми або можливості, яку ми хочемо вирішити за допомогою аналізу даних. Важливо чітко сформулювати, що саме потрібно досягти і які цілі мають бути досягнуті для успішного впровадження проєкту. На етапі розуміння бізнес-вимог важливо встановити вимоги і очікування від результатів проєкту з точки зору бізнесу. Це включає визначення метрик успіху, ключових показників ефективності, а також розуміння, які аналітичні здібності потрібні для досягнення цих цілей. На основі визначених бізнес-цілей і вимог формулюються конкретні завдання для аналізу даних. Це може включати збір необхідних даних, визначення методів аналізу та моделювання, що будуть використовуватися для досягнення бізнес-цілей.

Блок Business Understanding у моделі CRISP-DM допомагає встановити фундамент для успішного виконання проєкту з аналізу даних, розуміючи та адаптуючи аналітичні завдання до конкретних потреб бізнесу. Це фундамент всього проєкту. Тобто, чим краще відбудеться Business Understanding, тим краще можна розібратися, а що насправді потрібно замовнику, клієнту, product manager. Якщо ми говоримо про конкретні топіки, які тут піднімаються, це: визначення бізнес-цілей, тобто ми намагаємося це зрозуміти; оцінка поточної ситуації, тобто ми намагаємося розібратися, що зроблено, що можна зробити, що було, що є, де дані, бази, які задачі зроблені; визначення цілей аналітики – спроба розібратися на поточну ітерацію CRISP-DM, що ми хочемо проаналізувати, тобто як нам просувати проєкт далі. Тобто ми собі визначаємо обмежений масштаб задач, обмежений масштаб проблем, які ми вирішимо зараз і ми хочемо зрозуміти, як це просувати далі. І ми собі ставимо якісь цілі саме на аналітику, яку ми будемо робити; підготовка плану проєкту: робимо два плани і ми повинні прописати на найближчу ітерацію CRISP-DM, що ми будемо робити, які кроки; ще важливо з цілей, які ми поставили, визначити бізнес-метрику – це повинно бути зрозумілим, добре інтерпретованим бізнесом. Це повинно чітко проєктуватися із навчених алгоритмів на бізнес. Мова Python зручна для досліджень. У Python є багато інструментів для швидкого ітерування гіпотез, тобто ми легко можемо навчити алгоритм, відправити його у хмару і тестувати його на реальних користувачах.

У нас вже є задачі, є дані, вже все підготовлено, потрібно тільки брати та робити. Але у реальному житті все не так. У реальному житті до нас приходять з дуже абстрактною задачею з області і нам потрібно зрозуміти, як все зробити. Все це відбувається на етапі Business Understanding.

Наступний етап - *Data Understanding*. У моделі CRISP-DM блок Data Understanding має наступне призначення:

1) *Збір початкових даних*: на цьому етапі здійснюється збір початкових даних, які потенційно можуть бути використані для аналізу. Це включає ідентифікацію джерел даних, взаємодію з відповідними відділами або сторонніми постачальниками даних.

2) *Первинне розуміння даних*. Основна мета цього етапу - отримати загальне уявлення про структуру і вміст даних. Вивчення форматів, типів даних, обсягу, рівня якості та доступності даних.

3) *Первинний огляд даних* - виконання простих статистичних аналізів, включаючи розподіл значень, виявлення відсутності або аномалій у даних.

4) *Оцінка якості даних*. Аналіз наявних даних для визначення їхньої придатності для подальшого аналізу. Це включає оцінку повноти, достовірності, актуальності та зрозумілості даних.

5) Визначення потреб у підготовці даних - виявлення потреб в очищенні, перетворенні або об'єднанні даних для подальшого використання у моделюванні та аналізі.

Отже, блок Data Understanding у моделі CRISP-DM є першим кроком у розумінні і підготовці даних перед подальшим аналізом і моделюванням (рис. 1.2).

Ми подивилися наші дані і буває таке, що приходиться переосмислювати наші цілі і наші задачі. Якщо передбачити, що все добре, нам потрібно дані зібрати, описати дані (ми повинні дати їм характеристики як Data-аналітики) і ми дивимося дані нашими очима.

Рис. 1.2 Інструментарій для збору, опису та вивчення даних

У зборі, описанні та вивченні даних інструментарій у нас такий: Python; Pandas – дозволяє витягувати велику кількість статистик із даних, велику кількість інформації. Тобто все, що ми говоримо про математичну статистику – це все є Pandas; SQL – робота з базами даних. Якщо у нас дані не дуже добре структуровані, якщо це щось нереляційне, це може бути mongoDB. Є дві гарні

бібліотеки, які допомагають аналізом даних, допомагають з описом – це Matplotlib, Seaborn – бібліотеки для візуалізації даних. Картинками простіше спілкуватися з людьми із бізнесу й собі розуміти. Коли ми починаємо робити картинки, ми самі починаємо розуміти, що відбувається навколо.

Після того, як ми наші дані усвідомили, зробили аналітичну роботу, тепер ми ці дані хочемо підготувати - *Data Preparation*. У моделі CRISP-DM блок Data Preparation (підготовка даних) має наступне призначення:

1. *Вибір даних* - обрання підмножини даних, які будуть використовуватися для аналізу. Це може включати відбір необхідних змінних або записів, що мають значення для конкретного дослідження або задачі.

2. *Очищення даних* - очищення даних від аномалій, відсутніх значень, дублікатів, помилок у форматі або будь-яких інших проблем, які можуть вплинути на якість аналізу.

3. *Перетворення даних* - виконання необхідних перетворень даних, таких як масштабування, нормалізація, кодування категоріальних змінних, створення нових ознак на основі існуючих.

4. *Інтеграція даних* - об'єднання даних із різних джерел або джерел, обробка яких відбувалася окремо, у єдиний набір даних для подальшого аналізу.

5. *Форматування даних* - забезпечення відповідності структури даних вимогам моделі або інструментів, які будуть використовуватися для аналізу.

Отже, блок Data Preparation у моделі CRISP-DM є важливим етапом перед моделюванням, який спрямований на підготовку даних для відповідного і адекватного аналізу, забезпечуючи їх чистоту, цілісність та готовність до подальшого використання у моделях аналізу даних.

Підготувати дані означає їх очистити від шумів, зайвих даних. Буває так, що ми хочемо ці дані структурувати, переструктурувати (прибрати, добавити), змінити якісь властивості наших даних. Ще однією проблемою є пропуски даних, які бувають випадковими, спеціальними. Ці пропуски потрібно обробити: прибрати, заповнити середнім значенням, зробити медіаною, модою, тобто це все потрібно дивитися індивідуально по конкретному прикладу. Все потрібно

дивитися з точки зору візуалізації: поприбирали дані – подивилися з точки зору візуалізації, поструктурували дані – подивилися з точки зору візуалізації, заповнили пропуски – подивилися, як це виглядає з точки зору візуалізації. Це дуже важливо, коли ми говоримо про Data Preparation.

1.2 Етап моделювання даних

Після того, як ми визначеним чином підготували дані, переходимо на етап моделювання (рис. 1.3).

Рис. 1.3 Етап моделювання даних

Визначаємо, що важливо зробити на етапі моделювання. У моделі CRISP-DM блок Modelling (моделювання) забезпечує:

1. *Вибір моделей*. Визначення алгоритмів і моделей, які будуть використовуватися для розв'язання конкретної задачі або досягнення бізнес-цілей на основі аналізу даних.

2. *Побудову моделей***.** Розроблення і побудова різних варіантів моделей з використанням вибраних алгоритмів. Це може включати налаштування параметрів моделей, вибір оптимальних характеристик і функцій для включення в модель.

3. *Оцінку моделей*. Оцінка та порівняння ефективності різних моделей за допомогою метрик якості, які відповідають бізнес-цілям проєкту (наприклад, точність, чутливість, специфічність тощо).

4. *Вибір оптимальної моделі*. Вибір моделі або моделей, які найкраще відповідають вимогам бізнесу і досягають найкращих результатів за обраною метрикою оцінки.

5. *Валідацію моделі.* Перевірка стабільності та надійності обраної моделі на тестових даних, які не використовувалися під час тренування моделі.

Отже, блок Modelling у моделі CRISP-DM зосереджений на розробці та оцінці моделей аналізу даних, що дозволяє знайти найефективніше рішення для вирішення бізнес-проблеми на основі зібраних та підготовлених даних.

Ми повинні вибрати алгоритми, які нам підходять під нашу задачу, так як у нас вже є задача, у нас є дані. Після вибору алгоритмів, ми повинні методом наукового перебору вибрати з цього всього те, що буде працювати найкраще. Тут у нас буде дуже багато навчання і в рамках навчання є можна використовувати бібліотеку *mlflow*, яка дозволяє відслідковувати експерименти, дозволяє нам ними працювати. Тобто у нас уже будуть табличні дані по експериментам і в кінцевому результаті ми зможемо навчити якийсь алгоритм, який буде нам передбачати, як ми будемо проводити експерименти.

Коли ми говоримо про моделювання, якщо ми говоримо про ML – це бібліотека *Scikit Learn*. Якщо ми говоримо про глибоке навчання, то це фреймворки *PyTorch* або *TensorFlow*. Інструмент *TensorFlow* на сьогоднішній день доводить модель до production.

Оцінка якості моделі. Коли ми говоримо про Business Understanding, ми закладаємо бізнес-метрику. Це з точки зору бізнесу. На етапі моделювання ми хочемо оцінювати модель конкретно по різним метрикам, наприклад, порівняти по функціям втрат, порівняти по зсувам, по неправильності передбачення у одного чи іншого класу. Треба визначити, які ризики для нас несе кожна модель потенційно. Це те, що ми можемо вибрати на етапі моделювання і вибрати вже пул моделей, які перенесемо далі.

Наступний етап - *Evaluation*. Даний етап майже завжди завершає цикл, так як не завжди виходить перейти на рівень *Deployment*. Іноді буває, коли зробили Evaluation розуміємо, що треба було не так. Це приходить з напрацьованим досвідом по визначеній задачі. Тому краще не в кращому стані прогнати один етап CRISP-DM для того, щоб зробити гарний Business Understanding.

У моделі CRISP-DM блок Evaluation (оцінка) має наступне призначення:

1. *Оцінка результатів моделі* - проведення детального аналізу результатів розроблених моделей на основі тестових даних. Це включає оцінку точності, чутливості, специфічності та інших метрик, які визначають ефективність моделі.

2. *Порівняння з бізнес-вимогами* - порівняння досягнутих результатів з вимогами бізнесу та поставленими завданнями з моделювання. Це допомагає визначити, наскільки модель відповідає потребам бізнесу і чи варто її впроваджувати.

3. *Оцінка стійкості моделі* - визначення стабільності та надійності моделі під різними умовами або на різних даних. Це може включати аналіз чутливості моделі до змін у вхідних даних або параметрах.

4. *Документація результатів* - формування звіту або презентації з описом виявлених результатів, рекомендацій щодо використання моделі і можливих покращень.

5. *Підготовка до впровадження* - оцінка готовності моделі до впровадження в реальне виробниче середовище або використання для рішень бізнесу.

Отже, блок Evaluation у моделі CRISP-DM є ключовим етапом, який дозволяє зрозуміти, наскільки ефективна і придатна є модель для досягнення поставлених бізнес-цілей і розв'язання проблеми.

Маємо сказати про оцінку результатів моделювання, про оцінку процесу, як працюють наші алгоритми. Оцінка процесу дає зрозуміти, чи правильно ми робили Business Understanding, Data Understanding. Тобто на даному етапі ми дивимося те, що ми робили валідно чи потрібно робити по іншому. Після Evaluation Business Understanding виконується відмінно.

У моделі CRISP-DM блок *Deployment* (впровадження) має наступне призначення:

1. *Реалізація моделі* - перенесення розробленої моделі з відповідними алгоритмами, параметрами і структурами даних у виробниче середовище або систему, де вона буде використовуватися для прийняття реальних рішень.

2. *Інтеграція з існуючими системами* - забезпечення взаємодії моделі з існуючими бізнес-системами або інфраструктурою, що забезпечує її роботу.

3. *Тестування впровадження* - проведення тестів для перевірки правильності та стабільності роботи моделі в реальному виробничому середовищі перед її фактичним впровадженням.

4. *Моніторинг та підтримка* - забезпечення моніторингу роботи моделі після впровадження для виявлення відхилень в її роботі і прийняття вчасних заходів для їх виправлення. Це включає підтримку моделі, оновлення її при необхідності та виявлення нових можливостей для покращення.

5. *Навчання користувачів* - підготовка та навчання персоналу, який буде використовувати модель, щоб забезпечити її ефективне використання і розуміння особливостей її роботи.

Отже, блок *Deployment* у моделі CRISP-DM фокусується на переведенні моделі з фази тестування і оцінки у реальне використання, забезпечуючи її ефективне і надійне функціонування у виробничому середовищі (рис. 1.4).

Рис. 1.4 Інструменти Deployment

Перша задача – впровадити рішення, залити у хмару. Перед тим як завершити етап Deployment, потрібно впровадити моніторинг і розуміти як це підтримувати.

Ми повинні розуміти, що в якийсь момент наш алгоритм почне працювати погано і ми повинні отримати сповіщення, що все пішло не так, як ми цього хочемо. В результаті мають бути якісь звіти, які надсилаються після того, як наша система відпрацює. Насправді моніторинг, підтримка, звіти – це все дуже важливо. Тепер стосовно інструментарію. Якщо ми говоримо про Python, хмарні середовища, про мікросервісну архітектуру, то зазвичай це такий стек: є додаток, написаний на Flask – це веб-фреймворк. Flask завертається у нас у docker – засіб контейнеризації, тобто коли ми завертаємо наш додаток у незалежний контейнер і далі вже цей контейнер ми відправляємо у хмару – Google, Amazone, Microsoft – з ким у нас відносини в рамках нашої компанії, клієнта, туди і йдемо.

Моніторинг – це не тільки якість роботи моделі, але й навантаження на всю систему. Наприклад, ми такі моделі навчили, що тепер потрібно ставити в 10 разів більше комп'ютерів і відповідно це невигідно бізнесу. Й тому прийдеться щось змінювати в плані моделювання. Іноді треба написати код моделі або змінити й для цього потрібно розбиратися у статистиці. Потрібно розібратися, який є інструментарій для табличних даних (він вже стабілізувався). А картинки, звук, текст – проєкти над якими потрібно багато працювати.

Таблиця 1.1

Функції етапів методології CRISP-DM

№ п/п	Етап методології CRISP-DM	Функції
1	Бізнес-розуміння (Business Understanding)	Визначення бізнес цілей, оцінка поточної ситуації, визначення цілей аналітики, підготовка плану проєкту, визначення бізнес метрики
2	Розуміння даних (Data Understanding)	Збір даних, опис даних, вивчення даних
3	Підготовка даних (Data Preparation)	Очищення даних, структурування даних, заповнення пропусків
4	Моделювання (Modeling)	Вибір алгоритму, навчання моделей, оцінка якості моделей
5	Оцінка (Evaluation)	Оцінка результатів, оцінка процесу, визначення наступних кроків
6	Розгортання (Deployment)	Впровадження, планування моніторингу та підтримки, підготовка звіту

1.3 Визначення ролі датасетів у ML та DL

Датасет - це набір даних, який використовується для аналізу, моделювання та обробки інформації. Він є основним елементом для проведення досліджень, навчання моделей машинного навчання та інших аналітичних задач. Основні характеристики датасету: структура, тип даних, розмір, якість, джерела, призначення. Датасет зазвичай організований у вигляді таблиці, де рядки представляють окремі записи (об’єкти, приклади, спостереження), а стовпці — різні атрибути або характеристики цих записів. Датасет може містити різні типи даних, такі як числові, текстові, категоріальні, часові ряди тощо. Кількість записів (рядків) та атрибутів (стовпців) може варіюватися від кількох десятків до мільйонів і більше. Якість датасету визначається його повнотою, точністю, відсутністю помилок або пропущених значень. Датасети можуть бути зібрані з різних джерел, таких як бази даних, веб-скрапінг, опитування, сенсори, транзакційні системи тощо. Датасети можуть бути призначені для різних цілей, включаючи навчання, тестування та валідацію моделей машинного навчання, а також для проведення статистичних аналізів.

Датасети використовуються для навчання моделей ML, наприклад, для навчання алгоритмів з метою передбачення або класифікації нових даних; для аналізу даних: дослідження патернів, трендів та аномалій у даних; валідації гіпотез: перевірка наукових гіпотез за допомогою статистичного аналізу.

На сьогоднішній день відомими датасетами є:

- **Iris** - класичний датасет для класифікації, містить інформацію про різні види ірисів;
- **MNIST** - датасет зображень рукописних цифр, часто використовується для навчання і тестування моделей розпізнавання образів;
- **CIFAR-10** - датасет зображень для задач класифікації об'єктів;
- **Titanic** - датасет з інформацією про пасажирів "Титаніка", використовується для навчання моделей виживання.

Усі ці характеристики та приклади допомагають зрозуміти, як важливі датасети для науки про дані і яку роль вони відіграють у процесах аналізу та моделювання.

Існує багато ресурсів, де можна завантажити датасети для наукових досліджень, аналізу даних та навчання моделей машинного навчання. Найпопулярнішими та найзручнішими платформами є:

- **Kaggle** (Kaggle Datasets) - платформа для змагань у сфері науки про дані, де розміщено багато різноманітних датасетів;
- **UCI Machine Learning Repository** (UCI ML Repository) - одне з найвідоміших сховищ датасетів для машинного навчання та наукових досліджень;
- **Google Dataset Search** - пошуковий інструмент від Google для знаходження датасетів у мережі;
- **Amazon Web Services (AWS) Open Data** - датасети, розміщені на платформі AWS, доступні для аналізу та обробки;
- **Data.gov** - портал відкритих даних уряду США, що містить тисячі датасетів з різних галузей;
- **Harvard Dataverse** - платформа для зберігання та обміну науковими даними, що підтримується Гарвардським університетом;
- **Awesome Public Datasets** - список публічних датасетів на GitHub, який спільно підтримується;
- **Microsoft Azure Open Datasets** (Azure Open Datasets) - датасети, доступні на платформі Microsoft Azure;
- **StatLib** - бібліотека статистичних даних від Університету Карнегі-Меллона;
- **OpenML** - платформа для обміну, створення та дослідження датасетів для машинного навчання.

Ці ресурси пропонують широкий спектр датасетів, які можуть бути використані для різних цілей, включаючи навчання моделей машинного навчання, проведення аналітичних досліджень та валідацію гіпотез.

Таблиця 1.2

Типи даних

Табличні дані	**Зображення**	**Текст**	**Звук**
Оцінка кредитоздатності	Виявлення дефектів на виробництві	Виявлення спаму	Ідентифікація по голосу
Ймовірність переходу на інший тарифний план	Самокеровані автомобілі	Перекладачі	Переведення голосу в текст
Контекстна реклама	Ідентифікація по обличчю	Автодоповнення	Видалення шумів

Набір даних або датасет – це колекція даних, яка стосується певної теми або галузі. Набір даних включає різні типи інформації: текст, зображення, відео, аудіо і може зберігатися у різних форматах, наприклад, CSV, JSON або SQL. Таким чином, набір даних зазвичай включає структуровані дані для певної мети. Ми можемо використовувати набори даних для проведення маркетингових досліджень, аналізу конкурентів, порівняння цін, визначення та вивчення тенденцій або навчання моделей ML. Це лише кілька прикладів. Набори даних корисні у різних галузях та ситуаціях. Набори даних можна класифікувати декількома способами. Розглянемо деякі найбільш важливі типи наборів даних (рис. 1.5).

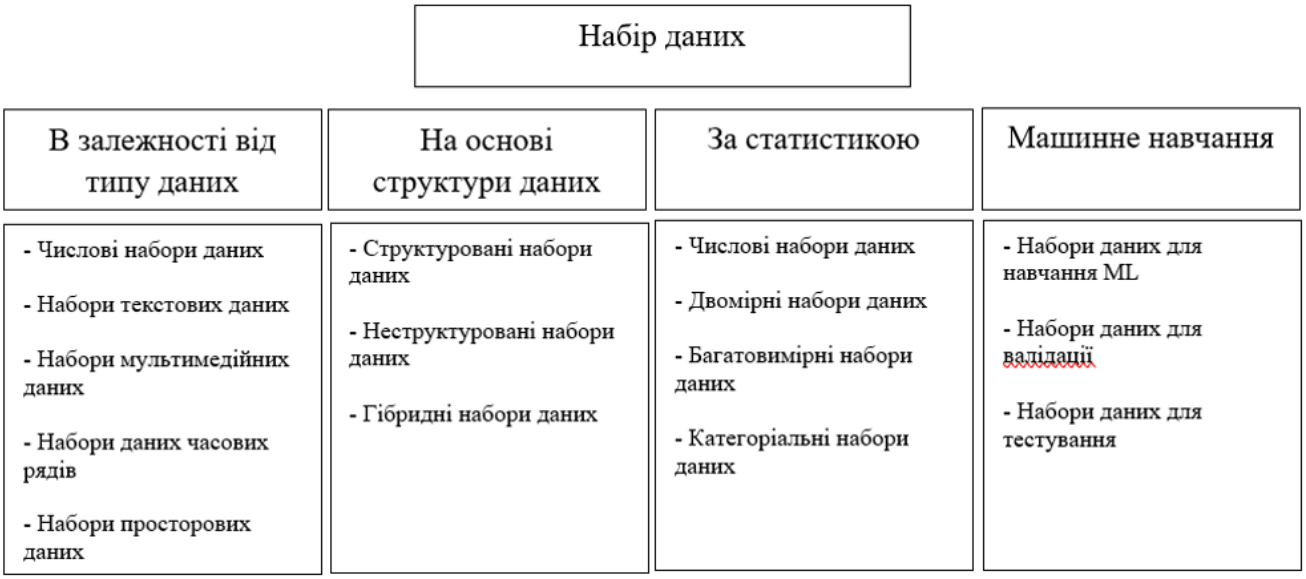


Рис. 1.5 Класифікація набору даних

Приклад набору даних приведений на рис. 1.6.

	A	B	C	D	E	F	G
1	Date	Average price in USD	Total Sold	Smal Avocados Sold	Large Avocados Sold	Extra Large Avocados Sold	City
2	12/27/2022	1.29	319746	292097	27351	298	Atlanta
3	12/20/2022	1.38	272580	251774	20702	103	Atlanta
4	12/13/2022	1.26	356227	324932	31019	276	Atlanta
5	12/6/2022	1.37	306947	283024	23741	182	Atlanta
6	11/29/2022	1.29	279486	250289	28890	308	Atlanta
7	11/22/2022	1.3	300032	269799	29732	501	Atlanta
8	11/15/2022	1.43	292814	263916	28442	456	Atlanta
9	11/8/2022	1.42	285413	250441	34483	488	Atlanta
10	11/1/2022	1.29	353690	290458	62980	253	Atlanta
11	10/25/2022	1.39	301727	236814	64608	304	Atlanta
12	10/18/2022	1.39	288733	228710	59732	291	Atlanta
13	10/11/2022	1.25	347580	255933	91047	600	Atlanta
14	10/4/2022	1.26	359222	265797	92780	644	Atlanta
15	9/27/2022	1.07	324659	262107	61871	681	Atlanta

Рис. 1.6 Приклад набору даних (avocado_prices.xlsx)

Числові набори даних містять числа і використовуються для кількісного аналізу. Набори текстових даних містять пости, текстові повідомлення та документи. Набори мультимедійних даних містять зображення, відео та аудіофайли. Набори даних часових рядів містять дані, які зібрані за визначений період часу для аналізу тенденцій та закономірностей. Набори просторових даних містять інформацію з географічною прив’язкою, наприклад, дані GPS.

На основі структури даних набори даних поділяються на структуровані, неструктуровані, гібридні. Структуровані набори даних організовані у визначені структури, щоб спростити запит та аналіз даних. Неструктуровані набори даних не мають чітко визначеної схеми. Вони можуть включати в себе різні типи даних. Гібридні набори даних включають як структуровані так і неструктуровані дані.

Числові набори даних (або числові дані) – це сукупності даних, де всі елементи є числами. Такі дані можуть бути використані для різних видів аналізу, включаючи статистичний аналіз, математичні обчислення, моделювання, машинне навчання та інші методи обробки даних. Двомірні набори даних (або двовимірні дані) – це набори даних, які мають дві виміри

або дві осі. Найпоширеніший приклад двомірних наборів даних – це таблиця, де дані організовані в рядки та стовпці. Кожен елемент даних у такій таблиці можна ідентифікувати за двома координатами: номером рядка та номером стовпця. Основними характеристиками двомірних наборів даних є: рядки та стовпці, матриці, координати, можливість різних типів даних (у таблицях різні стовпці можуть містити різні типи даних (числа, текст, дати тощо)). Двомірні набори даних широко використовуються у різних галузях для організації, аналізу та візуалізації інформації.

Багатовимірні набори даних (або багатовимірні дані) – це дані, які мають більше, ніж дві виміри. У таких наборах кожен елемент визначається кількома координатами або атрибутами. Ці дані зазвичай організовані в багатовимірні масиви або таблиці з багатьма стовпцями, де кожен стовпець представляє собою окремий вимір (або атрибут). Категоріальні набори даних (або категорійні дані) – це тип даних, які можуть приймати одне з кількох дискретних значень або категорій. Ці дані не мають числового значення або порядку і використовуються для класифікації об'єктів у групи або категорії. Категоріальні дані часто використовуються в статистиці, аналізі даних і машинному навчанні для класифікації, аналізу та прогнозування. Категоріальні дані допомагають класифікувати та аналізувати різні аспекти інформації, що особливо корисно в галузях, де важлива класифікація та групування об'єктів.

Набори даних для навчання машинного навчання (ML) – це структуровані дані, які використовуються для тренування, тестування та оцінки моделей машинного навчання. Такі набори даних можуть включати різні типи даних, включаючи числові, категоріальні, текстові, зображення та інші. Основною метою використання цих наборів є створення моделей, які можуть робити прогнози або класифікувати нові, невідомі дані на основі навчання на відомих прикладах.

Набори даних для валідації – це спеціально виділені підмножини даних, які використовуються для оцінки і налаштування моделей машинного навчання

під час процесу їхнього навчання. Основна мета валідаційних наборів даних – допомогти вибрати найкращу модель або налаштування гіперпараметрів, уникнути перенавчання (overfitting) і забезпечити, щоб модель добре узагальнювала нові дані.

Набори даних для тестування (або тестові набори даних) – це спеціально виділені підмножини даних, які використовуються для оцінки продуктивності моделі машинного навчання після її тренування. Вони відіграють критичну роль у перевірці того, наскільки добре модель узагальнює нові дані.

Датасети відіграють ключову роль у машинному та глибокому навчанні, оскільки вони слугують основою для навчання моделей. Датасети використовуються для тренування моделей. Чим більше даних, тим краще модель може навчитися розпізнавати закономірності та робити точні прогнози. Наприклад, для розпізнавання зображень необхідно мати великий набір зображень для того, щоб модель могла навчитися розрізняти різні об'єкти.

Після навчання модель необхідно перевірити на нових даних, яких вона раніше не бачила. Це допомагає оцінити, наскільки добре модель узагальнює свої знання на нові випадки. Для цього використовуються валідаційні та тестові датасети. Для оцінки продуктивності моделі використовуються метрики, які розраховуються на основі тестового датасету. Це дозволяє визначити, наскільки точні або надійні прогнози моделі.

Аналіз помилок, які модель робить на валідаційному або тестовому датасеті, дозволяє виявити слабкі місця моделі та покращити її шляхом внесення змін у архітектуру або дані. Для того, щоб модель була загальнозастосовною, датасет повинен бути репрезентативним та включати різноманітні приклади. Це дозволяє моделі бути стійкою до різних варіантів даних у реальному світі.

Якість і різноманіття датасету безпосередньо впливають на успішність машинного та глибокого навчання.

2 ДОСЛІДЖЕННЯ ОСОБЛИВОСТЕЙ МАШИННОГО ТА ГЛИБОКОГО НАВЧАННЯ

2.1 ML/DL – цілі та принципи роботи

Машинне навчання (ML) та глибоке навчання (DL) є двома підгалузями штучного інтелекту (AI), але вони мають свої особливості та відмінності (рис.2.1).

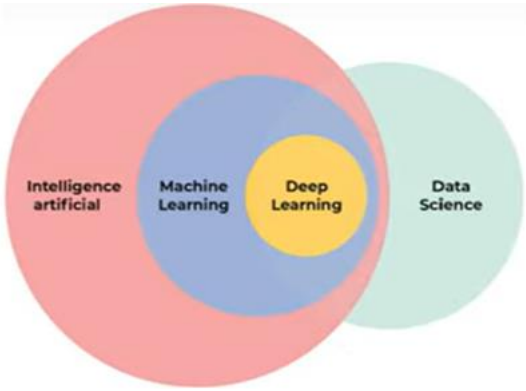


Рис. 2.1 ML/DL

Машинне навчання — це область штучного інтелекту, яка зосереджується на розробці алгоритмів, здатних навчатися на основі даних та робити прогнози або приймати рішення без прямого програмування. Основними методами ML є лінійна регресія, логістична регресія, дерева рішень, метод опорних векторів (SVM), кластеризація (наприклад, k-середні), ансамблеві методи (наприклад, Random Forest, Gradient Boosting). Підходи до навчання:

1) **Навчання з учителем -** використовується набір міток для навчання моделі (наприклад, класифікація, регресія);

2) **Навчання без вчителя -** моделі вивчають структуру даних без використання міток (наприклад, кластеризація, асоціативне навчання);

3) **Навчання з підкріпленням -** модель навчається через взаємодію із середовищем та отримання нагород або штрафів (наприклад, ігри, робототехніка).

Глибоке навчання — це підгалузь машинного навчання, яка використовує багатошарові нейронні мережі для моделювання складних патернів у даних. Основними методами DL є глибокі нейронні мережі (DNN), конволюційні нейронні мережі (CNN) для обробки зображень, рекурентні нейронні мережі

(RNN) для послідовних даних (наприклад, текст, час), архітектури, такі як LSTM і GRU для покращення RNN, генеративно-змагальні мережі (GAN).

Глибокі мережі можуть автоматично витягувати важливі характеристики з сирих даних, що зменшує необхідність ручної інженерії ознак. Глибоке навчання вимагає великих обсягів даних для ефективного навчання. Глибокі моделі потребують значних обчислювальних ресурсів (наприклад, GPU) для навчання. У табл. 2.1 приведено порівняльну характеристику ML та DL.

Таблиця 2.1

Порівняльна характеристика ML та DL

	Machine Learning	**Deep Learning**
Об'єм даних	Можна використати невеликі обсяги даних	Необхідні великі обсяги даних
Потреба в ресурсах	Не вимагає значних обчислювальних ресурсів	Потребує значних обчислювальних ресурсів
Конструювання ознак	Працює з точно визначеними ознаками (features)	Може самостійно розпізнавати та створювати ознаки
Підхід до навчання	Дискретний (навчання розбивається на окремі кроки)	Задача вирішується «наскрізним методом»
Час навчання	Відносно коротке навчання	Довге навчання

У табл. 2.2 відображено практичні сфери застосування ML, DL.

Таблиця 2.2

Практичні сфери застосування ML, DL

№ п/п	Сфери застосування ML	Сфери застосування DL
1	Розпізнавання мови	Прогнозування часових рядів
2	Розпізнавання жестів	Ранжування в пошукових системах
3	Розпізнавання текстів	Виявлення спаму
4	Розпізнавання образів	Категоризація документів
5	Біоінформатичні розрахунки	Біржовий технічний аналіз
6	Медична діагностика	Кредитний скоринг
7	Виявлення шахрайства	Прогнозування втрати клієнтів

Основні відмінності ML та DL полягають у *складності моделей*, так як ML може використовувати простіші моделі та алгоритми, а DL використовує складні багатошарові нейронні мережі; в *обсягах даних*: ML може працювати з меншими обсягами даних, у той час як DL вимагає великих обсягів даних для досягнення хороших результатів; *інженерія ознак*: у ML часто потрібна значна ручна інженерія ознак, у DL моделі автоматично виділяють ознаки; в обчислювальних ресурсах - ML потребує менше обчислювальних ресурсів, DL вимагає значних обчислювальних ресурсів.

Глибоке навчання є більш спеціалізованою та потужною формою машинного навчання, яка добре підходить для задач, що вимагають обробки великих обсягів даних та складних патернів.

Штучний інтелект (Artificial intelligence, AI) – це інтелект, який демонструється машинами, на відміну від звичайного інтелекту, який притаманний тваринам та людям. Цей інтелект притаманний машинам, алгоритмам, а не живим істотам. Це деяка подібність «розуму», яка вміє вирішувати задачі, які були поставлені розробниками.

Машинне навчання (Machine learning, ML) – це комп'ютерні алгоритми, які можуть автоматично покращуватися завдяки досвіду та використанню даних.

Алгоритми машинного навчання будують модель на основі вибіркових даних, які відомі як дані для навчання, для того, щоб робити прогнози або приймати рішення, не будучи виражено запрограмованими для цього. Алгоритми машинного навчання використовують дані, щоб знайти в них закономірності, а потім вирішують задачу самостійно без програміста.

Наприклад, ми захотіли навчити модель на класифікацію зображень (навчання із вчителем). Наприклад: на картинці голуб або кіт (рис. 2.2).

Рис. 2.2 Класифікація зображень

Наприклад, є картинка кота і *цільова мітка* кіт, є картинка голубу і *цільова мітка* голуба. Ці цільові мітки були отримані за допомогою спеціально навчених людей: розмітників (асесорів), які дивилися на наші дані і виставляли мітку класу. Навчання із вчителем полягає в навчанні моделі на розмічених прикладах.

Задачі, які вирішуються в області навчання із вчителем (їх 2 види): *класифікація* – коли модель намагається побудувати розділяючу площину між класами (наприклад, розділити клас плюсів від класу кіл) та регресія (рис. 2.3).

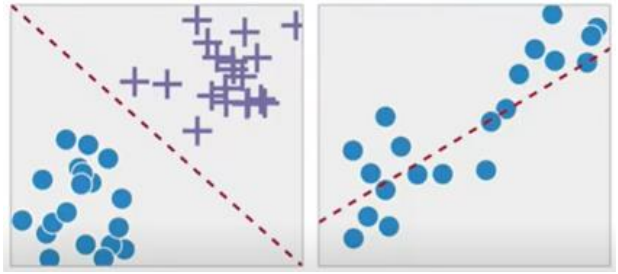


Рис. 2.3 Класифікація та регресія

У випадку класифікації цільове значення – дискретне. У випадку регресії цільове значення – дійсне. Дискретне і дійсне значення відносяться до типу даних.

Задача регресії – задача, де модель будує лінію, яка повторює закон, якому слідують наші дані.

Проаналізуємо приклади задач. Оскільки цільове значення у класифікації дискретне, категоріальне, то сюди підходять будь-які задачі, де ми можемо передбачити категорію об’єкту. Перша задача – класифікація зображень. Тут на картинках можуть бути числа від 0 до 9 і тому маємо класифікацію на 10 класів. Це задача багатокласової класифікації (рис. 2.4).

Рис. 2.4 Багатокласова класифікація

Друга задача – передбачення відтоку клієнтів. Клієнт або з нами, або вже не з нами. Тільки 2 стани – задача бінарної класифікації. Тільки на 2 класи (рис. 2.5).

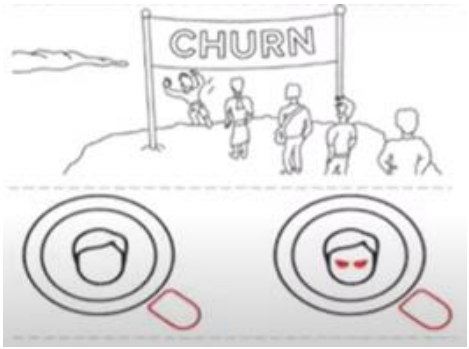


Рис. 2.5 Бінарна класифікація

Третя задача – знаходження шахрайських транзакцій: транзакція або шахрайська, або стандартна. Знову маємо бінарну класифікацію.

Четверта задача – кредитний скорінг – приймаємо рішення за допомогою моделі - кому видаємо кредит, а кому не видаємо, так як людина не добросовісна на думку моделі. Це також задача бінарної класифікації.

Переходимо до задач регресії – передбачення дійсного цільового значення. Перша задача – передбачення вартості нерухомості. Цікаво дізнатися, яку об’єктивну вартість можна поставити будинку за допомогою моделі ML. Друга задача – передбачення прибутку. Зазвичай це відноситься до торгових точок, в яких ми прогнозуємо потенційний прибуток на період, щоб знати як можна використовувати ці ресурси (рис. 2.6).

Рис. 2.6 Регресія

Третій приклад – передбачення продажів або попиту на товари. Дані задачі можуть бути корисними, коли ми хочемо спрогнозувати поставку товарів. Особливо актуально для продуктів, які швидко псуються.

Навчання без вчителя. Ці підходи основані на знаходженні закономірностей всередині даних (рис. 2.7).

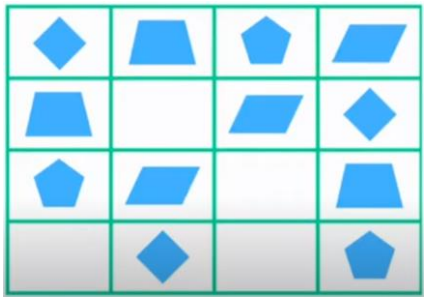


Рис. 2.7 Закономірності всередині даних

Моделі самостійно без підказок вчителя (розробника або асесора) намагаються зрозуміти суть задачі. У даному типі навчання у нас немає розмітки. Ми навіть не знаємо, що нам буде видавати модель ML. Але за допомогою моделі ML без вчителя ми можемо глибше зрозуміти влаштування наших даних. Приведемо приклад з котами та голубами для задачі без вчителя (рис. 2.8).

Рис. 2.8 Дані для кластеризації зображень

У нас є багато картинок котів і голубів, і ми їх хочемо відсортувати. Можемо скористатися хорошою переробкою зображень, а потім навчити модель кластеризації на сегментацію даних. Після того, як наша модель навчиться, ми зможемо отримати групи схожих зображень. В одній групі будуть фото котів, а в іншій групі будуть зображення голубів.

При цьому дані групи будуть називатися не як клас кіт і клас голуб, а як кластер 1 і кластер 2, так як модель нічого не знає про класи наших об’єктів. Вона тільки зрозуміла, що картинка кота більше схожа на іншу картинку кота, ніж на картинку голуба. Це перша задача з області навчання без вчителя – *кластеризація* (рис. 2.9).

Рис. 2.9 Кластеризація зображень

Інша задача називається *зниження розмірності*. Наші дані, на яких навчаємо моделі, дуже багатовимірні. Навіть наш набір даних у задачах класифікації по передбаченню відтоку клієнтів для операторів мобільного зв'язку може мати вигляд як представлено у табл. 2.3.

Таблиця 2.3

Набір даних у задачі класифікації

	Client	Gender	Minutes	SMS	Internet
0	0	0	10	3	3
1	1	1	0	100	600
2	2	0	400	230	20
3	3	0	30	25	400
4	4	1	300	60	50

По рядкам – клієнти. По стовбцям – характеристики, які описують наших клієнтів. Є унікальний ідентифікатор клієнта, є його належність до статі, є ознака кількості хвилин, які розмовляла людина, кількість SMS відправлених з номеру, кількість гігабайт, витрачених на користуванням Інтернетом. Можна знайти ще багато характеристик по нашим користувачам, але навіть на цьому прикладі маємо 5 ознак, 5 розмірностей і значить візуалізувати їх не зможемо. Якщо візьмемо тільки 2 ознаки – id-клієнта і його стать, то отримаємо візуалізацію на площині: id-клієнта по осі абсцис і стать клієнта – по осі ординат.

Якщо візьмемо 3 ознаки: id і стать клієнта, а також хвилини, то отримаємо візуалізацію у трьохвимірному просторі: id-клієнта по осі абсцис, стать клієнта по осі ординат, кількість хвилин – по третій осі.

Якщо візьмемо 4 ознаки (ті самі + кількість SMS), то отримаємо візуалізацію у чотирьохвимірному просторі (рис. 2.10).

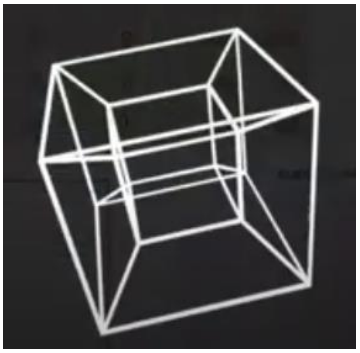


Рис. 2.10 Чотирьохвимірний простір

Але в силу фізіологічних особливостей будови наший очей і мозку ми не зможемо сприймати такий простір, а значить не зможемо проаналізувати наші дані. Але у нас є модель зниження розмірності, яка допоможе нам стиснути дані з більшою розмірністю.

Моделі зменшення розмірності можуть стискати наші дані і отримувати їх основну інформацію у просторі меншої розмірності. Зробимо те саме з нашими п’ятивимірними даними.

Стиснемо розмірність до двох компонентів за допомогою моделі зменшення розмірності (табл. 2.4).

Таблиця 2.4

Зменшення розмірності

	Component_1	Component_2
0	-0.150934	0.767394
1	-0.792986	-0.415525
2	0.955911	0.087406
3	-0.293932	0.301077
4	0.281941	-0.740352

Тепер можемо візуалізувати наших клієнтів на площині і повивчати.

Проаналізуємо декілька задач в області без вчителя. Спочатку кластеризація. Найпростіший приклад взяти дані по клієнтам магазину і сегментувати, отримавши тим самим групи клієнтів (рис. 2.11).

Рис. 2.11 Кластеризація

Якщо клієнти опинилися в одній групі, значить вони чимось схожі між собою, а значить це можемо використати у системі рекомендації і пропонувати людям купувати однакові товари. Другий приклад – кластеризація продуктів харчування. Можемо розбити на сегменти із схожою їжею і, наприклад, більш оптимально розміщувати їх в меню.

У задачах зниження розмірності всі приклади відбуваються по схожій схемі. Спочатку беремо дані, пропускаємо їх через модель зменшення розмірності, отримуємо стиснуте представлення. Далі візуалізуємо дані. Потім аналізуємо дані. Практично не важливо які дані, яка задача – черговість дій однакова.

Наступна область – навчання з підкріпленням. Цей підхід оснований на позитивному і негативному підкріпленнях. Є деякий агент, який існує в середовищі оточення і з ним він взаємодіє в процесі навчання (рис. 2.12).

Рис. 2.12 Навчання з підкріпленням

Агент отримує нагороди за хороші дії, які наближають його до кінцевої мети. Дана мета є в голові тільки у розробника. І агент отримує покарання за погані дії, які не наближують його до кінцевої мети. Агент не знає, що в нього за мета, він може тільки намацати правильні дії, щоб отримати якомога більше балів. За правильну для розробника дію агента хвалять, а за погані на думку розробника дії – агента карають.

На основі вищепроведених досліджень можна зробити висновки, що у навчанні із вчителем є цільові значення. Ми зараніше виконали розмітку. При цьому цільова змінна може бути дискретною або дійсною. Відповідно – це задачі класифікації або регресії.

У навчанні без вчителя – цільових значень немає. За допомогою цих задач ми можемо ще краще дізнатися про склад наших даних та внутрішні закономірності. Це кластеризація і зменшення розмірності.

У навчанні з підкріпленням є агенти, які навчаються виконувати поставлене завдання, але не напряму, а за допомогою позитивного та негативного підкріплення.

2.2 Дослідження особливостей побудови нейронних мереж

Нейронна мережа — це обчислювальна модель, наділена структурою та функціями біологічного мозку, яка використовується для розв'язання різних задач у галузі штучного інтелекту. Вона складається з вузлів, які імітують нейрони, і з'єднань між ними, які імітують синапси. Основна мета нейронної мережі — навчитися розпізнавати складні закономірності у даних і використовувати ці знання для прогнозування або класифікації нових даних (рис. 2.13).

До елементів нейронної мережі відносяться ноди (вузли, нейрони), зв’язки між вузлами, рівні. Рівні нейронної мережі бувають вхідними, прихованими та вихідними (рис. 2.14).

Вхідні рівні (або вхідні шари) в нейронній мережі виконують функцію прийому початкових даних і передавання їх далі у мережу для обробки. Основні функції

вхідного шару включають: прийом вхідних даних, підготовку даних для прихованих шарів, визначення розмірності мережі.

Вхідний шар отримує початкові дані, які можуть бути різними залежно від завдання. Наприклад, для зображень це можуть бути піксельні значення, для тексту — векторні представлення слів, а для табличних даних — числові або категоріальні значення. Вхідний шар передає дані у приховані шари без будь-яких змін. Кожен вузол вхідного шару зазвичай відповідає одному вхідному параметру і просто передає його значення на наступний шар. Кількість нейронів у вхідному шарі відповідає кількості вхідних параметрів даних. Це визначає, скільки інформації буде передано на наступні шари для обробки.

Вхідний шар не виконує ніяких обчислень або трансформацій даних, його основна функція — це передавання вхідної інформації в мережу для подальшої обробки прихованими і вихідними шарами.

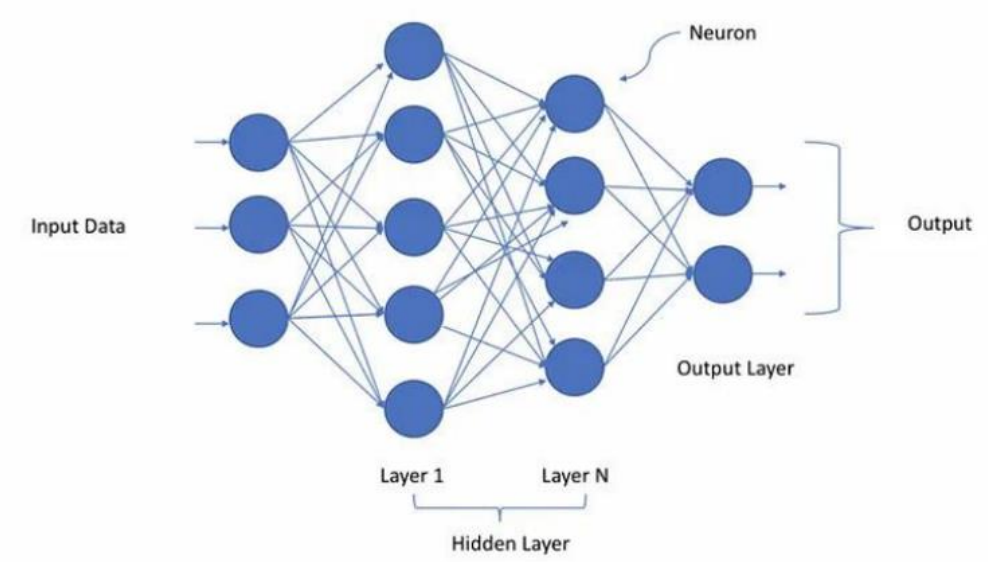


Рис. 2.13 Глибокі нейромережі в Deep Learning

Приховані рівні (шари) в нейронній мережі виконують основну обчислювальну роботу і відповідають за вивчення складних взаємозв'язків і закономірностей у даних. Основні функції прихованих рівнів включають: обчислення зважених сум вхідних сигналів, додавання зміщення (bias), застосування функції активації, передавання сигналів.

Кожен нейрон прихованого шару отримує сигнали від нейронів попереднього шару. Ці сигнали зважуються (множаться на вагові коефіцієнти) і сумуються. До зваженої суми додається зміщення (bias), яке допомагає моделі краще пристосовуватися до даних.

До обчисленої суми зі зміщенням застосовується активаційна функція. Ця функція додає нелінійність до моделі, що дозволяє нейронній мережі вивчати складні взаємозв'язки. Поширені активаційні функції включають ReLU (Rectified Linear Unit), сигмоїдну функцію, гіперболічний тангенс (tanh), тощо. Результат роботи активаційної функції передається далі на наступний шар (інший прихований шар або вихідний шар).

Приховані шари дозволяють нейронній мережі вивчати і представляти складні функції, які описують вхідні дані. Кількість і розмір прихованих шарів залежать від конкретного завдання і визначаються архітектурою мережі. Більша кількість прихованих шарів (глибина мережі) зазвичай дозволяє моделі краще визначати складні закономірності, але також вимагає більше даних для навчання і значні обчислювальні ресурси.

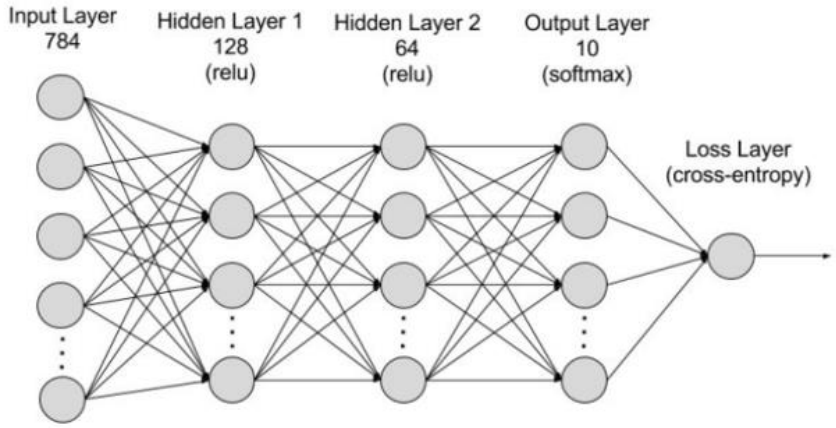


Рис. 2.14 Базова структура штучної нейронної мережі

Вихідні рівні (шари) в нейронній мережі виконують такі основні функції: обчислення вихідних значень, застосування активаційної функції, видавання результату.

Вихідний шар приймає сигнали від останнього прихованого шару, обробляє їх і виробляє кінцевий результат. Цей процес включає обчислення зважених сум вхідних сигналів і додавання зміщення, як і в прихованих шарах.

Вихідний шар зазвичай використовує функцію активації в залежності від типу задачі. Для задач класифікації з двома класами часто використовується сигмоїдна функція. Для задач багатокласової класифікації використовують функцію softmax, яка нормалізує вихідні значення, перетворюючи їх на ймовірності. Для задач регресії використовують лінійну функцію активації або не застосовують функцію активації взагалі (залишають лінійне поєднання вагових сум і зміщення). Після застосування функції активації вихідний шар видає кінцевий результат, який може бути використаний для прийняття рішень. У задачах класифікації результатом можуть бути ймовірності приналежності до кожного з класів. У задачах регресії результатом буде числове значення.

Таким чином, вихідний шар формує кінцевий результат нейронної мережі, на основі якого можна зробити передбачення або класифікацію вхідних даних.

Нейронні мережі відіграють важливу роль у сучасному світі, маючи значний вплив на різні сфери життя та технологій. Нейронні мережі використовуються для розпізнавання облич, обробки зображень, голосових помічників та систем автоматичного перекладу. Використовуються для діагностики захворювань, аналізу медичних зображень, прогнозування результатів лікування та розробки нових ліків. Допомагають в аналізі ринків, прогнозуванні курсів акцій, виявленні шахрайства та управлінні ризиками. Є ключовими для розвитку самокерованих автомобілів, забезпечуючи розпізнавання об'єктів, навігацію та прийняття рішень у режимі реального часу.

Актуальним є використання нейронних мереж для створення рекомендацій на основі поведінки користувачів, що застосовується в таких платформах, як Netflix, YouTube та Amazon. Нейронні мережі використовуються для обробки та аналізу великих обсягів даних, виявлення прихованих закономірностей та трендів. Розповсюджені у сферарах штучного інтелекту та автоматизації для розробки інтелектуальних систем, які можуть виконувати завдання, що вимагають людської

інтелектуальної діяльності, таких як чат-боти, віртуальні помічники та роботи. Поширеним є використання нейронних мереж для створення реалістичних ігор, анімацій, спецефектів у кіно та інтерактивних розваг. Адже нейронні мережі є важливим інструментом у розвитку сучасних технологій і мають великий потенціал для майбутніх інновацій.

2.3 Характеристика розповсюджених видів нейромереж

Нейромережі прямого розповсюдження (feed forward neural networks, FFNN) – прямолінійний вид нейромережі, в якому сусідні вузли шару пов'язані, а передавання інформації здійснюється безпосередньо від вхідного шару до вихідного (рис. 2.15). Мають відносно низьку функціональність, тому часто використовуються у комбінації з мережами інших видів.

Рис. 2.15 Нейромережі прямого розповсюдження

Нейромережі прямого розповсюдження мають особливості в архітектурі шарів, потоку даних, функціях активації, вагах. Вхідний шар - перший шар нейронів, який приймає вхідні дані. Кількість нейронів у цьому шарі відповідає розміру вхідного вектора. Приховані шари - один або більше шарів між вхідним та вихідним. Вони виконують більшість обчислень і перетворень даних. Вихідний шар - останній шар, який видає результат. Кількість нейронів у цьому шарі залежить від задачі (наприклад, кількість класів у задачі класифікації).

Дані проходять через мережу тільки в одному напрямку: від вхідного шару через приховані шари до вихідного шару. Зворотнього зв’язку немає.

Функції активації використовуються для введення нелінійностей у модель. Поширені функції активації включають сигмоїдальну функцію, ReLU (Rectified Linear Unit), tanh та інші.

Кожен зв’язок між нейронами має вагу, яка змінюється під час навчання для мінімізації функції втрат. Кожен нейрон зазвичай має зміщення, яке допомагає моделі краще підлаштовуватися під дані.

Вхідні дані проходять через мережу і обчислюється вихід. Функція втрат використовується для оцінки помилки між передбаченням і фактичними значеннями. Алгоритм, що використовується для оновлення ваг та зміщень у нейронних мережах з метою мінімізації функції втрат, називається алгоритмом зворотного розповсюдження помилки (backpropagation). Цей алгоритм, у поєднанні з методами оптимізації, такими як градієнтний спуск (gradient descent), забезпечує ефективне навчання нейронних мереж.

Гіперпараметрами є кількість шарів, кількість нейронів у кожному шарі, типи активаційних функцій, швидкість навчання та інші параметри, які визначають структуру та поведінку нейромережі.

Теорема універсальної апроксимації стверджує, що навіть одношарова (з одним прихованим шаром) FFNN з достатньою кількістю нейронів може апроксимувати будь-яку неперервну функцію на компактному просторі.

Ці особливості роблять нейромережі прямого розповсюдження потужними інструментами для розв'язання різноманітних задач у сфері машинного навчання та штучного інтелекту.

Нейронна мережа Хопфілда (Hopfield network, HN) — нейронна мережа із симетричною матрицею зв'язків (рис. 2.16). Під час отримання вхідних даних кожен вузол є вхідним нейроном, у процесі навчання стає прихованим, а потім стає вихідним. Значення нейронів встановлюються відповідно до бажаного шаблону, після чого обчислюються ваги, які надалі не змінюються. Результат завжди буде зводитися до одного з цих шаблонів.

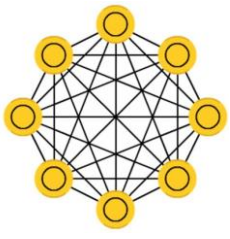


Рис. 2.16 Нейромережі Хопфілда

Нейромережі Хопфілда мають характерні особливості в архітектурі, динаміці роботи, які відрізняють їх від інших типів нейромереж. Нейромережа Хопфілда складається з одного шару нейронів, де кожен нейрон пов'язаний з усіма іншими нейронами (повнозв'язна мережа). Вага зв'язку між нейронами *i* та *j* рівна вазі зв'язку між нейронами *j* та *i* ($w_{ij}=w_{ji}$), що забезпечує симетричність матриці ваг. Нейрони оновлюються асинхронно, тобто один за одним, а не всі одночасно. Кожен нейрон може бути в одному з двох станів (зазвичай -1 або 1, або 0 та 1).

Нейромережа Хопфілда має енергетичну функцію, яка завжди зменшується при оновленні станів нейронів, що забезпечує збіжність до стабільного стану (мінімуму енергетичної функції). Енергетична функція визначається як:

$$E = -\frac{1}{2}\sum_{i \neq j} w_{ij} s_i s_j + \sum_i \theta_i s_i, \tag{2.1}$$

де S_i — стан нейрона *i*, w_{ij} - вага зв'язку між нейронами *i* та *j*, θ_i — поріг нейрона *i*.

Нейромережа Хопфілда може зберігати та відтворювати асоціативну пам'ять. Це означає, що вона здатна відновити повну картину (вектор станів) з часткової або пошкодженої інформації. Навчання в нейромережі Хопфілда часто здійснюється за допомогою правила Хебба, де ваги обчислюються як:

$$w_{ij} = \frac{1}{N}\sum_{\mu=1}^{P} \xi_i^\mu \xi_j^\mu, \tag{2.2}$$

де ξ_i^μ - значення *i*-го нейрона в μ-му патерні, *P* - кількість патернів, *N* - кількість нейронів.

Кількість патернів, які можна надійно зберігати, обмежена і становить приблизно $0.15N$, де *N* – кількість нейронів. Нейромережа може застрягати в локальних мінімумах енергетичної функції, що призводить до неправильного відновлення патернів.

Ці особливості роблять нейромережі Хопфілда корисними для задач асоціативної пам'яті та оптимізації, хоча вони мають певні обмеження щодо обсягу пам'яті та стабільності роботи.

Автоенкодер (autoencoder) – подібна до FFNN архітектура. Основною ідеєю є автоматичне кодування (стиснення) інформації (рис. 2.17). Сама мережа за формою нагадує симетричний пісочний годинник, у ній приховані шари менші за вхідний та вихідний. Автоенкодер можна навчати методом зворотнього поширення похибки, подаючи вхідні дані та задаючи похибку рівній різниці між входом та виходом нейромережі.

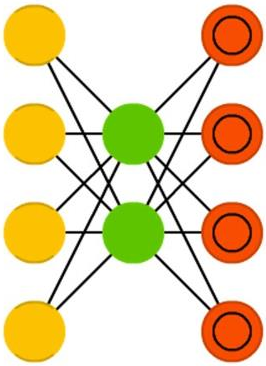


Рис. 2.17 Нейромережі автоенкодери

Кодувальник (encoder) - перша частина автоенкодера, яка зменшує розмірність вхідних даних і перетворює їх у більш компактне представлення (код). Bottleneck - шар з найменшою кількістю нейронів, розташований між кодувальником і декодувальником. Цей шар містить стиснуте представлення вхідних даних. Декодувальник (decoder) - друга частина автоенкодера, яка відновлює дані з компактного представлення до їх оригінального розміру.

Обробка даних у таких нейромережах відбувається наступним чином. Вхідні дані проходять через кодувальник, перетворюючись у стиснуте представлення, потім через декодувальник, де вони відновлюються до початкового вигляду. Мета навчання автоенкодера – зробити так, щоб відновлені дані якомога точніше відповідали вхідним даним. Автоенкодери використовують функцію втрат, яка вимірює різницю між вхідними даними та відновленими даними. Зазвичай це середньоквадратична похибка (Mean Squared Error, MSE).

У табл. 2.5 приведені найпоширеніші типи автоенкодерів.

Таблиця 2.5

Типи автоенкодерів

№ п/п	Назва	Особливості побудови
1	***Недосконалі (undercomplete)***	Мають bottleneck-шар меншого розміру, ніж вхідні дані, що змушує мережу вивчати важливіші ознаки даних.
2	***Перевершуючі (overcomplete)***	Мають bottleneck -шар більшого розміру, ніж вхідні дані, але використовують регуляризацію для запобігання перенавчанню.
3	***Спарс (sparse) автоенкодери***	Використовують регуляризацію для обмеження кількості активних нейронів у прихованих шарах.
4	***Шумозахищені (denoising)***	Навчаються відновлювати пошкоджені або зашумлені вхідні дані, що допомагає навчити мережу витягати важливі ознаки.
5	***Варіаційні (variational) автоенкодери (VAE)***	Використовують стохастичний підхід до генерації даних, що дозволяє створювати нові дані, схожі на тренувальні.

Автоенкодери можуть використовуватись для зменшення розмірності даних, зберігаючи важливі ознаки. Використовуються для очищення даних від шуму. Можуть виявляти аномальні зразки даних, які погано відновлюються автоенкодером. Варіаційні автоенкодери використовуються для генерації нових зразків даних. Навчання автоенкодера здійснюється за допомогою методів зворотнього поширення помилки (backpropagation) та градієнтного спуску (gradient descent).

Ці особливості роблять автоенкодери потужними інструментами для задач зменшення розмірності, очищення даних, виявлення аномалій та генерації нових даних.

Згорткові нейромережі (convolutional neural networks, CNN) в основному використовуються для обробки зображень (рис. 2.18). Такі мережі зазвичай використовують «сканер», який не парсить всі дані одночасно. Наприклад, у зображенні розміром 200×200 нейромережа зчитує квадрат розміру 20x20, потім зміщується на 1 піксель і рахує новий квадрат, і т.д. Вхідні дані передаються через згорткові шари, в яких не всі вузли з'єднані між собою.

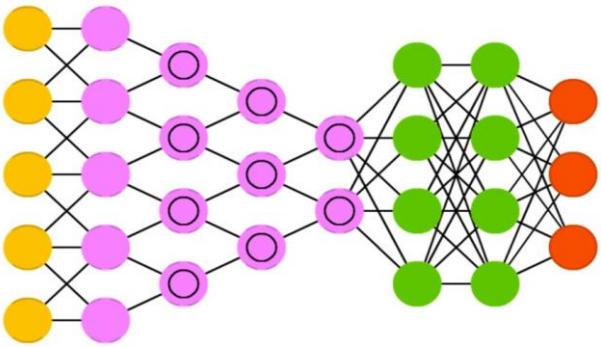


Рис. 2.18 Згорткові нейромережі

Згорткові нейронні мережі мають кілька ключових особливостей побудови, які відрізняють їх від інших типів нейронних мереж. Згорткові шари (Convolutional Layers) застосовують фільтри (ядра згортки) до вхідного зображення або попереднього шару, що дозволяє виділяти локальні ознаки. Кожен фільтр проходить через вхідне зображення, створюючи карти ознак (feature maps). Перевагою є зменшення кількості параметрів порівняно з повнозв'язними шарами.

Шари підвибірки (Pooling Layers) виконують операції підвибірки (наприклад, max pooling або average pooling), зменшуючи розмір карт ознак. Зменшують розмірність даних і кількість параметрів, що допомагає зменшити обчислювальну складність та уникнути перенавчання.

В якості функції активації зазвичай використовують функцію ReLU для введення нелінійності у модель.

Шари нормалізації (Normalization Layers) виконують нормалізацію виходів між шарами, наприклад, Batch Normalization, що прискорює навчання та стабілізує його.

Шари об'єднання (Fully Connected Layers) зазвичай розташовані в кінці архітектури CNN. Всі нейрони пов'язані між собою, що дозволяє поєднувати ознаки, виділені попередніми шарами, для виконання класифікації або інших завдань.

Dropout Layers використовуються для запобігання перенавчанню. Випадковим чином "вимикають" деякі нейрони під час навчання, що сприяє узагальненню моделі.

Ці особливості роблять CNN дуже ефективними для обробки зображень та інших даних з просторовою структурою, забезпечуючи високу точність і продуктивність у різних завданнях машинного навчання.

Розгорткові нейромережі (deconvolutional networks, DN) є зворотніми до згорткових нейронних мереж (рис. 2.19). Наприклад, ми передаємо мережі слово "кіт" (певним чином закодоване) і мережа генерує картинки, схожі на реальні зображення котів: приблизно в цьому ключі працює розгорткова нейромережа.

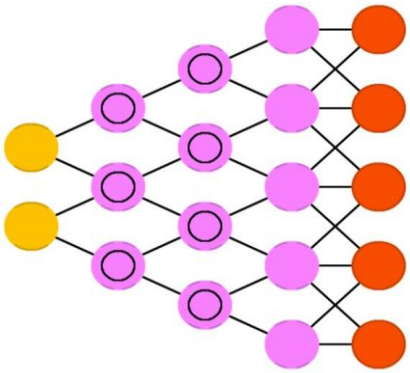


Рис. 2.19 Розгорткові нейронні мережі

Нейронні мережі DN мають свої характерні особливості. Використовуються нелінійні функції активації, що й у згорткових мережах, такі як ReLU, щоб ввести нелінійність у модель. Знаходять застосування у генеративних моделях, для суперрезолюції зображень - поліпшення якості зображень шляхом збільшення їх роздільної здатності, розпізнавання і класифікації кожного пікселя у зображенні.

Розгорткові мережі часто використовуються як частина архітектур, які симетричні до згорткових мереж. Наприклад, у автоенкодерах або U-Net, де згорткові шари зменшують розмірність, а розгорткові її відновлюють. Навчання здійснюється за допомогою зворотного розповсюдження помилки та градієнтного спуску, як і в інших нейромережах. Однак, з урахуванням особливостей розгорткових шарів для відновлення просторової структури даних. Ці особливості роблять розгорткові нейромережі потужними інструментами для задач відновлення зображень, семантичної сегментації та генерації нових даних з високою роздільною здатністю.

Глибинна мережа переконань (deep belief network, DBN) містить кілька прихованих шарів, у кожному з яких нейрони не пов’язані між собою, проте пов’язані із нейронами сусіднього шару. Такі мережі навчаються поблочно, використовуючи тактику «жадібного навчання», яка полягає у виборі локальних оптимальних рішень, що не гарантують оптимального кінцевого результату (рис. 2.20).

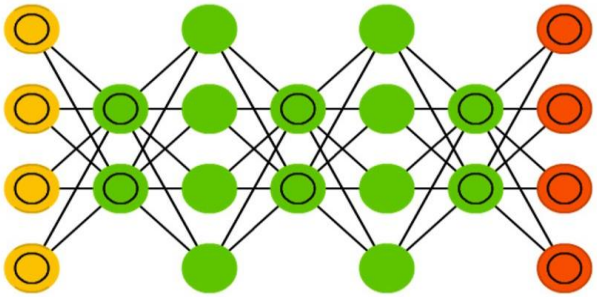


Рис. 2.20 Глибинна мережа переконань

Мережі даного типу знаходять застосування у розпізнаванні образів; генеративних моделях; використовуються для зменшення розмірності даних, зберігаючи їх важливі ознаки; використовуються у рекомендаційних системах для вивчення переваг користувачів; для задач розпізнавання образів, таких як розпізнавання рукописних цифр та класифікації зображень; для генерації нових зразків даних. DBN є потужними інструментами для задач, які вимагають витягнення складних ознак та моделювання складних розподілів у даних.

2.4 Огляд мов програмування та популярних бібліотек для роботи з нейронними мережами

Розповсюдженими мовами програмування для роботи з нейронними мережами є такі мови як Python, R, Julia, JavaScript, C++, Java, MATLAB.

Python - найпоширеніша мова для розробки та досліджень у сфері нейронних мереж і машинного навчання. Багато бібліотек для нейронних мереж, такі як TensorFlow, PyTorch, Keras, Theano написані на мові Python або мають підтримку цієї мови.

Мова програмування R часто використовується для статистичних обчислень і аналізу даних. Вона також підходить для побудови та тестування простих моделей нейронних мереж, а такі бібліотеки як Keras та Tensorflow підтримують роботу з нейронними мережами на R.

Мова Julia швидко набуває популярності у сфері нейронних мереж та обчислень з великими даними завдяки своїй швидкості й простоті у синтаксисі. Вона підтримується у бібліотеках Flux.jl і Knet.jl.

JavaScript (зокрема, з використанням TensorFlow.js) також використовується для роботи з нейронними мережами, особливо для веб-застосунків. За допомогою TensorFlow.js можна створювати та запускати нейронні мережі прямо у браузері.

Мова програмування C++ зазвичай використовується для побудови високопродуктивних моделей та роботи з великими обчисленнями. TensorFlow та деякі інші фреймворки містять низькорівневий код на C++, що дає змогу досягти вищої продуктивності.

Java хоча й менш популярна для нейронних мереж, використовується у великих корпоративних проєктах та для побудови серверних рішень. Бібліотеки, такі як DL4J (DeepLearning4J), дозволяють працювати з нейронними мережами на Java.

MATLAB традиційно використовується у наукових дослідженнях і має набір інструментів для побудови нейронних мереж. Він часто використовується в академічному середовищі та для роботи з моделями вбудованих систем.

Python залишається найбільш популярним вибором через зручність, простоту, широкий спектр бібліотек і підтримку спільноти.

Популярними фреймворками для роботи з нейронними мережами є наступні: TensorFlow, Keras, PyTorch, Theano.

Фреймворк *TensorFlow* створений компанією Google. Він надає потужні інструменти для побудови, тренування і розгортання моделей, а також має широке ком’юніті і підтримує багато бібліотек та технологій (рис. 2.21).

Характеристиками TensorFlow є висока продуктивність, гнучкість і модульність, широкий набір інструментів для розробки, широка підтримка

платформ, велика спільнота та ресурси, підтримка роботи з графами обчислень, підтримка багатомовності, постійний розвиток та оновлення. Google активно підтримує і розвиває TensorFlow, випускаючи регулярні оновлення, нові функції та оптимізації. Це гарантує, що фреймворк залишатиметься актуальним і буде адаптуватися до нових вимог і трендів у галузі машинного навчання.

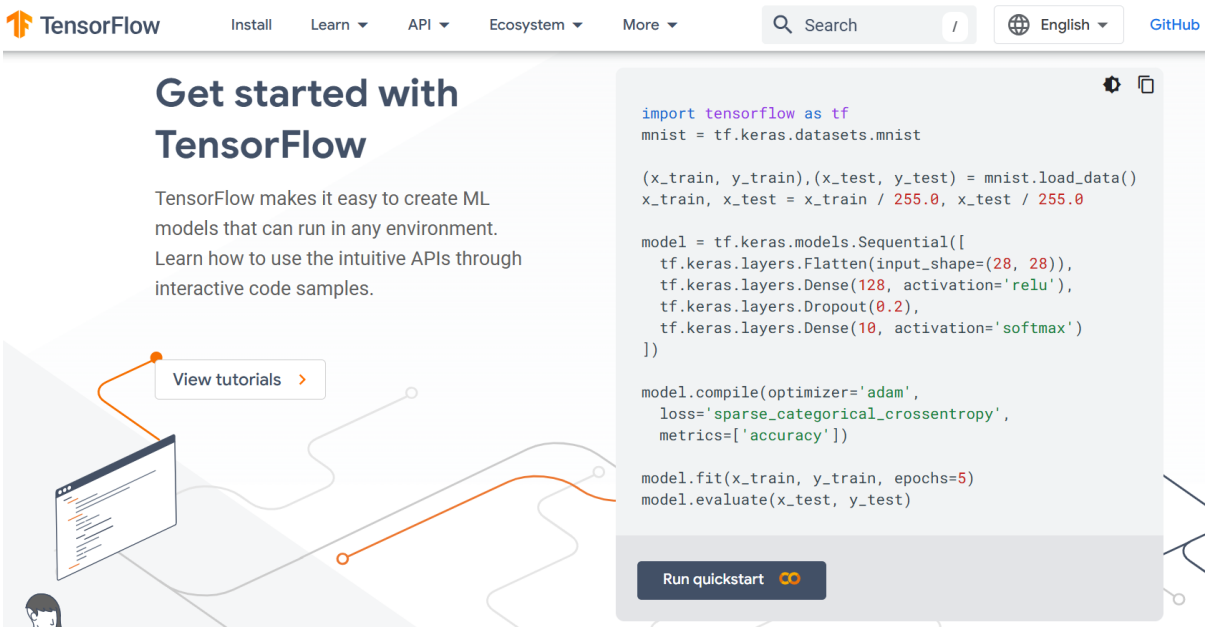


Рис. 2.21 Фреймворк TensorFlow

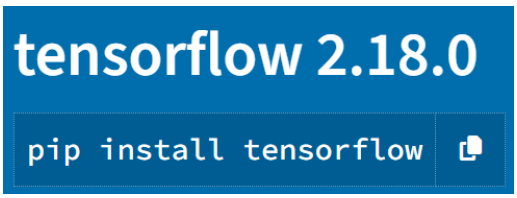


Рис. 2.22 Версія TensorFlow станом на 11.11.2024

TensorFlow може використовувати як центральний процесор (CPU), так і графічний процесор (GPU) для прискорення обчислень, що дозволяє значно скоротити час тренування моделей. TensorFlow також підтримує роботу на кластерах, що робить його гарним вибором для обробки великих даних.

TensorFlow містить наступні модулі для завдань:

- *TensorFlow Hub* — бібліотека попередньо навчених моделей, які можна адаптувати для власних задач;

- *TensorFlow Lite* — полегшена версія TensorFlow для мобільних пристроїв і вбудованих систем;

- *TensorFlow.js* — дозволяє запускати моделі прямо у браузері;

- *TensorFlow Extended (TFX)* — набір інструментів для розгортання моделей машинного навчання у виробниче середовище.

Хоча основна робота відбувається з Python, TensorFlow підтримує й інші мови програмування, включаючи C++, JavaScript, R і Swift, що робить його доступним для розробників з різними технологічними бекграундами.

Фреймворк *Keras* є популярним у сфері розробки нейронних мереж, надає високорівневий API для спрощеного створення і тренування моделей. Створений для швидкої розробки та експериментування з нейронними мережами, орієнтуючись на простоту використання, модульність та зручний інтерфейс (рис. 2.23).

Характеристиками Keras є простота використання, модульна структура, інтеграція з TensorFlow, підтримка різних типів нейронних мереж, вбудовані інструменти для тренування і оцінки моделей, можливості для швидкого прототипування, підтримка основних обчислювальних платформ, широкий набір готових моделей.

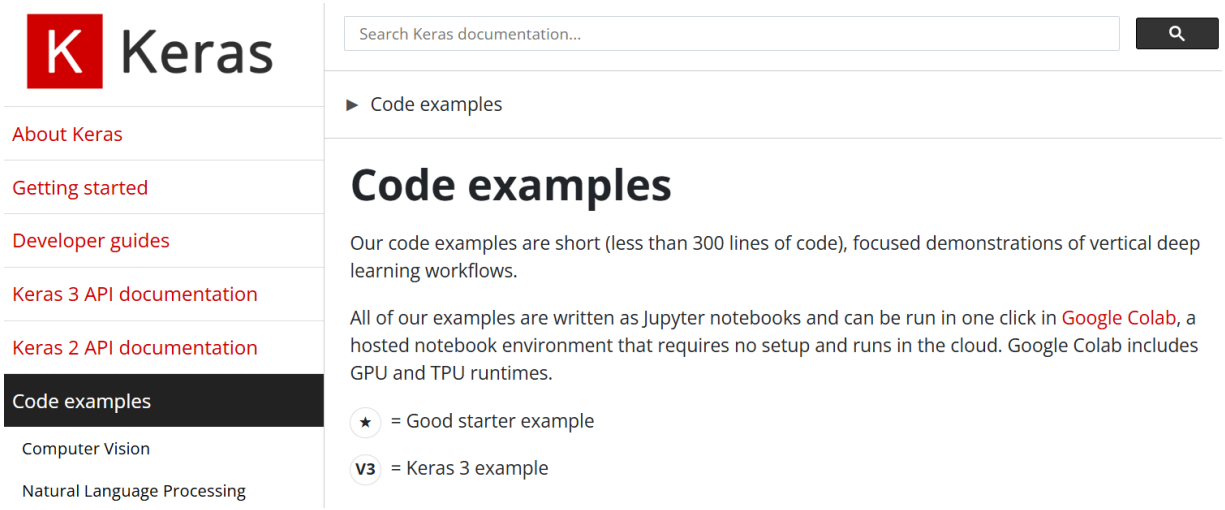


Рис. 2.23 Фреймворк Keras

Фреймворк побудований за модульною структурою, де кожен елемент моделі (шари, функції активації, оптимізатори) є окремим модулем, який можна легко налаштовувати та комбінувати. Це дозволяє швидко створювати та експериментувати з різними архітектурами нейронних мереж.

PyTorch — фреймворк для машинного та глибокого навчання, розроблений компанією Facebook AI Research. Набув популярності серед дослідників і розробників завдяки інтуїтивному інтерфейсу, гнучкості, а також можливостям для створення і тестування моделей нейронних мереж. PyTorch особливо цінують у наукових колах і для швидкого прототипування моделей завдяки динамічному підходу до обчислень (рис. 2.24).

Характеристиками PyTorch є гнучкість і модульність, простий і зрозумілий синтаксис, широка підтримка нейронних мереж, підтримка роботи з GPU та CPU, спільнота та популярність у наукових дослідженнях.

PyTorch має низку доповнень, які допомагають створювати і розгортати моделі:

- *TorchVision* - для роботи з зображеннями;
- *TorchText* - для роботи з текстовими даними;
- *TorchAudio* - для роботи з аудіо;
- *PyTorch Lightning* - для спрощення коду та підвищення ефективності при масштабуванні проєктів;
- *Hugging Face Transformers* - для роботи з трансформерами, що забезпечують NLP-моделі (GPT, BERT).

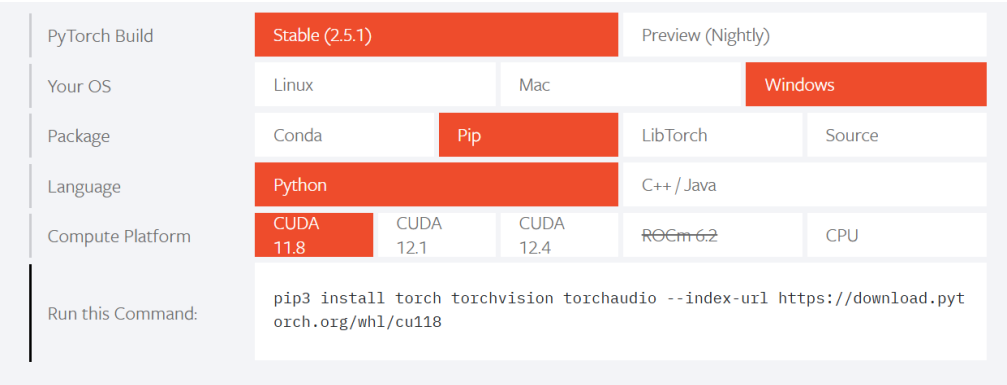


Рис. 2.24 Фреймворк PyTorch

PyTorch швидко став популярним серед дослідників і наукових кіл, оскільки дозволяє легко експериментувати з новими моделями та алгоритмами. Велика спільнота та підтримка відкритого коду сприяють тому, що багато нових ідей у галузі глибокого навчання реалізуються на PyTorch.

Theano — бібліотека для наукових обчислень, розроблена в Університеті Монреаля, яка дозволяє ефективно будувати та обчислювати математичні вирази з багатовимірними масивами. Є одним із перших фреймворків, створених для розробки нейронних мереж та інших моделей машинного навчання з можливістю обчислень на GPU (рис. 2.25).

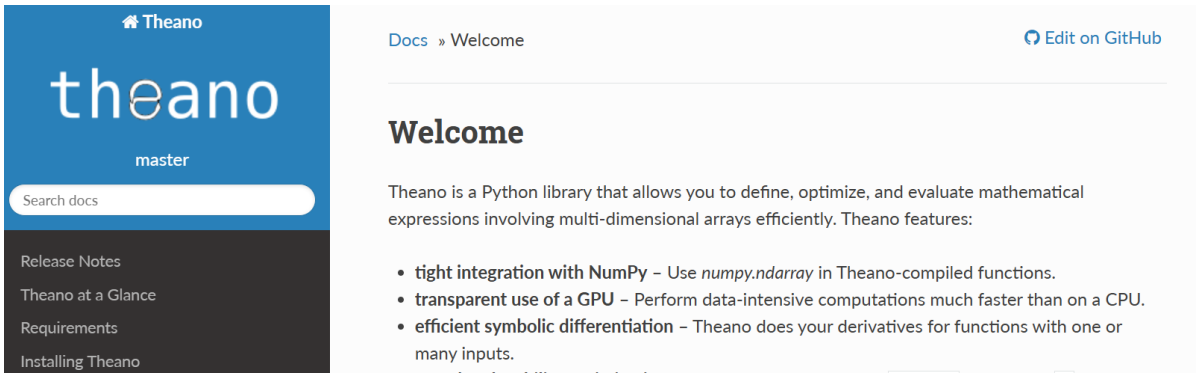


Рис. 2.25 Фреймворк Theano

Характеристиками Theano є компіляція обчислювальних графів для оптимізації швидкодії, підтримка GPU та CPU, можливість автоматичного диференціювання, оптимізація використання пам'яті.

Theano автоматично компілює математичні вирази та оптимізує їх для прискорення обчислень. Він дозволяє виконувати складні обчислення з великою швидкістю, що особливо корисно для тренування нейронних мереж. Бібліотека має спеціальні оптимізації для роботи з великими масивами даних і багатовимірними тензорами.

Theano використовує символьну математичну систему, де кожен компонент виразу представлений як символ. Це дозволяє будувати складні обчислювальні графи, автоматично знаходити похідні та виконувати чисельне обчислення з високою точністю. Ця властивість полегшує реалізацію градієнтного спуску, що є критичним для тренування нейронних мереж.

3 РОЗРОБКА МОДЕЛІ НЕЙРОННОЇ МЕРЕЖІ ДЛЯ ІДЕНТИФІКАЦІЇ ЗОБРАЖЕНЬ

3.1 Виконання імпорту та характеристика необхідних бібліотек

Провівши аналіз середовищ для розробки нейронних мереж, визначено наступне. Google Colab — хмарне середовище для обчислень, що пропонує зручний інструмент для розробки та навчання нейронних мереж, так як воно забезпечує безкоштовне використання обчислювальних ресурсів, основане на Jupyter Notebook, підтримує Python та інші бібліотеки для машинного навчання, версії Python та оновлення середовищ, спільний доступ та роботу у реальному часі.

Jupyter Notebook є одним із найпопулярніших середовищ для обчислень і наукових досліджень, зокрема для розробки нейронних мереж. Його функціональність робить його зручним для експериментів і побудови прототипів у галузі машинного навчання. Jupyter Notebook дозволяє працювати з такими бібліотеками, як TensorFlow, PyTorch, Keras, Scikit-learn та багатьма іншими, що робить його універсальним середовищем для навчання, налаштування та тестування нейронних мереж. Jupyter Notebook підтримує безліч розширень, які додають нові можливості, такі як автозбереження, підтримка інтерактивних віджетів, перевірка синтаксису. Це дозволяє адаптувати середовище до конкретних потреб розробки. Тому, для розробки нейронної мережі, вибране середовище Jupyter Notebook.

Використаємо бібліотеку warnings для фільтрації попереджень, для того, щоб зробити код більш чистим та читабельним, контролюючи відображення попереджень. Це особливо корисно при аналізі даних і навчанні моделей, де зайві попередження можуть ускладнювати роботу з результатами.

```
import sys
print("version:", sys.version)
import warnings
warnings.filterwarnings("ignore")
```

```
version: 3.10.9 | packaged by Anaconda, Inc. | (main, Mar  1 2023, 18:18:15) [MSC v.1916 64 bit (AMD64)]
```

зробити код більш чистим та читабельним, контролюючи відображення попереджень. Це особливо корисно при аналізі даних і навчанні моделей, де зайві попередження можуть ускладнювати роботу з результатами.

Для програмування нейронної мережі використаємо фреймворки TensorFlow, Keras, характеристика яких виконана у роботі у розділі 2.4. В якості датасету використано MNIST - набір даних, який використовується для навчання та тестування моделей комп'ютерного зору, зокрема для задач розпізнавання рукописних цифр. Нейронні мережі вчаться на основі даних. Для цього вони потребують великих і різноманітних наборів даних, щоб знайти шаблони і вивчити залежності між вхідними та вихідними значеннями. Якщо нейронна мережа не має даних для тренування, вона не зможе створити точну модель. Чим більший і різноманітніший датасет, тим більше варіацій в даних можуть бути охоплені.

```
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import utils
from tensorflow.keras.preprocessing import image
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
%matplotlib inline
```

Бібліотека *NumPy* є однією з основних для наукових обчислень в Python і має багато переваг, які роблять її дуже популярною серед розробників і дослідників. NumPy значно швидше за стандартні Python-методи при роботі з великими масивами і матрицями, дозволяє створювати і ефективно працювати з багатовимірними масивами (матрицями, тензорами), що значно спрощує обчислення. Масиви в NumPy займають значно менше пам'яті у порівнянні з звичайними списками Python, оскільки вони зберігають дані в компактній і ефективній формі. NumPy є основою для багатьох інших бібліотек в Python, що дозволяє легко інтегруватися з іншими бібліотеками для більш складних задач, таких як обробка даних, статистика, машинне навчання та оптимізація. NumPy дозволяє ефективно обробляти великий обсяг даних, що робить його важливим

інструментом для роботи з великими наборами даних, де продуктивність і ефективність пам'яті мають велике значення.

Matplotlib — популярна бібліотека для створення візуалізацій в Python. Підтримує велику кількість графічних елементів, є дуже сумісною з іншими бібліотеками Python, дозволяє зберігати графіки у різних форматах. Інтегрується з Jupyter Notebook і підтримує відображення графіків безпосередньо у блоках коду, що є дуже корисним для інтерактивного аналізу даних і презентацій. Також є підтримка відображення графіків у реальному часі в середовищах, таких як JupyterLab.

3.2 Визначення способу створення моделі нейронної мережі

У Keras (бібліотека, що є частиною TensorFlow) клас *Sequential* використовується для створення моделей, які складаються з лінійної послідовності шарів, де кожен шар має лише один вхід та один вихід. Однак, є кілька інших підходів для побудови моделей, які можна порівняти з Sequential. *Functional API* є більш гнучким та потужним способом побудови моделей, який дозволяє створювати моделі з більш складними архітектурами, такими як моделі з кількома входами та виходами, або моделі з розгалуженнями (branching), які не є лінійними. Functional API підходить для більш складних моделей з багатими і гнучкими архітектурами, такими як багатоканальні мережі або моделі з кількома входами/виходами. Можна створювати моделі, підкласовуючи *Model*, що дозволяє отримати повний контроль над поведінкою мережі, у тому числі під час виконання прямого та зворотного проходу. Model - найгнучкіший підхід, що дозволяє створювати дуже складні моделі з кастомною поведінкою та логікою.

Sequential - спосіб створення моделей, які мають лінійну структуру, і підходить для базових задач, де немає необхідності в складних зв'язках між шарами. Оскільки у нас модель з послідовними шарами, доцільно використовувати модель Sequential для створення нейронної мережі для розпізнавання зображень.

```
model = Sequential()
model.add(Dense(2, input_dim = 2, use_bias=False))
model.add(Dense(1, use_bias=False))
```

Команда *model.summary()* використовується для виведення резюме моделі. Вона надає загальну інформацію про архітектуру моделі, зокрема про її шари, кількість параметрів та їх розміри. Це корисний інструмент для перевірки структури моделі після її створення.

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 2)                 4

 dense_1 (Dense)             (None, 1)                 2

=================================================================
Total params: 6 (24.00 Byte)
Trainable params: 6 (24.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Коефіцієнти у нейронах нейронної мережі, також відомі як ваги (weights) та biases, виконують важливу роль у процесі навчання моделі та її здатності адаптуватися до даних. Ваги відповідають за масштабування входів нейронів. Вони визначають, наскільки важливий кожен вхід для виходу нейрона. Вага модулює величину входу перед тим, як вона потрапить до нейрона, і є основним параметром, який змінюється під час навчання моделі.

Вага визначає силу впливу кожного вхідного значення на вихід нейрону. Під час тренування моделі нейронна мережа намагається оптимізувати ваги, щоб мінімізувати різницю між передбаченими і фактичними значеннями.

Формула для нейрона:

$$z = w_1 \cdot x_1 + w_2 \cdot x_2 + \cdots + w_n \cdot x_n + b ,$$

де $w_1, w_2, \ldots, w_n$ — ваги;

$x_1, x_2, \ldots, x_n$ — вхідні значення;

b – відхилення байєса;

z - зважене сумування вхідних значень.

```
weights = model.get_weights()
print(weights)
```

```
[array([[ 0.24123538, -0.37492055],
       [ 0.16657782, -1.1274049 ]], dtype=float32), array([[-0.70594555],
       [ 0.8774115 ]], dtype=float32)]
```

Є можливість задавати коефіцієнти $w_1, w_2, \ldots, w_n$ наступним чином:

```
w1 = 0.42
w2 = 0.15
w3 = -0.56
w4 = 0.83
w5 = 0.93
w6 = 0.02
new_weight = [np.array([[w1,w3],[w2,w4]]), np.array([[w5],[w6]])]
print(new_weight)
model.set_weights(new_weight)
```

Команда *model.set_weights()* використовується для задання значень вагів і байєсів конкретної моделі. Ця команда дозволяє вручну встановити ваги для всіх шарів моделі:

```
[array([[ 0.42, -0.56],
       [ 0.15,  0.83]]), array([[0.93],
       [0.02]])]
```

Це може бути корисним у наступних ситуаціях:

- при перенесенні ваг з однієї моделі на іншу (наприклад, при використанні попередньо навчених моделей);
- при відновленні стану моделі збереженого набору вагів;
- для ручного встановлення вагів після навчання або для експериментів з певними значеннями.

```
x1 = 7.2
x2 = -5.8
x_train = np.expand_dims(np.array([x1, x2]), 0)
x_train.shape
```

```
(1, 2)
```

```
k = np.array([x1, x2])
k.shape
```

```
(2,)
```

```
y_linear = model.predict(x_train)
print(y_linear)
```

```
1/1 [==============================] - 0s 155ms/step
[[1.8262998]]
```

Команда *model.get_weights()* в середовищі машинного навчання використовується для отримання ваг (параметрів) поточної моделі. Вона повертає список, що містить масиви з вагами всіх шарів моделі. Кожен елемент цього списку – це масив ваг для відповідного шару. Ці ваги є результатом процесу навчання моделі та дозволяють зберегти або проаналізувати стан кожного шару.

Застосування *model.get_weights()* є корисним, якщо потрібно:

- зберегти ваги моделі для майбутнього використання або аналізу;
- порівняти ваги різних моделей;
- відновити модель після змін, наприклад, для перевірки різниці у результатах;
- використати ваги для передавання знань в інші моделі (трансферне навчання).

```
model.get_weights()
```

```
[array([[ 0.42, -0.56],
        [ 0.15,  0.83]], dtype=float32),
 array([[0.93],
        [0.02]], dtype=float32)]
```

```
model(x_train)
```

```
<tf.Tensor: shape=(1, 1), dtype=float32, numpy=array([[1.8262998]], dtype=float32)>
```

Виконаємо обчислення:

$$Y = N_1 * w_5 + N_2 * w_6 \quad (3.1)$$

$$N1 = x_1 * w_1 + x_2 * w_2 \quad (3.2)$$

$$N2 = x_1 * w_3 + x_2 * w_4 \quad (3.3)$$

```
N1 = x1 * w1 + x2 * w2
N2 = x1 * w3 + x2 * w4
print(N1)
print(N2)
```

```
2.154
-8.846
```

```
Y_linear = N1 * w5 + N2 * w6
print(Y_linear)
```

```
1.8263000000000003
```

3.3 Вибір функцій активації для створення нейронної мережі

Функції активації у нейронних мережах відіграють ключову роль у визначенні виходу нейрона для заданих вхідних даних, а також вносять нелінійність у модель. Без них нейронна мережа була б просто лінійною комбінацією шарів, що обмежувало б її здатність вирішувати складні задачі, оскільки лінійна модель не здатна добре описувати нелінійні залежності. Функція активації вирішує, чи буде нейрон активований (тобто, передасть інформацію до наступного шару). Вона трансформує зважену суму входів нейрона у вихідний сигнал, який буде або передано далі, або обнулено, залежно від типу функції.

Види функцій активації: Binary Step Function, Linear Activation Function, Sigmoid/Logistic, Tanh, ReLU, Leaky ReLU.

Функція активації *ReLU* (Rectified Linear Unit) є однією з найбільш популярних функцій активації, особливо для прихованих шарів глибоких нейронних мереж (рис. 3.1).

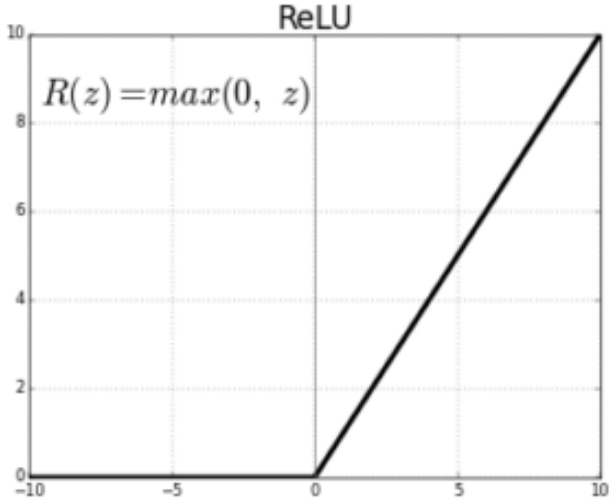


Рис. 3.1 Функція активації ReLU

Функція активації ReLU описується формулою:

$$R(z)=max(0,z) \tag{3.4}$$

Це означає, що для всіх від’ємних значень z результатом є 0, а для додатніх значень результат дорівнює z.

ReLU додає нелінійність у модель, але простішу, ніж інші функції, такі як сигмоїдна або гіперболічний тангенс. Завдяки цьому модель здатна навчатися

складним залежностям, не потрапляючи в надмірну обчислювальну складність. На відміну від Sigmoid та Tanh, ReLU не має області, де похідна дуже близька до 0 для великих значень аргументу. Це зменшує проблему зникаючого градієнта, що є важливою перевагою при тренуванні глибоких мереж. ReLU є дуже простою у розрахунках, оскільки вимагає лише порівняння із значенням 0. Це зменшує обчислювальне навантаження під час навчання. Моделі з ReLU-шарами часто збігаються швидше під час тренування, ніж моделі з сигмоїдними або tanh-функціями, оскільки ReLU дозволяє зберігати градієнти більшими для великих значень z.

```
def relu(x):
    return np.clip(x, 0, np.inf)
```

```
model_relu = Sequential()
model_relu.add(Dense(2, input_dim = 2, activation = 'relu', use_bias=False))
model_relu.add(Dense(1, activation = 'relu', use_bias=False))
model_relu.summary()
model_relu.set_weights(new_weight)
```

Метод *summary()* використовується для отримання короткої характеристики структури моделі. Цей метод корисний для отримання інформації про архітектуру моделі, особливо якщо модель складна і містить багато шарів. Метод *set_weights()* дозволяє вручну встановити ваги моделі. Він приймає як аргумент список *new_weight*, де кожен елемент списку відповідає вагам визначеного шару.

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_2 (Dense)             (None, 2)                 4

 dense_3 (Dense)             (None, 1)                 2

=================================================================
Total params: 6 (24.00 Byte)
Trainable params: 6 (24.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
y_relu = model_relu.predict(x_train)
print(y_relu)
```

```
1/1 [==============================] - 0s 80ms/step
[[2.0032198]]
```

```
H1_relu = relu(x1 * w1 + x2 * w2)
H2_relu = relu(x1 * w3 + x2 * w4)
print(H1_relu)
print(H2_relu)
```

```
2.154
0.0
```

Отримуємо:

```
Y_relu = relu(H1_relu * w5 + H2_relu * w6)
print(Y_relu)
```

```
2.0032200000000002
```

Функція активації *Sigmoid* є однією з найпопулярніших активаційних функцій, особливо для вихідних шарів моделей, що розв'язують задачі двокласової класифікації (рис. 3.2).

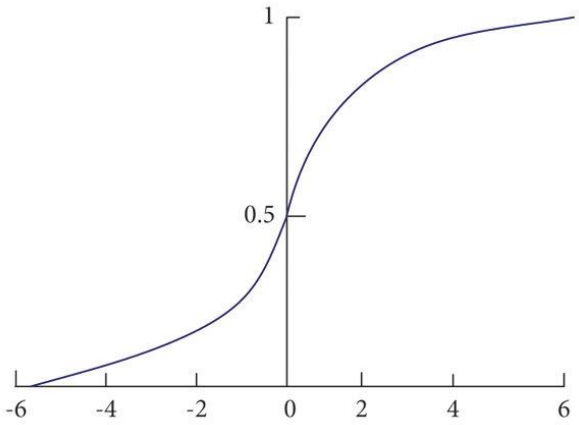


Рис. 3.2 Функція активації Sigmoid

Функція активації Sigmoid описується формулою:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3.5}$$

Ця формула забезпечує плавне перетворення значень, обмежуючи вихідний результат між 0 і 1. Коли значення *x* дуже велике, результат наближається до 1; коли *x* дуже мале, результат наближається до 0.

Sigmoid – неперервна та диференційована функція, що дозволяє застосовувати алгоритми зворотного поширення помилки. Це одна з причин, чому вона історично була популярною у ранніх моделях нейронних мереж. Вихідні значення функції Sigmoid можуть інтерпретуватися як ймовірність належності до певного класу. У нейронних мережах функція активації Sigmoid часто використовується в

останньому шарі для задач двокласової класифікації, оскільки дозволяє отримувати значення, обмежені між 0 та 1, що зручно для розрахунку ймовірностей.

```
def sigmoid(x):
    return 1/1+np.e**(-x)
```

Розробляємо нейронну мережу:

```
model_sigmoid = Sequential()
model_sigmoid.add(Dense(2, input_dim = 2, activation = 'sigmoid', use_bias=False))
model_sigmoid.add(Dense(1, activation = 'sigmoid', use_bias=False))
model_sigmoid.summary()
model_sigmoid.set_weights(new_weight)
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 2)                 4

 dense_5 (Dense)             (None, 1)                 2

=================================================================
Total params: 6 (24.00 Byte)
Trainable params: 6 (24.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Отримуємо:

```
y_sigmoid = model_sigmoid.predict(x_train)
print(y_sigmoid)
```

```
1/1 [==============================] - 0s 82ms/step
[[0.6970569]]
```

```
H1_sigmoid = sigmoid(x1 * w1 + x2 * w2)
H2_sigmoid = sigmoid(x1 * w3 + x2 * w4)
print(H1_sigmoid)
print(H2_sigmoid)
```

```
1.116019151774409
6947.547135018005
```

```
Y_sigmoid = sigmoid(H1_sigmoid * w5 + H2_sigmoid * w6)
print(Y_sigmoid)
```

```
1.0
```

Функція активації *гіперболічний тангенс (tanh)* є популярною активаційною функцією для нейронних мереж, особливо для прихованих шарів (рис. 2.28). Функція активації tanh описується формулою:

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.6}$$

Функція tanh обмежує вихідні значення між -1 і 1. Для великих значень *x* функція прямує до 1, а для малих - до -1. Tanh є гладкою та диференційованою функцією, що дозволяє використовувати її у зворотному поширенні помилки, як і у випадку з функцією Sigmoid. Подібно до Sigmoid, функція tanh вносить нелінійність у модель, дозволяючи їй опрацьовувати складні нелінійні зв'язки у даних.

Оскільки вихідні значення tanh центровані відносно нуля, це знижує зміщення середнього значення, що дозволяє швидше збігатися під час навчання, ніж у випадку з Sigmoid. Гіперболічний тангенс має діапазон від -1 до 1, що забезпечує ширше охоплення значень порівняно з Sigmoid. Це дозволяє функції бути більш чутливою до зміни вхідних даних, особливо для негативних значень.

Гіперболічний тангенс є хорошим вибором для моделей середньої глибини та задач, де важливо мати симетричну активацію. Однак у дуже глибоких моделях частіше застосовують ReLU або інші функції для уникнення проблеми зникаючого градієнта.

```
def th(x):
    return (np.e ** x - np.e ** (-x)) / (np.e ** x + np.e ** (-x))
```

Нейронна мережа з функцією активації гіперболічного тангенсу (tanh):

```
model_th = Sequential()
model_th.add(Dense(2, input_dim = 2, activation=th, use_bias=False))
model_th.add(Dense(1, activation=th, use_bias=False))
model_th.summary()
model_th.set_weights(new_weight)
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_9 (Dense)             (None, 2)                 4

 dense_10 (Dense)            (None, 1)                 2

=================================================================
Total params: 6 (24.00 Byte)
Trainable params: 6 (24.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
y_th = model_th.predict(x_train)
print(y_th)
```

```
1/1 [==============================] - 0s 113ms/step
[[0.70906264]]
```

```
H1_th = th(x1 * w1 + x2 * w2)
H2_th = th(x1 * w3 + x2 * w4)
print(H1_th)
print(H2_th)
```

```
0.9734366670870239
-0.9999999585531038
```

```
Y_th = th(H1_th * w5 + H2_th * w6)
print(Y_th)
```

```
0.709062603515898
```

3.4 Оцінка якості роботи моделі нейронної мережі

Метрики у нейронних мережах використовують для оцінки якості роботи моделі, тобто для вимірювання, наскільки добре модель виконує поставлене завдання. В залежності від типу завдання метрики можуть бути різними, але основна їх функція – надавати числові показники якості, які допомагають розробникам та дослідникам зрозуміти, наскільки точно модель прогнозує або класифікує дані.

Метрики дозволяють порівнювати результати роботи моделі з реальними значеннями, застосовуються для відстеження процесу навчання на кожній епосі, що дозволяє бачити, чи модель покращує свої результати з часом. Якщо метрика показує стабільні значення або починає погіршуватися, це може свідчити про перенавчання або недонавчання.

Метрики дозволяють об'єктивно порівнювати різні архітектури або налаштування нейронних мереж, щоб вибрати оптимальну модель для певного завдання. Значення метрик можуть слугувати показником для налаштування гіперпараметрів, таких як розмір шару, кількість епох, швидкість навчання, щоб досягати найкращої продуктивності. Метрики є важливим інструментом для аналізу та контролю ефективності нейронних мереж, що допомагає забезпечити досягнення поставлених цілей і оптимізувати моделі для конкретних завдань.

MSE (Mean Squared Error, середньоквадратична похибка) - метрика, яку часто використовують для оцінки якості моделей регресії, включаючи нейронні мережі.

Вона показує середню квадратичну різницю між передбаченими моделлю значеннями та реальними значеннями у тестовому наборі даних.

Формула обчислення MSE:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \quad , \tag{3.7}$$

де n – загальна кількість прикладів у тестовій вибірці;

y_i – справжнє значення для i-го прикладу;

$\acute{y}_i$ – передбачене моделлю значення для i-го прикладу.

MSE є основною метрикою у задачах регресії, таких як прогнозування цін, обсягів продажів або будь-якої іншої неперервної змінної, де важливо мінімізувати відхилення від реальних значень.

MAE (Mean Absolute Error, середня абсолютна похибка) - метрика оцінки якості моделей регресії, яка вимірює середню величину помилок між передбаченими значеннями моделі та реальними значеннями. На відміну від MSE, MAE обчислює середнє абсолютне значення похибок, тобто різниці між прогнозом і справжнім значенням без зведення до квадрату. Це робить MAE менш чутливою до великих помилок, що може бути корисним у деяких випадках.

Формула обчислення MAE:

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \quad , \tag{3.8}$$

де n – загальна кількість прикладів у тестовій вибірці;

y_i – справжнє значення для i-го прикладу;

$\acute{y}_i$ – передбачене моделлю значення для i-го прикладу.

MAE часто використовують у задачах, де моделі повинні стабільно передбачати результати з помірними відхиленнями, а викиди не повинні впливати на загальну оцінку якості.

Cross-Entropy Loss (CEloss, крос-ентропійна похибка) - функція втрат, яка широко використовується у задачах класифікації для оцінки якості передбачень

моделі. CEloss вимірює різницю між розподілом ймовірностей, який видає модель для кожного класу, і справжнім розподілом ймовірностей (який зазвичай має значення 1 для справжнього класу і 0 для інших). Мета полягає в мінімізації цієї різниці, що забезпечує коректну класифікацію прикладів.

Формула обчислення Cross-Entropy Loss для бінарної класифікації:

$$\mathrm{CE}_{\mathrm{binary}} = -\frac{1}{n}\sum_{i=1}^{n}\left(y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right) \quad , \tag{3.9}$$

де n – кількість прикладів;

y_i – справжня мітка (0 або 1) для i-го прикладу;

$\acute{y}_i$ – ймовірність, передбачена моделлю, що i-й приклад належить до класу 1.

Для багатокласової класифікації, формула узагальнюється на випадок декількох класів:

$$\mathrm{CE}_{\mathrm{multiclass}} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{k} y_{i,j}\log(\hat{y}_{i,j}) \quad , \tag{3.10}$$

де k – кількість класів;

$y_{i,j}$ – справжня ймовірність (1 або 0) для класу j у i-му прикладі;

$\acute{y}_{i,j}$ – ймовірність, передбачена моделлю для класу j у i-му прикладі.

CEloss найчастіше застосовується для задач класифікації, таких як розпізнавання образів, обробка тексту та класифікація природних мов, де потрібно передбачити, до якого класу належить приклад.

Оптимізатори у нейронних мережах виконують функцію оновлення вагових коефіцієнтів моделі, щоб мінімізувати значення функції втрат (наприклад, MSE, MAE або Cross-Entropy Loss). Цей процес є основою навчання нейронної мережі і визначає, як ефективно та швидко модель зможе навчитися виконувати завдання. Завдання оптимізатора — знаходити мінімальне значення функції втрат, що забезпечує високу точність передбачень моделі.

Оптимізатор шукає мінімум функції втрат, коригуючи ваги мережі так, щоб зменшити різницю між передбаченими значеннями і справжніми мітками (цільовими значеннями) у навчальних даних. Таким чином, оптимізатор

допомагає моделі зменшувати помилки під час навчання. Більшість оптимізаторів використовують метод градієнтного спуску або його варіації для оновлення вагів мережі. Вони обчислюють похідну функції втрат за кожним параметром (вагою), щоб визначити напрямок і величину зміни. Оптимізатор оновлює ваги у напрямку, протилежному градієнту, щоб зменшити втрати.

Деякі оптимізатори (наприклад, Adam, RMSprop, AdaGrad) можуть адаптувати швидкість навчання у процесі навчання, щоб забезпечити стабільніший і швидший пошук мінімуму. Наприклад, на початку навчання швидкість може бути більшою, щоб швидко знаходити напрямок, а з часом зменшуватися, щоб точніше досягти мінімуму.

Адаптивні оптимізатори (наприклад, Adam, Nadam) допомагають уникнути великих коливань під час оновлення ваг, особливо у випадках складних або нерівних поверхонь функції втрат. Це дозволяє отримати стабільніші результати та уникнути переобчислень. Адаптивні оптимізатори менш чутливі до початкового значення швидкості навчання, що спрощує налаштування і дозволяє отримати кращі результати навіть без ретельного підбору гіперпараметрів.

Оптимізатори є ключовими елементами навчання нейронної мережі, оскільки від них залежить, наскільки ефективно і точно модель навчиться вирішувати поставлене завдання. Дослідивши поширені оптимізатори, такі як SGD (Stochastic Gradient Descent), Momentum, AdaGrad, RMSprop, Adam, було вибрано останній завдяки своїй здатності адаптувати швидкість навчання для кожного параметра та ефективно поєднувати властивості кількох інших оптимізаторів.

Adam автоматично змінює швидкість навчання для кожного параметра, що допомагає моделі швидше знаходити оптимальні значення. Це особливо корисно, коли різні параметри мають різні масштаби або коли навчання проходить на складній поверхні функції втрат. Adam менш чутливий до початкового значення швидкості навчання порівняно з традиційними методами, такими як SGD. Це означає, що Adam може добре працювати з типовими значеннями гіперпараметрів (наприклад, часто рекомендують швидкість навчання 0.001) і не вимагає тривалого налаштування.

Adam добре працює з великими та розрідженими даними, що може бути корисним для задач, де дані мають високу розмірність або є розрідженими (наприклад, у текстових або рекомендаційних системах). Завдяки популярності Adam є численні дослідження та реалізації, які дозволяють легко використовувати та адаптувати його для специфічних завдань. Adam підходить для широкого кола задач глибокого навчання, від комп'ютерного зору до обробки природної мови та рекомендаційних систем.

3.5 Результати виконання моделювання в середовищі JN

Виконаємо розробку нейронної мережі для розпізнавання зображень з використанням мови програмування Python.

```
(x_train_org, y_train_org), (x_test_org, y_test_org) = mnist.load_data()
```

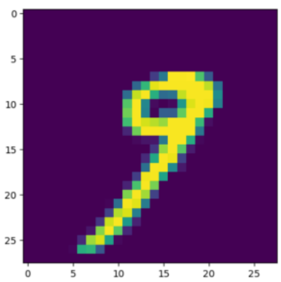
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 23s 2us/step
```

```
x_train_org[0]
```

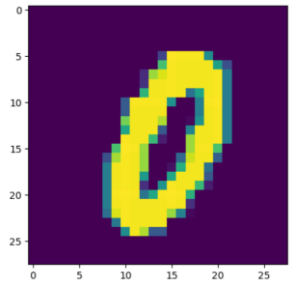
```
array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   3,
         18,  18,  18, 126, 136, 175,  26, 166, 255, 247, 127,   0,   0,
          0,   0],
```

…

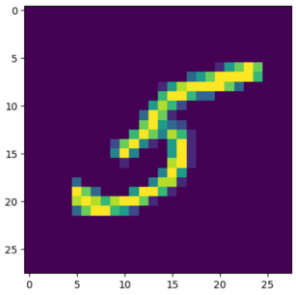
```
n = 33
plt.imshow(x_train_org[n], cmap='viridis')
plt.show()
```



```
n = 34
plt.imshow(x_train_org[n], cmap='viridis')
plt.show()
```



```
n = 35
plt.imshow(x_train_org[n], cmap='viridis')
plt.show()
```



```
x_train = x_train_org.reshape(60000, 784)
x_test = x_test_org.reshape(10000, 784)
print(x_train_org.reshape)
print(x_train.shape)
```

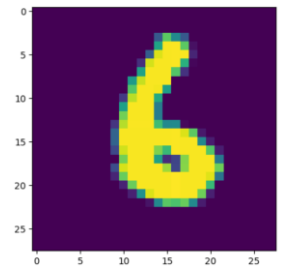
```
<built-in method reshape of numpy.ndarray object at 0x000001BBA28E7270>
(60000, 784)
```

```
x_train = x_train.astype('float32')
x_train = x_train / 255
x_test = x_test.astype('float32')
x_test = x_test / 255
```

```
y_train_org[35]
```

```
5
```

```
n = 36
plt.imshow(x_train_org[n], cmap='viridis')
plt.show()
```



```
y_train_org[36]
```

```
6
```

Датасет для нейронних мереж, що займаються розпізнаванням зображень, зазвичай організований таким чином, щоб забезпечити ефективне навчання моделі за допомогою структурованих та маркованих даних. Основні компоненти таких датасетів включають зображення та відповідні мітки (категорії або класи), а також можуть містити додаткову інформацію, таку як анотації або метадані.

Правильна організація датасету є критично важливою для успішного навчання нейронної мережі, оскільки від цього залежить, як добре модель зможе узагальнити інформацію та правильно класифікувати або детектувати об'єкти на нових зображеннях.

```
utils.to_categorical(y_train_org[0], 10)
```

```
array([0., 0., 0., 0., 0., 1., 0., 0., 0., 0.], dtype=float32)
```

```
y_train = utils.to_categorical(y_train_org, 10)
y_test = utils.to_categorical(y_test_org, 10)
```

```
print(y_train.shape)
```

```
(60000, 10)
```

```
print(y_train[n])
```

```
[0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
```

```
print(y_train_org.shape)
```

```
(60000,)
```

```
print(y_train_org[36])
```

```
6
```

```
model = Sequential()
model.add(Dense(800, input_dim = 784, activation = 'relu'))
model.add(Dense(400, activation = 'relu'))
model.add(Dense(10, activation = 'softmax'))
```

Sequential - тип моделі, де шари додаються один за одним. Це найпростіший тип моделі в Keras, який передбачає, що кожен шар передає свій вихід наступному шару у послідовності. *input_dim = 784* - параметр, який вказує розмір вхідного вектору. У цьому випадку вхідний вектор має розмір 784, що може бути розміром вектора для зображення 28x28 пікселів (наприклад, зображення із

схожими характеристиками до тих, що використовуються у наборі даних MNIST, де кожне зображення є чорно-білим з розміром 28x28). *activation = 'relu'* - функція активації для цього шару.

Виконаємо опис архітектури нейронної мережі. Вхідний вектор має розмірність 784 (це може бути зображення розміру 28x28 пікселів, переведене в одновимірний масив). Перший прихований шар містить 800 нейронів і використовує активацію ReLU. Другий иприхований шар містить 400 нейронів, також з активацією ReLU. Вихідний шар містить 10 нейронів і використовує активацію softmax, що перетворює вихід у ймовірності класів.

Компіляція моделі нейронної мережі є важливим етапом, що готує модель до навчання. Після того, як створена архітектура моделі, компіляція визначає, як модель буде навчатися, і налаштовує всі необхідні компоненти для процесу тренування.

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_11 (Dense)            (None, 800)               628000

 dense_12 (Dense)            (None, 400)               320400

 dense_13 (Dense)            (None, 10)                4010

=================================================================
Total params: 952410 (3.63 MB)
Trainable params: 952410 (3.63 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

Запускаємо навчання моделі нейронної мережі.

```
model.fit(x_train, y_train, batch_size=128, epochs=15, verbose=1, validation_split=0.2)
```

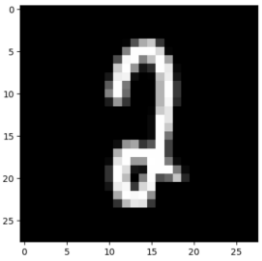
```
Epoch 1/15
375/375 [==============================] - 10s 22ms/step - loss: 0.2274 - accuracy: 0.9336 - val_loss: 0.1092 - val_accuracy:
0.9667
Epoch 2/15
375/375 [==============================] - 8s 21ms/step - loss: 0.0838 - accuracy: 0.9748 - val_loss: 0.0844 - val_accuracy: 0.
9730
Epoch 3/15
375/375 [==============================] - 8s 21ms/step - loss: 0.0496 - accuracy: 0.9844 - val_loss: 0.0836 - val_accuracy: 0.
9750
Epoch 4/15
375/375 [==============================] - 8s 21ms/step - loss: 0.0352 - accuracy: 0.9887 - val_loss: 0.0834 - val_accuracy: 0.
9766
```

```
Epoch 5/15
375/375 [==============================] - 8s 21ms/step - loss: 0.0260 - accuracy: 0.9918 - val_loss: 0.0840 - val_accuracy: 0.
9771
Epoch 6/15
375/375 [==============================] - 8s 21ms/step - loss: 0.0239 - accuracy: 0.9922 - val_loss: 0.0779 - val_accuracy: 0.
9791
Epoch 7/15
375/375 [==============================] - 8s 21ms/step - loss: 0.0164 - accuracy: 0.9946 - val_loss: 0.0861 - val_accuracy: 0.
9787
Epoch 8/15
375/375 [==============================] - 8s 21ms/step - loss: 0.0157 - accuracy: 0.9948 - val_loss: 0.1061 - val_accuracy: 0.
9745
Epoch 9/15
375/375 [==============================] - 8s 21ms/step - loss: 0.0161 - accuracy: 0.9947 - val_loss: 0.0991 - val_accuracy: 0.
9773
Epoch 10/15
375/375 [==============================] - 8s 21ms/step - loss: 0.0118 - accuracy: 0.9962 - val_loss: 0.1018 - val_accuracy: 0.
9783


Epoch 11/15
375/375 [==============================] - 8s 21ms/step - loss: 0.0134 - accuracy: 0.9953 - val_loss: 0.1282 - val_accuracy: 0.
9746
Epoch 12/15
375/375 [==============================] - 8s 21ms/step - loss: 0.0090 - accuracy: 0.9973 - val_loss: 0.1093 - val_accuracy: 0.
9783
Epoch 13/15
375/375 [==============================] - 8s 21ms/step - loss: 0.0148 - accuracy: 0.9954 - val_loss: 0.1002 - val_accuracy: 0.
9797
Epoch 14/15
375/375 [==============================] - 8s 20ms/step - loss: 0.0103 - accuracy: 0.9970 - val_loss: 0.1096 - val_accuracy: 0.
9770
Epoch 15/15
375/375 [==============================] - 8s 21ms/step - loss: 0.0058 - accuracy: 0.9981 - val_loss: 0.0977 - val_accuracy: 0.
9802

<keras.src.callbacks.History at 0x1bba5f02b00>
```

```
n_rec = 1356
plt.imshow(Image.fromarray(x_test_org[n_rec]).convert('RGBA'))
plt.show()
```




```
x = x_test[n_rec]
x = np.expand_dims(x, axis=0)
```

Функція *predict()* у нейронній мережі виконує роль виконання передбачень на основі навчених ваг і параметрів моделі. Вона не є частиною процесу навчання, але важлива для оцінки продуктивності моделі та для отримання результатів після навчання. Функція predict() використовується для отримання результатів від моделі після її тренування. Вона приймає вхідні дані (наприклад, зображення, текст або інші вхідні ознаки) і генерує відповідні виходи.

```
prediction = model.predict(x)
```

```
WARNING:tensorflow:5 out of the last 5 calls to <function Model.make_predict_function.<locals>.predict_function at 0x000001BBA9
2D2170> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating
@tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. Fo
r (1), please define your @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=True option that can avo
id unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://
www.tensorflow.org/api_docs/python/tf/function for  more details.
1/1 [==============================] - 0s 107ms/step
```

Результат виконання функції predict():

```
print(prediction)
```

```
[[1.33180865e-05 9.80238113e-08 9.99731481e-01 3.38007267e-05
  1.11515886e-13 4.26566205e-09 4.46896875e-09 9.98641735e-06
  2.11305320e-04 3.27115487e-11]]
```

```
np.argmax(prediction)
```

```
2
```

```
prediction = model.predict(x)
```

```
1/1 [==============================] - 0s 35ms/step
```

```
np.argmax(prediction)
```

```
2
```

Однією з ключових переваг нейронних мереж є їх здатність автоматично виявляти важливі характеристики та ознаки в зображеннях без необхідності вручну визначати специфічні риси, як це було у традиційних методах комп'ютерного зору.

Зображення мають високу просторову кореляцію між пікселями, що означає, що пікселі, які знаходяться поруч, зазвичай мають схожі значення. Нейронні мережі добре справляються з варіаціями в зображеннях, такими як зміщення, обертання, зміна масштабу, освітлення або кольору. Завдяки своїй здатності вчити абстрактні ознаки, мережі можуть узагальнювати інформацію і правильно класифікувати об'єкти навіть за змінних умов.

Завдяки спеціалізованим апаратним засобам, таким як графічні процесори (GPU), нейронні мережі здатні обробляти великі масиви даних за дуже короткий час. Це дозволяє швидко навчати моделі і проводити передбачення навіть на великих наборах зображень.

ВИСНОВКИ

У роботі було розроблено нейронну мережу для розпізнавання зображень, яка ефективно виконує поставлені завдання. Рівень точності моделі, якого вдалося досягти, змінюється від 0.9216 до 0.9973. Рівень точності моделі — важливий показник, але інші метрики також можуть бути критичними залежно від типу задачі. Вибір метрики слід обґрунтувати, враховуючи особливості даних та мету моделі. Вибрано метрику accuracy, яка легко і швидко розраховується, тому її доцільно використовувати як основний критерій для початкової оцінки моделі, особливо на етапах розробки та тестування.

Показники функції втрат, яка вимірює різницю між передбаченнями моделі та фактичними (цільовими) значеннями, змінювалися від 0.24 до 0.0107, що свідчить про непоганий результат роботи моделі.

Проведені експерименти показали, що запропонована нейронна мережа демонструє високу точність у розпізнаванні зображень, однак можливі покращення. Наприклад, вказати, що додаткове налаштування гіперпараметрів або збільшення обсягу тренувальної вибірки може ще більше покращити результат.

Робота також стикається з деякими обмеженнями, такими як потреба у значних обчислювальних ресурсах для тренування глибоких моделей, потреба у великій кількості навчальних даних. Модель, розроблена в межах роботи, може бути використана для різних прикладних завдань, таких як розпізнавання об’єктів у реальному часі, аналіз зображень у медичній діагностиці.

Доцільно використовувати Python та бібліотеки TensorFlow, Keras, NumPy, Matplotlib для реалізації таких моделей, так як вони мають глибокий функціонал для навчання, високу продуктивність, гнучкість у процесі досліджень, велику спільноту та активну підтримку, інструменти для відстеження та оптимізації процесу навчання. TensorFlow підходить як для дослідників, так і для інженерів, і дозволяє ефективно використовувати машинне навчання на практиці.

ПЕРЕЛІК ПОСИЛАНЬ

1. http://realpython.com/python3-object-oriented-programming/
2. http://leetcode.com/problemset/all/
3. http://perso.limsi.fr/pointal/
4. http://media/python:cours:mementopython3-english.pdf
5. https://numpy.org/
6. https://matplotlib.org/
7. https://pytorch.org/
8. https://www.tensorflow.org/
9. Jake VanderPlas Python Data Science Handbook: Essential Tools for Working with Data, 2022, 551p.
10. Heller, Nicholas et al. (2020). The KiTS19 Challenge Data: 300 Kidney Tumor Cases with Clinical Context, CT Semantic Segmentations, and Surgical Outcomes. arXiv: 1904. 00445 [q-bio.QM].
11. Hollo, Kaspar (2019). Exploring the Value of Weakly-Supervised Deep Learning Approaches for Artefact Segmentation in Brightfield Microscopy Images. URL: https://comserv. cs . ut . ee / home / files / hollo _ softwareengineering _ 2021.
12. Wang, Haofan et al. (2020). Score-CAM: Score-Weighted Visual Explanations for Convo-lutional Neural Networks. arXiv: 1910.01279 [cs.CV].
13. Yang, Guanyu et al. (2020). “Weakly-supervised convolutional neural networks ofrenal tumor segmentation in abdominal CTA images”. In: BMC Medical Imaging 20.1, p. 37. ISSN: 1471-2342. DOI: 10.1186/s12880- 020- 00435- w. URL: https://doi.org/10.1186/s12880-020-00435-w.
14. Mozaffari, M., Saad, W., Bennis, M., Nam, Y.-H., Debbah, M. A tutorial on UAVs for wireless networks: Applications challenges and open problems. IEEE Commun. Surveys Tuts., 21(3), 2019. — 2334-2360 c.
15. Zhang, J., Zhou, L., Tang, Q., Ngai, E. C.-H., Hu, X., Zhao, H., et al. Stochastic computation offloading and trajectory scheduling for UAV-assisted mobile edge computing. IEEE Internet Things J., 6(2), 2019. — 3688-3699 c.

16. Zhou, F., Hu, R. Q., Li, Z., Wang, Y. Mobile edge computing in unmanned aerial vehicle networks. IEEE Wireless Commun., 27(1), 2020. — 140-146 с.

17. Mishra, D., Natalizio, E. A survey on cellular-connected UAVs: Design challenges enabling 5G/B5G innovations and experimental advancements. Comput. Netw., 182, Dec. 2020.

18. Hua, M., Wang, Y., Li, C., Huang, Y., Yang, L. UAV-aided mobile edge computing systems with one by one access scheme. IEEE Trans. Green Commun. Netw., 3(3), 2019. — 664-678 с.

19. Multiple Access Schemes for Cellular Systems. Electronic Notes, Oct. 2020. — [Електронний ресурс]. Режим доступу: https://www.electronics-notes.com/articles/connectivity/cellular-mobile-phone/multiple-access-schemes-technology-techniques.php

20. Yao, H., Mai, T., Jiang, C., Kuang, L., Guo, S. AI Routers & Network Mind: A Hybrid Machine Learning Paradigm for Packet Routing. IEEE Computational Intelligence Magazine, 14(4), 2019. — 21-30 с.

21. Li, L., Wen, X., Lu, Z., Pan, Q., Hu, W. J. A. Z. Energy-efficient UAV-enabled MEC system: Bits allocation optimization and trajectory design. Sensors, 19(20), 2019. — 4521 с.

22. Wu, G., Miao, Y., Zhang, Y., Barnawi, A. Energy efficient for UAV-enabled mobile edge computing networks: Intelligent task prediction and offloading. Comput. Commun., 150, Jan. 2020. — 556-562 с.

23. What’s the Difference Between FDD and TDD. Electronic Design, Oct. 2020. — [Електронний ресурс]. Режим доступу: https://www.electronicdesign.com/technologies/communications/article/21801257/whats-the-difference-between-fdd-and-tdd

24. Selvaraju, Ramprasaath R. et al. (2019). “Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization”. In: International Journal of ComputerVision 128.2, pp. 336–359. DOI: 10 . 1007 / s11263 - 019 - 01228 - 7. URL: https://doi.org/10.1007\%2Fs11263-019-01228-7.

25. Грас Д. Data Science. Наука про дані з нуля: Пер. з англ. – 2-е видання., перероб. та доп. – 2021. – 416 с.

26. Лі, Кай-Фу Штучний інтелект: 10 передбачень для майбутнього / Кай-Фу Лі, Чень Цюфань; пер. з англ. А. Райчук. – Київ: Форс Україна, 2022. – 464 с.

27. Зінченко О.В., Фесенко М.А., Березівський М.Ю., Кисіль Т.М. Технології смарт-систем. – Навчальний посібник. – К.: ДУІКТ, 2023. – 162 с.

28. Золотухіна О.А., Негоденко О.В., Резник С.Ю., Разіна С.Я. Якість та тестування інформаційних систем. Навчальний посібник підготовлено до друку для самостійної роботи студентів вищих навчальних закладів. Київ: ННІІТ ДУТ, 2020. – 128 с.

29. Ю. В. Нікольський, В. В. Пасічник, Ю. М. Щербина Системи штучного інтелекту: навчальний посібник. – Львів: «Магнолія-2006», 2024. – 279 с.

30. Лосєв М.Ю. Програмування мовою Python: навчальний посібник / М. Ю. Лосєв, В. М. Федорченко. – Харків, - Львів: Видавництво ПП «Новий Світ - 2000», 2023. – 178 с.

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

(Презентація)