

МІНІСТЕРСТВО ФІНАНСІВ УКРАЇНИ  
ДЕРЖАВНИЙ ПОДАТКОВИЙ УНІВЕРСИТЕТ

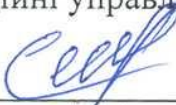
ФАКУЛЬТЕТ ФІНАНСІВ ТА ЦИФРОВИХ ТЕХНОЛОГІЙ  
КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І  
СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

**Застосування штучних нейронних мереж для розв'язання прикладних задач**

Здобувача вищої освіти другого (магістерського) рівня,  
II курсу, групи КМІ-23-1, спеціальності 122 «Комп'ютерні науки»,  
освітня програма «Інформаційні управляючі системи і технології в економіці»

  
(підпис)

Дзями Євгенія  
Олександровича

Науковий керівник

  
(підпис)

Ніжегородцев  
Владислав  
Олександрович, к.пед.н.,  
доцент

Керівник освітньо-  
професійної програми

  
(підпис)

Горбовий Артур Юліанович,  
д.т.н, професор  
(прізвище, ім'я, по батькові, науковий  
ступінь, вчене звання)

Допущено до захисту за рішенням кафедри протокол від «10» січня 2025 №11

Завідувач кафедри

  
(підпис)

Філоненко Михайло  
Миколайович, к.ф.-м.н.,  
доцент  
(прізвище, ім'я, по батькові,  
науковий ступінь, вчене звання)

Ірпінь – 2025

Реєстрація № 10

27.01.2025

(дата)



(підпис відповідальної особи)

Результати перевірки

до захисту

(до захисту, на доопрацювання)

27.01.2025

(дата)



(підпис)

Результат захисту

3 (задовільно)

(оцінка)

06.02.2025

(дата)



(підпис)

Голова комісії:



(підпис)

В.М. Франчук

Члени комісії:



(підпис)

А.Ю. Горбовий



(підпис)

М.М. Філоненко



(підпис)

І.Д. Погорєловська

# БЛАНК ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

| Об'єкти контролю                                                                                                                                                                     | Діапазон оцінки в балах | Фактична кількість балів | Примітки | Дата, П.І.Б.: керівника, голови комісії членів комісії, секретаря, (підписи)                                                                                           |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------|--------------------------|----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ КОМІСІЄЮ ПО ЗАХИСТУ                                                                                                                                |                         |                          |          |                                                                                                                                                                        |
| 1. Зміст кваліфікаційної роботи (якість теоретичного матеріалу, його зв'язок з практичним, вміння представляти матеріал в логічній послідовності, наявність висновків та пропозицій) | від 0 до 30             | 18                       |          | 06.02.2025<br>Франчук В.М. <i>В.М.</i><br>Горбовий А.Ю. <i>А.Ю.</i><br>Філоненко М.М. <i>М.М.</i><br>Погореловська І.Д. <i>І.Д.</i><br>Суховерхівська Л.П. <i>Л.П.</i> |
| 2. Дотримання вимог щодо оформлення                                                                                                                                                  | від 0 до 10             | 7                        |          |                                                                                                                                                                        |
| 3. Вчасне виконання роботи на всіх етапах у встановлені кафедрою терміни                                                                                                             | від 0 до 5              | 4                        |          |                                                                                                                                                                        |
| 4. Рецензент                                                                                                                                                                         | від 0 до 5              | 4                        |          |                                                                                                                                                                        |
| РАЗОМ БАЛІВ                                                                                                                                                                          | від 0 до 50             | 33                       |          |                                                                                                                                                                        |

Оцінювання кваліфікаційної роботи науковим керівником - 34 балів.

## ПЕРЕВЕДЕННЯ ОЦІНКИ ЗА 100-БАЛЬНОЮ ШКАЛОЮ ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

| Оцінка за 100-бальною шкалою | Оцінка за шкалою ЄКТС | Оцінка за національною шкалою | Дата       | П.І.Б.: голови, членів комісії, секретаря, (підпис)                                                                                                      |
|------------------------------|-----------------------|-------------------------------|------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| 67                           | D                     | 3 (задовільно)                | 06.02.2025 | Франчук В.М. <i>В.М.</i><br>Горбовий А.Ю. <i>А.Ю.</i><br>Філоненко М.М. <i>М.М.</i><br>Погореловська І.Д. <i>І.Д.</i><br>Суховерхівська Л.П. <i>Л.П.</i> |

## ПРАВИЛА ПЕРЕВЕДЕННЯ ОЦІНОК ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

| Оцінка за 100-бальною шкалою | Оцінка за шкалою ЄКТС | Оцінка за національною шкалою |
|------------------------------|-----------------------|-------------------------------|
| 90-100                       | A                     | 5 (відмінно)                  |
| 80-89                        | B                     | 4 (добре)                     |
| 70-79                        | C                     |                               |
| 60-69                        | D                     |                               |
| 50-59                        | E                     | 3 (задовільно)                |
| 35-49                        | FX                    | 2 (незадовільно)              |

Факультет фінансів та цифрових технологій

Кафедра комп'ютерних та інформаційних технологій і систем

Галузь знань 12 «Інформаційні технології»

Спеціальність 122 «Комп'ютерні науки»

Освітня програма «Інформаційні управляючі системи і технології в економіці»

**Затверджую**

Завідувач кафедри  Філоненко М.М.

«23» грудня 2023

**Завдання на кваліфікаційну роботу**

**здобувачу вищої освіти**

Дзямі Євгенію Олександровичу

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи: Застосування штучних нейронних мереж для розв'язання прикладних задач

Затверджена наказом ректора від «20» грудня 2023 № 1701

2. Строк здачі закінченої роботи «7» січня 2025

3. Мета роботи: Описати особливості застосування штучних нейронних мереж для розв'язання прикладних задач

Об'єкт дослідження: Об'єктом дослідження є процес використання штучних нейронних мереж для розв'язання прикладних задач

Предмет дослідження: Предметом дослідження є ефективність та практичність використання штучних мереж в контексті розв'язання прикладних задач

Орієнтовна база дослідження: Штучна нейронна мережа YOLOV3 для детектування, трекінгу та підрахунку

4. Вихідні дані до роботи: Робота з нейронами та тренування нейронної мережі

5. Перелік графічного матеріалу: Рис.3.2, 3.3, 3.4 (Приклади роботи нейронної мережі)

6. Зміст кваліфікаційної роботи (перелік питань за кожним розділом)

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ЗАСТОСУВАННЯ ШТУЧНИХ НЕЙРОННИХ МЕРЕЖ ДЛЯ РОЗВ'ЯЗАННЯ ПРИКЛАДНИХ ЗАДАЧ

1.1. Класифікація задач та моделі Data Mining

1.2. Нейронні мережі як спосіб видобутку знань

1.3. Завдання чіткого поділу двох класів. Математичний апарат

## РОЗДІЛ 2. ТЕОРЕТИЧНІ АСПЕКТИ ДЛЯ ЗАСТОСУВАННЯ ШТУЧНИХ НЕЙРОННИХ МЕРЕЖ

2.1. Використання бібліотеки Keras для тренування простої нейронної мережі. Створення нейронних мереж та блоків

2.2. Приклади роботи з нейронами

2.3. Приклади тренування нейронної мережі

## РОЗДІЛ 3. ПРАКТИЧНЕ ЗАСТОСУВАННЯ ШТУЧНОЇ НЕЙРОННОЇ МЕРЕЖІ YOLOv3 ДЛЯ ДЕТЕКТУВАННЯ, ТРЕКІНГУ І ПІДРАХУНКУ ПІШОХОДІВ У РЕЖИМІ РЕАЛЬНОГО ЧАСУ

3.1. Алгоритми навчання та тестування нейронної мережі YOLOv3

3.2. Результати навчання, отримані метрики

7. Календарний план виконання роботи

| № п/п | Назва етапів кваліфікаційної роботи                                                                                                                                                                  | Строк виконання етапів роботи |            |
|-------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------|------------|
|       |                                                                                                                                                                                                      | За планом                     | Фактично   |
| 1     | 2                                                                                                                                                                                                    | 3                             | 4          |
| 1     | Вибір теми, підбір літературних джерел, формування структури кваліфікаційної роботи                                                                                                                  | До 15.12.2023                 | 15.12.2023 |
| 2     | <b>I розділ</b> (розкрити сутність і стан дослідження наукової проблематики роботи, після чого навести її загальну характеристику; огляд літератури за темою і вибір конкретних напрямів досліджень) | 27.05.2024                    | 28.05.2024 |
| 3     | <b>II розділ</b> (аналіз експериментальної частини про методики досліджень)                                                                                                                          | 02.09.2024                    | 04.09.2024 |
| 4     | <b>III розділ</b> (проведені теоретичні і (або) експериментальні дослідження та проектні рішення)                                                                                                    | 18.11.2024                    | 23.11.2024 |
| 5     | <b>Висновки</b>                                                                                                                                                                                      | 10.12.2024                    | 08.12.2024 |
| 6     | Доопрацювання роботи з урахуванням зауважень наукового керівника                                                                                                                                     | 23.12.2024                    | 23.12.2024 |
| 7     | Отримання відгуку керівника                                                                                                                                                                          | 28.01.2025                    | 25.01.2025 |
| 8     | Отримання зовнішньої рецензії                                                                                                                                                                        | 31.01.2025                    | 31.01.2025 |

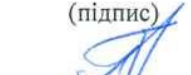
8. Дата видачі завдання «23» грудня 2023

9. Керівник кваліфікаційної роботи

  
(підпис)

Ніжегородцев В.  
(прізвище, ініціали)

10. Керівник освітньо-професійної програми

  
(підпис)

Горбовий А.Ю.  
(прізвище, ініціали)

11. Завдання до виконання отримав здобувач вищої освіти

  
(підпис)

Дзяма Є. О.  
(прізвище, ініціали)

## Зміст

|                                                                                                                                                        |    |
|--------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| ВСТУП .....                                                                                                                                            | 3  |
| РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ЗАСТОСУВАННЯ ШТУЧНИХ<br>НЕЙРОННИХ МЕРЕЖ ДЛЯ РОЗВ’ЯЗАННЯ ПРИКЛАДНИХ ЗАДАЧ .....                                             | 5  |
| 1.1. Класифікація задач та моделі Data Mining .....                                                                                                    | 5  |
| 1.2. Нейронні мережі як спосіб видобутку знань.....                                                                                                    | 8  |
| 1.3. Завдання чіткого поділу двох класів. Математичний апарат .....                                                                                    | 12 |
| РОЗДІЛ 2. ТЕОРЕТИЧНІ АСПЕКТИ ДЛЯ ЗАСТОСУВАННЯ ШТУЧНИХ<br>НЕЙРОННИХ МЕРЕЖ.....                                                                          | 19 |
| 2.1. Використання бібліотеки Keras для тренування простої нейронної мережі.<br>Створення нейронних мереж та блоків.....                                | 19 |
| 2.2. Приклади роботи з нейронами.....                                                                                                                  | 30 |
| 2.3. Приклади тренування нейронної мережі .....                                                                                                        | 35 |
| РОЗДІЛ 3. ПРАКТИЧНЕ ЗАСТОСУВАННЯ ШТУЧНОЇ НЕЙРОННОЇ МЕРЕЖІ<br>YOLOv3 ДЛЯ ДЕТЕКТУВАННЯ, ТРЕКІНГУ І ПІДРАХУНКУ ПІШОХОДІВ У<br>РЕЖИМІ РЕАЛЬНОГО ЧАСУ ..... | 44 |
| 3.1. Алгоритми навчання та тестування нейронної мережі YOLOv3 .....                                                                                    | 44 |
| 3.2. Результати навчання, отримані метрики .....                                                                                                       | 46 |
| ВИСНОВОК.....                                                                                                                                          | 52 |
| СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ .....                                                                                                                       | 54 |
| Додатки.....                                                                                                                                           | 58 |

## ВСТУП

Основне завдання аналітика є генерація гіпотез. Він вирішує її, ґрунтуючись на своїх знаннях та досвіді. Проте знання не тільки у людини, а й у накопичених даних, які піддаються аналізу. Такі знання часто називають прихованими, так як вони містяться в гігабайтах та терабайтах інформації, які людина не в змозі досліджувати самотійно. У зв'язку з цим існує висока ймовірність пропустити гіпотези, які можуть принести значну вигоду.

Вочевидь, що з виявлення прихованих знань необхідно застосовувати спеціальні методи автоматичного аналізу, з яких доводиться практично добувати знання з великого об'єму інформації. За цим напрямком міцно закріпився термін видобування даних або Data Mining. Класичне визначення цього терміна у 1996 р. зробив один із засновників цього напрямку П'ятецький-Шاپіро.

Data Mining – дослідження та виявлення «машиною» (алгоритмами, засобами штучного інтелекту) у сирих даних прихованих знань, які раніше не були відомі, нетривіальні, практично корисні, доступні для інтерпретації людиною.

Знання мають бути нові, раніше невідомі. Витрачені зусилля на відкриття знань, які вже відомі користувачеві, не окупаються. Тому цінність становлять саме нові, раніше невідомі знання.

Знання мають бути нетривіальними. Результати аналізу повинні відображати неочевидні, несподівані закономірності даних, що становлять так звані приховані знання. Результати, які можна отримати більш простими способами (наприклад, візуальним переглядом), не виправдовують залучення потужних методів Data Mining.

Знання мають бути практично корисними. Знайдені знання повинні бути застосовні, в тому числі і на нових даних, з досить високим рівнем достовірності. Корисність полягає в тому, щоб ці знання могли принести певний результат при їх застосуванні.

Знання мають бути доступними для розуміння людини. Знайдені закономірності мають бути логічно зрозумілі, інакше існує ймовірність, що вони є випадковими.

У Data Mining представлення отриманих знань служать моделі. Види моделей залежать від методів їх створення.

**Мета дослідження** – описати особливості застосування штучних нейронних мереж для розв’язання прикладних задач.

У рамках даної випускної кваліфікаційної роботи було поставлено такі **завдання:**

- розглянути класифікацію задач та моделі Data Mining;
- описати нейронні мережі як спосіб видобутку знань;
- визначити завдання чіткого поділу двох класів;
- навести приклади застосування штучних нейронних мереж для розв’язання прикладних задач;
- описати практичне застосування штучної нейронної мережі YOLOV3 для детектування, трекінгу і підрахунку пішоходів у режим реального часу.

## **РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ЗАСТОСУВАННЯ ШТУЧНИХ НЕЙРОННИХ МЕРЕЖ ДЛЯ РОЗВ’ЯЗАННЯ ПРИКЛАДНИХ ЗАДАЧ**

### **1.1. Класифікація задач та моделі Data Mining**

Методи Data Mining допомагають вирішити багато завдань, із якими стикається аналітик. З них основними є: класифікація, регресія, пошук асоціативних правил та кластеризація. Нижче наведено короткий опис основних завдань аналізу даних. Завдання класифікації зводиться до визначення класу об’єкта за його характеристиками. Слід зазначити, що у цій задачі безліч класів до яких може бути віднесений об’єкт, заздалегідь відомо. Завдання регресії, подібно до завдання класифікації, дозволяє визначити за відомими характеристиками об’єкта значення деякого його параметра. На відміну від завдання класифікації значенням параметра є не кінцева безліч класів, а безліч дійсних чисел [15].

При пошуку асоціативних правил метою є знаходження частих залежностей (або асоціацій) між об’єктами або подіями. Знайдені залежності видаються як правила і можна їх використовувати для кращого розуміння природи аналізованих даних, та передбачення появи подій. Завдання кластеризації полягає в пошуку незалежних груп (кластерів) та їх характеристик у всій кількості аналізованих даних. Вирішення цього завдання допомагає краще зрозуміти дані. Крім того, угруповання однорідних об’єктів дозволяє скоротити їх число, а отже, і полегшити аналіз.

Перелічені завдання призначення діляться на описові і передбачувальні [6].

Описові (descriptive) завдання приділяють увагу поліпшенню розуміння аналізованих даних. Ключовий момент у таких моделях – легкість і прозорість результатів для сприйняття людиною, і тому мають бути відомими. До такого виду завдань належать кластеризація та пошук асоціативних правил [7].

Вирішення передбачуваних (predictive) завдань розбивається на два етапи. На першому етапі на підставі набору даних з відомими результатами будується модель, до якої відносять задачі класифікації та регресії деяких подій.

За способами вирішення завдання поділяють на supervised learning (навчання з вчителем) і unsupervised learning (навчання без вчителя). У разі supervised learning завдання для аналізу даних вирішується в кілька етапів. Спочатку за допомогою будь-якого алгоритму Data Mining будується модель аналізованих даних – класифікатор. Потім класифікатор піддається навчанню. Іншими словами, перевіряється якість його роботи і, якщо вона не задовільна, відбувається додаткове навчання класифікатора. Так триває до того часу, поки не буде досягнуто необхідний рівень якості або стане ясно, що обраний алгоритм не працює коректно з даними, або ж самі дані не мають структури, яку можна виявити. До цього типу завдань відносять завдання класифікації та регресії [3].

Unsupervised learning поєднує завдання, які виявляють описові моделі, наприклад закономірності в покупках, які здійснюються клієнтами великого магазину. Очевидно, що якщо ці закономірності є, то модель має їх подати і недоречно говорити про її навчання. Перевагою таких завдань є можливість їх вирішення без будь-яких попередніх знань про аналізовані дані. До них належать завдання кластеризації та пошуку асоціативних правил.

Мета технології Data Mining – знаходження даних таких моделей, які можуть бути знайдені звичайними методами. Існують два види моделей: передбачувальні та описові [12].

Передбачувальні моделі будуються на підставі вибору даних із відомими результатами. Вони використовуються для прогнозування результатів на основі інших наборів даних. При цьому, природно, потрібно, щоб модель працювала максимально точно, була статистично значущою і виправданою[22].

До них належать такі моделі:

- класифікації – описують правила або набір правил, відповідно до яких

можна віднести опис будь-якого нового об'єкта до одного з класів. Такі правила будуються на основі інформації про існуючі об'єкти шляхом розбиття їх на класи[9];

- послідовностей – описують функції, що дозволяють прогнозувати зміну безперервних числових параметрів. Вони будуються на підставі даних про зміну деякого параметра за минулий період часу[9].

Описові моделі приділяють увагу суті залежностей у наборі даних та взаємному впливу різних чинників, тобто при побудові емпіричних моделей різних систем.

До них належать такі види моделей:

- регресійні – описують функціональні залежності між залежними та незалежними показниками та змінними у зрозумілій людині формі. Слід зазначити, що такі моделі описують функціональну залежність як між безперервними числовими параметрами, так і між категоріальними[16];

- кластеризації – описують групи (кластери), на які можна розділити об'єкти, дані про які піддаються аналізу. Грукуються об'єкти (спостереження, події) з урахуванням даних (властивостей), що описують сутність об'єктів. Об'єкти всередині кластера мають бути «схожими» один на одного і відрізнятися від об'єктів, що увійшли до інших кластерів. Чим більше схожі об'єкти всередині кластера і чим більше відмінностей між кластерами, тим точніше кластеризація[16];

- моделі винятків – описують виняткові ситуації у записах (наприклад, окремих пацієнтів), які різко відрізняються чимось від основної множини записів (групи хворих). Навіть сама ідентифікація цих записів, не кажучи про їх подальший аналіз та детальний розгляд, може виявитися дуже корисною для розуміння сутності об'єктів, що вивчаються[16];

- підсумкові – виявлення обмежень на дані аналізованого масиву. Наприклад, при вивченні вибірки даних пацієнтам не старше 30 років, які перенесли інфаркт міокарда, виявляється, що всі пацієнти, описані в цій

вибірці, або курять більше 5 пачок сигарет на день, або мають вагу не нижче 95 кг. Подібні обмеження є важливими для розуміння даних масиву; по суті справи – це нове знання, витягнуте в результаті аналізу. Таким чином, Data Summarization – це знаходження будь-яких фактів, які вірні для всіх або майже всіх записів в добірці даних, що вивчається, але які досить рідко зустрічалися б у всьому мислимому різноманітті записів такого ж формату і, наприклад, характеризувалися тими самими розподілами значень полів[16];

– асоціації – виявлення закономірностей між пов’язаними подіями. Прикладом такої закономірності служить правило, що вказує, що з події X випливає подія Y. Такі правила називаються асоціативними[16].

Для побудови розглянутих моделей використовуються різні методи та алгоритми Data Mining. Зважаючи на те, що технологія Data Mining розвивалася і розвивається на стику таких дисциплін, як статистика, теорія інформації, машинне навчання, теорія баз даних, цілком закономірно, що більшість алгоритмів та методів Data Mining були розроблені на основі різних технологій та концепцій. Однією з технологій, що часто реалізується методами Data Mining є технологія нейронних мереж.

## **1.2. Нейронні мережі як спосіб видобутку знань**

Штучні нейронні мережі це клас моделей, заснованих на біологічній аналогії з мозком людини та призначених після проходження етапу так званого навчання на наявних даних для вирішення різноманітних завдань аналізу даних. При застосуванні цих методів насамперед постає питання вибору конкретної архітектури мережі (числа «шарів» та кількості «нейронів» у кожному з них). Розмір та структура мережі повинні відповідати (наприклад, у сенсі формальної обчислювальної складності) суті досліджуваного явища. Оскільки на початковому етапі аналізу природа явища зазвичай відома погано, вибір архітектури є непростим завданням і часто пов’язаний із тривалим процесом

«проб і помилок» (проте останнім часом стали з'являтися нейронно-мережні програми, в яких для вирішення трудомісткої задачі пошуку найкращої архітектури мережі застосовуються методи штучного інтелекту) [3].

Потім побудована мережа піддається процесу навчання. На цьому етапі нейрони мережі ітеративно обробляють вхідні дані і коригують свої ваги так, щоб мережа найкращим чином прогнозувала дані, на яких виконується «навчання». Після навчання на наявних даних мережа готова до роботи та може використовуватись для побудови прогнозів.

Нейронна мережа, отримана в результаті навчання, виражає закономірності, присутні в даних. При такому підході вона виявляється функціональним еквівалентом деякої моделі залежностей між змінними, подібними до тих, що будуються в традиційному моделюванні. Однак, на відміну від традиційних моделей, у випадку нейронних мереж ці залежності не можуть бути записані в явному вигляді, подібно до того, як це робиться в статистиці (наприклад, «А позитивно корельовано для спостережень, у яких величина С мала, а D велика»). Іноді нейронні мережі видають прогноз дуже високої якості; однак вони є типовим прикладом нетеоретичного підходу до дослідження (іноді це називають «чорною скринькою»). При такому підході зосереджуються виключно на практичному результаті, в даному випадку на точності прогнозів та їх прикладної цінності, а не на суті механізмів, що лежать в основі явища або відповідності отриманих результатів будь-якої теорії [7].

Слід, однак, відзначити, що методи нейронних мереж можуть застосовуватися і в дослідженнях, спрямованих на побудову моделі, що пояснює явища, оскільки нейронні мережі допомагають вивчати дані з метою пошуку значущих змінних або груп таких змінних, і отримані результати можуть полегшити процес подальшої побудови моделі. Понад те, зараз є нейромережеві програми, які з допомогою складних алгоритмів можуть знаходити найважливіші вхідні змінні, що безпосередньо допомагає будувати модель [6].

Одна з головних переваг нейронних мереж полягає в тому, що вони

принаймні теоретично можуть апроксимувати будь-яку безперервну функцію, і тому досліднику немає необхідності заздалегідь приймати будь-які гіпотези щодо моделі і навіть у ряді випадків про те, які змінні дійсно важливі. Однак істотним недоліком нейронних мереж є та обставина, що остаточне рішення залежить від початкових установок мережі та, як уже зазначалося, його практично неможливо інтерпретувати у традиційних аналітичних термінах, які зазвичай застосовуються при побудові теорії явища.

Штучний нейрон імітує у першому наближенні властивості біологічного нейрона. На вхід штучного нейрона надходить кілька сигналів, кожен із яких є виходом іншого нейрона. Кожен вхід множиться на відповідну вагу, аналогічну синаптичній силі, і всі результати підсумовуються, визначаючи рівень активації нейрона. На рис.1.1 представлена модель, що реалізує цю ідею [14].

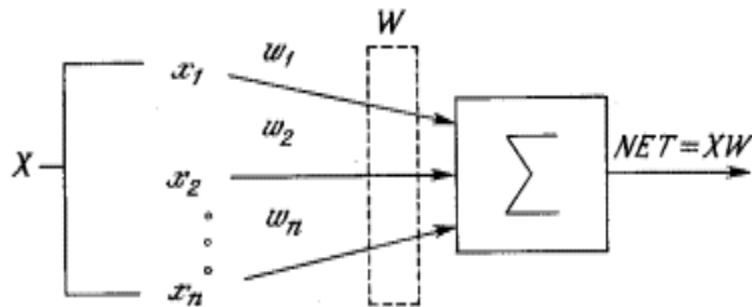


Рис. 1.1. Активаційні функції штучного нейрона

Хоча мережеві парадигми дуже різноманітні, в основі багатьох їх лежить ця конфігурація. Тут безліч вхідних сигналів, позначених  $x_1, x_2, \dots, x_n$  надходять на штучний нейрон. Ці вхідні сигнали, що сукупно позначаються вектором  $X$ , відповідають сигналам, що приходять в синапси біологічного нейрона. Кожен сигнал множиться на відповідну вагу  $w_1, w_2, \dots, w_n$ , і надходить на сумарний блок, позначений  $\Sigma$ . Кожна вага відповідає силі одного біологічного синаптичного зв'язку. Підсумовуючий блок, що відповідає тілу біологічного елемента, складає зважені входи алгебраїчно, створюючи вихід, який ми називатимемо  $NET$ . У векторних позначках це може бути компактно записано

таким чином:  $NET = XW$ .

Сигнал  $NET$  далі, як правило, перетворюється на активаційну функцію  $F$  і дає вихідний нейронний сигнал  $OUT$ . Активаційна функція може бути звичайною лінійною функцією [24].

$$OUT = K(NET),$$

де  $K$  – постійна, порогової функції

$$OUT = 1, \text{ якщо } NET > T,$$

$$OUT = 0 \text{ в інших випадках,}$$

$$OUT = \frac{1}{1 + e^{-NET}}$$

де  $T$  – деяка постійна порогова величина, або функція, що більш точно моделює нелінійну передатну характеристику біологічного нейрона і представляє нейронній мережі великі можливості.

В усьому різноманітті штучних нейронних мереж можна назвати дві базові архітектури – шаруваті і пов'язані мережі.

Шаруваті мережі: нейрони розташовані в кілька шарів. Нейрони першого шару одержують вхідні сигнали, перетворюють їх і через точки розгалуження передають нейронам другого шару. Далі спрацьовує другий шар і т. д. до  $k$ -го шару, що видає вихідні сигнали. Якщо обумовлено невідоме, кожен вихідний сигнал  $i$ -го шару подається на вхід всіх нейронів  $i + 1$ -го. Число нейронів у кожному шарі може бути будь-яким і ніяк не пов'язане з кількістю нейронів в інших шарах. Стандартний спосіб подачі вхідних сигналів: всі нейрони першого шару одержують кожен вхідний сигнал [21].

Повнозв'язкові мережі – кожен нейрон передає свій вихідний сигнал іншим нейронам, включаючи себе. Вихідними сигналами мережі можуть бути всі або деякі вихідні сигнали нейронів після кількох тактів функціонування

мережі. Усі вхідні сигнали подаються всім нейронам.

Елементи шаруватих та повнозв'язкових мереж можуть вибиратися по-різному. Існує, втім, стандартний вибір – нейрон з неоднорідним адаптивним лінійним суматором на вході (рис.1.2).

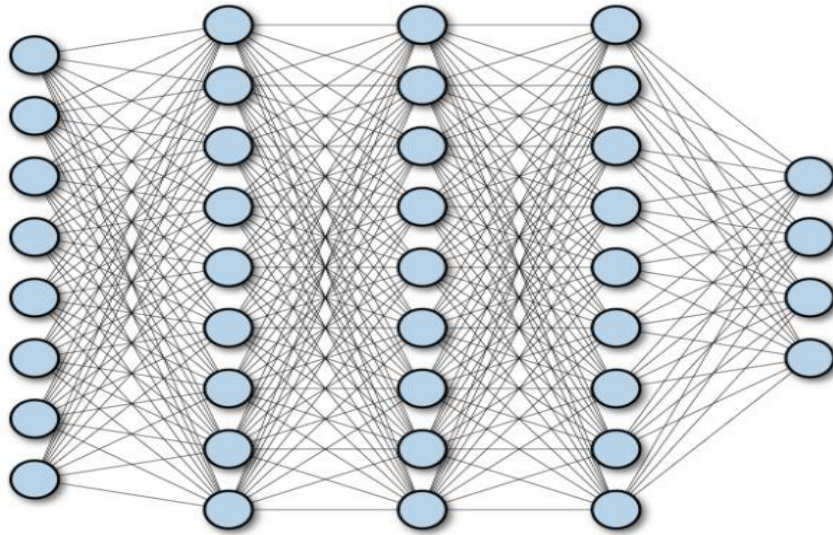


Рис.1.2. Схема повнозв'язкої нейронної мережі

### 1.3. Завдання чіткого поділу двох класів. Математичний апарат

Завдання чіткого поділу двох класів за навчальною вибіркою ставиться так: є два набори векторів  $x^1, \dots, x^m$  і  $y^1, \dots, y^m$ . Заздалегідь відомо, що  $x^i$  відноситься до першого класу, а  $y^i$  – до другого. Потрібно побудувати правило, тобто визначити таку функцію  $f(x)$ , що при  $f(x) > 0$  вектор  $x$  відноситься до першого класу, а при  $f(x) < 0$  до другого.

Координати класифікованих векторів є значеннями деяких ознак (властивостей) досліджуваних об'єктів. Це завдання виникає у багатьох випадках: при діагностиці хвороб та визначенні несправностей машин за непрямими ознаками, при розпізнаванні зображень та сигналів тощо [19].

Строго кажучи, класифікуються не вектори властивостей, а об'єкти, які мають ці властивості. Це зауваження стає важливим у тих випадках, коли виникають труднощі з побудовою вирішального правила – наприклад, тоді,

коли зустрічаються об'єкти, що належать до різних класів, мають однакові ознаки. У цих випадках можливі кілька рішень [19]:

- 1) шукати додаткові ознаки, що дозволяють розділити класи;
- 2) примиритися з неминучістю помилок, призначити за кожен тип помилок свій штраф ( $c_{12}$  – штраф за те, що об'єкт першого класу віднесено до другого,  $c_{21}$  – за те, що об'єкт другого класу віднесено до першого) та будувати розділяюче правило так, щоб мінімізувати математичне очікування штрафу;
- 3) перейти до нечіткого поділу класів – будувати так звані «функції належності»  $f_1(x)$  і  $f_2(x)$  оцінювати ступінь впевненості при віднесенні об'єкта до  $i$ -го класу ( $i=1,2$ ), для одного і того ж  $x$  може бути так, що  $f_1(x)>0$ , і  $f_2(x)>0$ .

Лінійний поділ класів полягає у побудові лінійного вирішального правила - тобто такого вектора  $\alpha$  та числа  $\alpha_0$  (називається порогом), що при  $(x, \alpha) > \alpha_0$   $x$  відноситься до першого класу, а при  $(x, \alpha) < \alpha_0$  – до другого.

Пошук такого вирішального правила можна розглядати як поділ класів у проекції на пряму. Вектор  $\alpha$  задає пряму, на яку ортогонально проектується всі точки, а число  $\alpha_0$  – точка на цій прямій, що відокремлює перший клас від другого.

Найпростіший і часом дуже зручний вибір полягає у проектуванні на пряму, що з'єднує центри мас вибірок. Центр мас обчислюється у припущенні, що маси всіх точок однакові та рівні 1. Це відповідає завданню  $\alpha$  у вигляді [23]:

$$\alpha = (y^1 + y^2 + \dots + y^m)/m \quad (x^1 + x^2 + \dots + x^n)/n \quad (1.1)$$

У багатьох випадках зручніше мати справу із векторами одиничної довжини. Нормуючи  $\alpha$ , отримуємо:

$$\alpha = ((y^1 + y^2 + \dots + y^m)/m - (x^1 + x^2 + \dots + x^n)/n) / \| (y^1 + y^2 + \dots + y^m)/m - (x^1 + x^2 + \dots + x^n)/n \| \quad (1.2)$$

Вибір  $\alpha_0$  може виконуватися з різних міркувань. Найпростіший варіант

посередині між центрами мас вибірок:

$$\alpha_0 = (((y^1 + y^2 + \dots + y^m)/m, \alpha) + ((x^1 + x^2 + \dots + x^n)/n, \alpha))/2 \quad (1.3)$$

Більш тонкі методи побудови кордону розділу класів  $\alpha_0$  враховують різні ймовірності появи об'єктів різних класів та оцінки щільності розподілу точок класів на прямій. Чим менша ймовірність появи даного класу, тим більше межа розділу наближається до центру тяжіння відповідної вибірки.

Можна кожному класу побудувати наближену щільність ймовірностей розподілу проекцій його точок на пряму (це набагато простіше, ніж багатомірного розподілу) і вибирати  $\alpha_0$  мінімізуючи ймовірність помилки. Нехай вирішальне правило має вигляд: при  $(x, \alpha) > \alpha_0 x$  відноситься до першого класу, а при  $(x, \alpha) < \alpha_0$  до другого. У такому разі ймовірність помилки дорівнюватиме

$$p = p_1 \int_{-\infty}^{\alpha} p_1(x) dx + p_2 \int_{\alpha_0}^{\infty} p_2(x) dx \quad (1.4)$$

де,  $p_1, p_2$  – апріорні ймовірності приналежності об'єкта відповідного класу,

$p_1(\chi), p_2(\chi)$  – щільності ймовірності для розподілу проекцій  $\chi$  точок  $x$  у кожному класі.

Прирівнявши похідну нулю ймовірності помилки по  $\alpha_0$ , отримаємо: число  $\alpha_0$ , що доставляє мінімум ймовірності помилки, є коренем рівняння:

$$p_1 p_1(\chi) = p_2 p_2(\chi), \quad (1.5)$$

або (якщо у цього рівняння немає рішень) оптимальним є правило, що відносить всі об'єкти до одного з класів.

Якщо прийняти гіпотезу про нормальність розподілів:

$$p(y) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(y-a)^2/2\sigma^2} \quad (1.6)$$

то для визначення  $\alpha_0$  отримаємо:

$$\frac{p_1}{\sqrt{2\pi}\sigma_1} e^{-(y-a_1)^2/2\sigma_1^2} = \frac{p_2}{\sqrt{2\pi}\sigma_2} e^{-(y-a_2)^2/2\sigma_2^2} \quad (1.8)$$

$$\ln\left(\frac{p_1}{p_2}\right) - \ln\left(\frac{\sigma_1}{\sigma_2}\right) - \frac{(y-a_1)^2}{2\sigma_1^2} + \frac{(y-a_2)^2}{2\sigma_2^2} = 0 \quad (1.9)$$

Якщо це рівняння має два корені  $y=\alpha_1, \alpha_2$ , ( $\alpha_1 < \alpha_2$ ) то найкращим вирішальним правилом буде: при  $\alpha_1 < (x, \alpha) < \alpha_2$  об'єкт належить одному класу, а при  $\alpha_1 > (x, \alpha)$  або  $(x, \alpha) > \alpha_2$  іншому (якому саме, визначається тим, який із результатів  $p_i p_i(\chi)$  більше). Якщо коріння немає, то оптимальним є віднесення до одного з класів. Випадок єдиного кореня становить інтерес лише тоді, коли  $\sigma_1 = \sigma_2$ . При цьому рівняння перетворюється на лінійне і ми приходимо до вихідного варіанту єдиної точки, що розділяє  $\alpha_0$ .

Таким чином, розділяюче правило з єдиною точкою, що розділяє  $\alpha_0$  не є найкращим для нормальних розподілів і треба шукати дві точки, що розділяють.

Якщо відразу ставити завдання про оптимальний поділ багатомірних нормальних розподілів, то отримаємо, що найкращою поверхнею, що розділяє, є квадрика. Припускаючи, що коваріаційні матриці класів збігаються (в одновимірному випадку це припущення про те, що  $\sigma_1 = \sigma_2$ ), отримуємо лінійну розділяючу поверхню:

$$(x, y) = x < \Sigma_1^{-1} y, \text{ де } \Sigma \text{ загальна підступна матриця класів.}$$

Важлива можливість удосконалити роздільне правило полягає у використанні оцінки не просто ймовірності помилки, а середнього ризику: кожній помилці приписується «ціна»  $c_i$  та мінімізується сума  $c_1 p_1 p_1(\chi) + c_2 p_2 p_2(\chi)$ . Відповідь виходить практично такою ж (усюди  $p_i$  замінюються на  $c_i p_i$ ), але така добавка важлива для багатьох додатків.

Вимога безпомилковості розділяючого правила на навчальній вибірці

принципово відрізняється від критеріїв оптимальності, що обговорювалися. На основі цієї вимоги будується персептрон Розенблатта «дідусь» сучасних нейронних мереж.

Однією з перших штучних мереж, здатних до перцепції (сприйняття) та формування реакції на сприйнятий стимул, став PERCEPTRON Розенблатта (F. Rosenblatt, 1957).

Найпростіший класичний персептрон містить нейроподібні елементи трьох типів (рис. 1.3). S-елементи формують сітківку сенсорних клітин, які приймають двійкові сигнали від зовнішнього світу. Далі сигнали надходять у шар асоціативних або А-елементів.

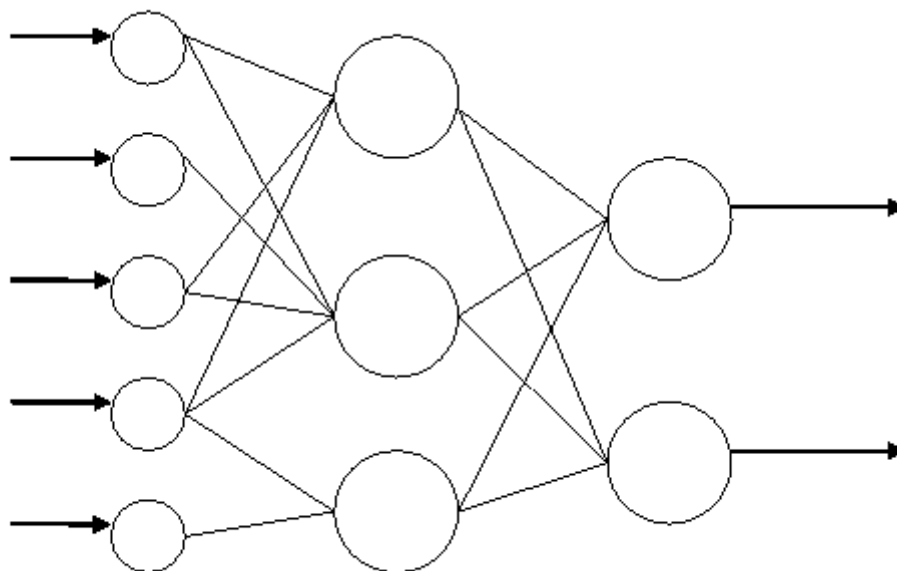


Рис. 1.3. Елементарний персептрон Розенблатта

Тільки асоціативні елементи, що являють собою формальні нейрони, виконують нелінійну обробку інформації і мають ваги зв'язків, що змінюються. R-елементи з фіксованою вагою формують сигнал реакції персептрону на вхідний стимул [20].

Представлена на рисунку мережа є одношаровою, так як має тільки один шар нейропроцесорних елементів.

З сьогоденних позицій одношаровий персептрон становить лише

історичний інтерес.

Алгоритм навчання персептрону. Навчання мережі полягає у підстроюванні вагових коефіцієнтів кожного нейрона. Нехай є набір пар векторів  $(x^a, y^a)$ ,  $a = 1 \dots p$ , званий навчальною вибіркою. Будемо називати нейронну мережу навченою на даній навчальній вибірці, якщо при подачі на входи мережі кожного вектора  $x^a$  на виходах щоразу виходить відповідний вектор  $y^a$ .

Запропонований Ф. Розенблаттом метод навчання полягає в ітераційному підстроюванні матриці ваг, що послідовно зменшує помилку у вихідних векторах. Алгоритм включає кілька кроків:

1. Початкові значення ваги всіх нейронів  $W(t = 0)$  вважаються випадковими.

2. Мережі пред'являється вхідний образ  $x^a$ , у результаті формується вихідний образ  $y^a \neq y^a$ .

3. Обчислюється вектор помилки  $\delta^a = (y^a - y^a)$ , що робиться мережею на виході. Подальша ідея полягає в тому, що зміна вектора вагових коефіцієнтів в області малих помилок має бути пропорційно помилці на виході, і дорівнює нулю, якщо помилка дорівнює нулю.

4. Вектор ваг модифікується за такою формулою:

$$W(t + \Delta t) = W(t) + nx^a * (\delta^a)^* (\delta^a)^T \quad (1.10)$$

Тут  $0 < n < 1$  – темп навчання.

5. Кроки 1-3 повторюються для всіх навчальних векторів. Один цикл послідовного пред'явлення усієї вибірки називається епохою. Навчання завершується після кількох епох, а) коли ітерації зійдуться, тобто вектор терезів перестає змінюватися, або б) коли повна підсумована по всіх векторах абсолютна помилка стане меншою за деяке малого значення.

Представлений алгоритм відноситься до широкого класу алгоритмів навчання з вчителем оскільки відомі як вхідні вектори, так і необхідні значення вихідних векторів (є вчитель, здатний оцінити правильність відповіді учня).

Візьмемо за основу при побудові гіперплощини, що розділяє класи, відсутність помилок на вибірці. Щоб задовольнити цю умову, доведеться вирішувати систему лінійних нерівностей [14]:

$$\begin{aligned}(x^i, \alpha) &> \alpha_0 \quad (i=1, \dots, n) \\ (y^j, \alpha) &< \alpha_0 \quad (j=1, \dots, m).\end{aligned}$$

Тут  $x^i$  ( $i=1, \dots, n$ ) – вектори з навчальної вибірки, які відносяться до першого класу, а  $y^j$  ( $j=1, \dots, m$ ) – до другого.

Зручно переформулювати завдання. Збільшимо розмірність всіх векторів на одиницю, додавши ще одну координату -  $\alpha_0$  до  $\alpha$ ,  $x_0 = 1$  – для всіх  $x$  і  $y_0 = 1$  для всіх  $y$ . Збережемо для нових векторів попередні позначення – це не призведе до плутанини.

Нарешті, укладаємо:  $z^i = x^i$  ( $i=1, \dots, n$ ),  $z^j = -y^j$  ( $j=1, \dots, m$ ).

Тоді отримаємо систему  $n+m$  нерівностей:

$$(z^i, \alpha) > 0 \quad (i=1, \dots, n+m),$$

яку будемо вирішувати щодо  $\alpha$ . Якщо безліч рішень не пуста, то будь-який його елемент  $\alpha$  породжує вирішальне правило, безпомилкове на навчальній вибірці.

Ітераційний алгоритм розв'язання цієї системи надзвичайно простий. Він заснований на тому, що для будь-якого вектора  $x$  його скалярний квадрат  $(x, x)$  більший за нуль. Нехай  $\alpha$  деякий вектор, який претендує на роль розв'язання нерівностей  $(z^i, \alpha) > 0$  ( $i=1, \dots, n+m$ ), проте частина з них не виконується. Додамо ті  $z^i$ , для яких нерівності мають невірний знак, до вектора  $\alpha$  і знову перевіримо всі нерівності  $(z^i, \alpha) > 0$  і т.д. Якщо вони спільні, процес сходиться за кінцеве число кроків. Більше того, додавання  $z^i$  до  $\alpha$  можна робити відразу після того, як помилка  $((z^i, \alpha) < 0)$  виявлено, не чекаючи перевірки всіх нерівностей і цей варіант алгоритму теж сходиться.

## РОЗДІЛ 2. ТЕОРЕТИЧНІ АСПЕКТИ ДЛЯ ЗАСТОСУВАННЯ ШТУЧНИХ НЕЙРОННИХ МЕРЕЖ

### 2.1. Використання бібліотеки Keras для тренування простої нейронної мережі. Створення нейронних мереж та блоків

Програмістам на Python немає необхідності заново винаходити колесо. Такі бібліотеки, як Tensorflow, Torch, Theano і Keras, вже визначили основні структури даних для нейронної мережі, залишивши необхідність лише декларативно описати структуру нейронної мережі.

Keras надає ще й певну свободу – можливість вибрати кількість шарів, кількість нейронів, тип шару та функцію активації. На практиці елементів досить багато, але цього разу обійдемося простими прикладами.

Як уже згадувалося, є два спеціальні рівні, які мають бути визначені на основі конкретної проблеми – розмір шару введення та розмір шару виведення. Решта «приховані шари» використовуються вивчення складних нелінійних абстракцій завдання.

У цьому матеріалі будемо використовувати Python та бібліотеку Keras для передбачення рукописних цифр із бази даних MNIST.

Список необхідних бібліотек:

- numpy;
- matplotlib;
- sklearn;
- tensorflow;

Запуск з інтерпретатора Python. Для запуску чистої установки Python (будь-яка версія 3.6) встановлюємо необхідні модулі за допомогою `pip`.

Запускаємо код у віртуальному середовищі:

```
!pip install matplotlib
```

```
!pip install sklearn
```

```
!pip install tensorflow
```

Якщо ці модулі встановлені, тепер можна запускати весь код у проєкті.

Імпортуємо модулі та бібліотеки:

```

import numpy as np
import matplotlib.pyplot as plt
import gzip
from typing import List
from sklearn.preprocessing import OneHotEncoder
import tensorflow.keras as keras
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import itertools
%matplotlib inline

```

MNIST – це величезна база даних рукописних цифр, яка використовується як бенчмарк та точка знайомства з машинним навчанням та системами обробки зображень. Вона ідеально підходить, щоб зосередитись саме на процесі навчання нейронної мережі. MNIST – дуже чиста база даних, а це розкіш у світі машинного навчання.

Мета – за допомогою набору даних з 60 000 зображень рукописних цифр (представлених у вигляді зображень 28x28 пікселів, кожен з яких є градацією сірого від 0 до 255) натренувати систему класифікувати кожне значення відповідним ярликом (зображеною цифрою).

Набір даних складається з тренувальних та тестових даних, але для спрощення тут використовуватиметься лише тренувальний. Ось так його завантажуюємо:

```

%%bash
rm -Rf train-images-idx3-ubyte.gz
rm -Rf train-labels-idx1-ubyte.gz
wget -q http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
wget -q http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz

```

Читання міток. Є 10 цифр: (0-9), тому кожна мітка має бути цифрою від 0 до 9. Завантажений файл train-labels-idx1-ubyte.gz кодує мітки наступним чином:

Файл ярлика тренувального набору (train-labels-idx1-ubyte):

| [offset] | [type]           | [value]          | [Description]            |
|----------|------------------|------------------|--------------------------|
| 0000     | 32 bit integer   | 0x00000801(2049) | magic number (MSB first) |
| 0004     | 32 bit integer   | 60000            | number of items          |
| 0008     | unsigned byte ?? |                  | label                    |
| 0009     | unsigned byte ?? |                  | label                    |
| .....    | .....            | .....            | .....                    |
| xxxx     | unsigned byte ?? |                  | label                    |

Значення позначок від 0 до 9.

Перші 8 байт (або перші 2 32-бітових цілих числа) можна пропустити, тому що вони містять метадані файли, необхідні для низькорівневих мов програмування.

Для парсингу файлу виконуємо наступні операції:

- відкрити файл за допомогою бібліотеки `gzip`, щоб його можна було розпакувати;
- прочитати весь масив байтів на згадку;
- пропустити перші 8 байт;
- перебирати кожен байт і приводити його до цілого числа.

Примітка: якщо цей файл з неперевіреного джерела, знадобиться набагато більше перевірок. Але припустимо, що цей конкретний файл є надійним і підходить для цілей матеріалу.

Лістинг 1 (Додаток А):

```
with gzip.open('train-labels-idx1-ubyte.gz') as train_labels:
    data_from_train_file = train_labels.read()
# Пропускаємо перші 8 байт
label_data = data_from_train_file[8:]
assert len(label_data) == 60000
# Конвертуємо кожен байт у ціле число.
# Це буде число від 0 до 9
```

```

labels = [int(label_byte) for label_byte in label_data]
assert min(labels) == 0 and max(labels) == 9
assert len(labels) == 60000

```

Читання зображень:

| [offset] | [type]           | [value]          | [Description]     |
|----------|------------------|------------------|-------------------|
| 0000     | 32 bit integer   | 0x00000803(2051) | magic number      |
| 0004     | 32 bit integer   | 60000            | number of images  |
| 0008     | 32 bit integer   | 28               | number of rows    |
| 0012     | 32 bit integer   | 28               | number of columns |
| 0016     | unsigned byte ?? |                  | pixel             |
| 0017     | unsigned byte ?? |                  | pixel             |
| .....    | .....            | .....            | .....             |
| xxxx     | unsigned byte ?? |                  | pixel             |

Читання зображень трохи відрізняється від читання позначок. Перші 16 байт містять відомі метадані. Їх можна пропустити та переходити відразу до читання зображень. Кожна з них представлена у вигляді масиву 28\*28 байтів без знака. Все що потрібно читати по одному зображенню за раз і зберігати їх в масив.

```

SIZE_OF_ONE_IMAGE = 28**2
images = []
# Перебір тренувального файлу та читання одного зображення за раз
with gzip.open('train-images-idx3-ubyte.gz') as train_images:
    train_images.read(4 * 4)
    ctr = 0
    for _ in range(60000):
        image = train_images.read(size=SIZE_OF_ONE_IMAGE)
        assert len(image) == SIZE_OF_ONE_IMAGE

```

```

# Конвертувати в NumPy
image_np = np.frombuffer(image, dtype='uint8') / 255
images.append(image_np)
images = np.array(images)
images.shape

```

Висновок: (60000, 784).

У списку 60000 зображень. Кожне з них представлене бітовим вектором розміром `SIZE_OF_ONE_IMAGE`. Спробуємо побудувати зображення за допомогою бібліотеки `matplotlib`:

```

def plot_image(pixels: np.array):
    plt.imshow(pixels.reshape((28, 28)), cmap='gray')
    plt.show()
plot_image(images[25])

```

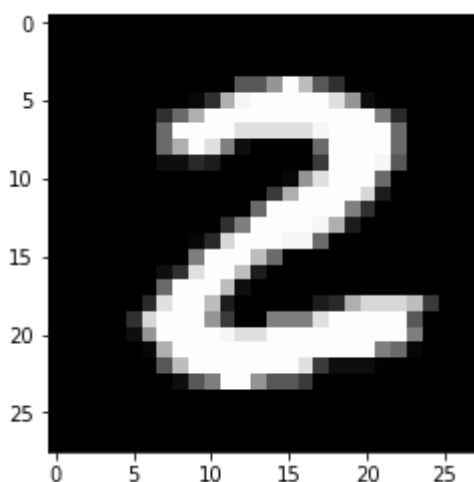


Рис. 2.1. Рукописна цифра з бази MNIST

Кодування міток зображення за допомогою One-hot encoding.

Використовуватимемо one-hot encoding для перетворення цільових міток на вектор.

```

labels_np = np.array(labels).reshape((-1, 1))
encoder = OneHotEncoder(categories='auto')
labels_np_onehot = encoder.fit_transform(labels_np).toarray()
labels_np_onehot

```

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [1., 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., 1., 0.]])
```

Були успішно створені вхідні дані та векторний висновок, який надходитиме на вхідний та вихідний шари нейронної мережі. Вектор введення з індексом *i* відповідатиме вектору виведення з індексом *i*.

Вступні дані:

```
labels_np_onehot[999]
```

Висновок:

```
array([0., 0., 0., 0., 0., 0., 1., 0., 0., 0.])
```

Вступні дані:

```
plot_image(images[999])
```

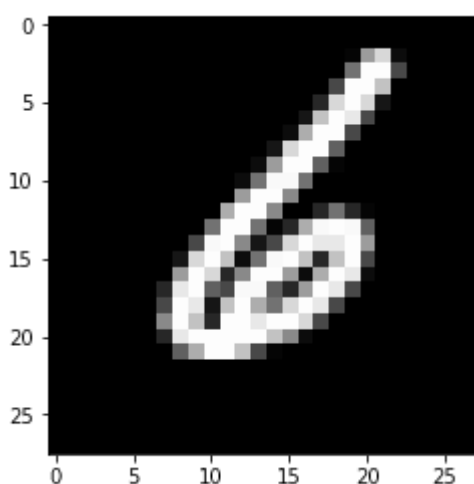


Рис. 2.2. Виведення цифри 6

У прикладі вище видно, що зображення з індексом 999 представляє цифру 6. Асоційований з ним вектор містить 10 цифр (оскільки є 10 міток), а цифра з індексом 6 дорівнює 1. Це означає, що мітка правильна.

Поділ датасету на тренувальний та тестовий. Для перевірки того, що нейронна мережа була правильно натренована, беремо певний відсоток тренувального набору (60 000 зображень) і використовуємо його в тестових

цілях.

Вступні дані:

```
X_train, X_test, y_train, y_test = train_test_split(images, labels_np_onehot)
print(y_train.shape)
print(y_test.shape)
(45000, 10)
(15000, 10)
```

Тут видно, що весь набір із 60 000 зображень був розбитий на два: один із 45 000, а інший з 15 000 зображень.

Тренування нейронної мережі за допомогою Keras:

```
model = keras.Sequential()
model.add(keras.layers.Dense(input_shape=(SIZE_OF_ONE_IMAGE,), units=128,
activation='relu'))
model.add(keras.layers.Dense(10, activation='softmax'))
model.summary()
model.compile(optimizer='sgd',
              loss = 'categorical_crossentropy',
              metrics=['accuracy'])
```

Висновок:

Model: "sequential"

Layer (type) Output Shape Param #

dense (Dense) (None, 128) 100480

dense\_1 (Dense) (None, 10) 1290

Total params: 101,770

Trainable params: 101,770

Non-trainable params: 0

Для навчання нейронної мережі виконаємо цей код.

```
model.fit(X_train, y_train, epochs=20, batch_size=128)
```

Висновок:

Train on 45000 samples

Epoch 1/20

45000/45000 [] - 2s 54us/sample - loss: 1.3391 - accuracy: 0.6710

Epoch 2/20

45000/45000 [] - 2s 39us/sample - loss: 0.6489 - accuracy: 0.8454

...

Epoch 20/20

45000/45000 [] - 2s 40us/sample - loss: 0.2584 - accuracy: 0.9279

Перевіряємо точність на тренувальних даних.

```
model.evaluate(X_test, y_test)
```

Висновок:

```
[0.2567395991722743, 0.9264]
```

Подивимося результати. Візьмемо випадкове зображення – картинку з індексом 1010. Беремо передбачену мітку (у даному випадку – 4, тому що на п'ятій позиції стоїть цифра 1).

```
y_test[1010]
```

Висновок:

```
array([0., 0., 0., 0., 1., 0., 0., 0., 0., 0.])
```

Побудуємо зображення відповідної картини

```
plot_image(X_test[1010])
```

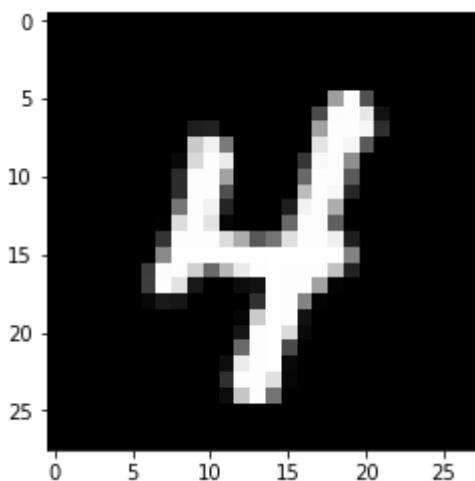


Рис. 2.3. Виведення картини

Розуміння виведення активаційного шару softmax. Пропустимо цифру через нейронну мережу та подивимося, який висновок вона передбачить.

Вступні дані:

```
predicted_results = model.predict(X_test[1010].reshape((1, -1)))
```

Виведення шару softmax – це розподіл ймовірностей для кожного висновку. І тут їх може бути 10 (цифри від 0 до 9). Але очікується, що кожне зображення буде відповідати лише одному.

Оскільки це розподіл ймовірностей, їхня сума приблизно дорівнює 1 (одиниці).

```
predicted_results.sum()
1.0000001
```

Читання виведення шару softmax для конкретної цифри. Як можна бачити далі, 5-ий індекс дійсно близький до 1 (0,99), а це означає, що він з великою ймовірністю є 4 ... а це так і є.

```
predicted_results
array([[1.2202066e-06, 3.4432333e-08, 3.5151488e-06, 1.2011528e-06, 9.9889344e-01,
        3.5855610e-04, 5.576822333e-05, 1.0446112e-04, 8.6736667e-04]],
      dtype=float32)
```

Перегляд матриці помилок:

```
predicted_outputs = np.argmax(model.predict(X_test), axis=1)
expected_outputs = np.argmax(y_test, axis=1)
```

```
predicted_confusion_matrix = confusion_matrix(expected_outputs,
predicted_outputs)
```

```
predicted_confusion_matrix
array([[1402, 0, 4, 3, 1, 6, 20, 2, 21, 2],
       [1, 1684, 9, 5, 4, 9, 1, 3, 9, 3],
       [13, 8, 1280, 9, 19, 5, 12, 15, 17, 8],
       [6, 8, 37, 1404, 1, 53, 3, 17, 33, 15],
       [4, 7, 8, 0, 1345, 1, 18, 3, 8, 54],
       [17, 8, 9, 31, 25, 1157, 25, 3, 24, 12],
       [9, 6, 10, 0, 10, 12, 1431, 0, 6, 1],
       [3, 11, 17, 4, 23, 2, 1, 1484, 5, 40],
```

```
[11, 16, 24, 40, 9, 25, 13, 3, 1348, 25],
[5, 5, 6, 16, 31, 6, 0, 43, 7, 1381]],
dtype=int64)
```

Візуалізуємо дані:

```
def plot_confusion_matrix(cm, classes,
                           title='Confusion matrix',
                           cmap=plt.cm.Blues):
    """
    Це функція принтерів і plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = 'd'
    thresh = cm.max()/2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()

# Compute confusion matrix
class_names = [str(idx) for idx in range(10)]
```

```

cnf_matrix = confusion_matrix(expected_outputs, predicted_outputs)
np.set_printoptions(precision=2)
# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names,
                      title='Confusion matrix, без normalization')
plt.show()

```

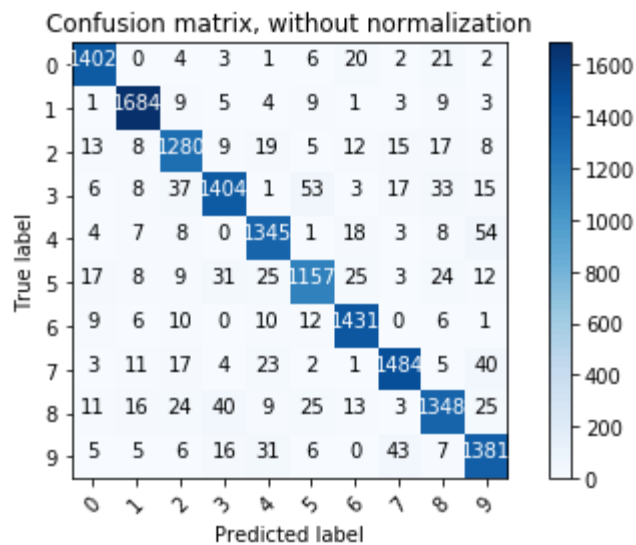


Рис. 2.4. Візуалізація даних

Для початку необхідно визначитися з тим, що являють собою базові компоненти нейронної мережі – нейрони. Нейрон приймає вступні дані, виконує з ними певні математичні операції, та виводить результат.

Нейрон з двома вхідними даними виглядає так:

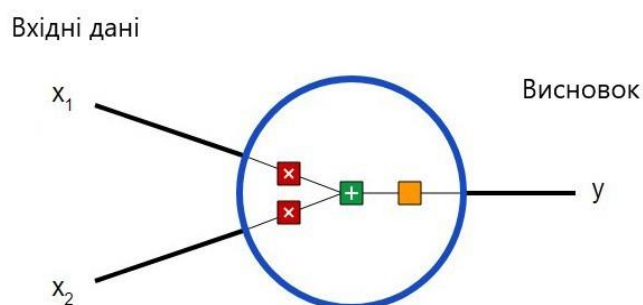


Рис. 2.5. Нейрон з двома вхідними даними

Тут трапляються три речі. По-перше, кожен вхід множиться на вагу (на схемі позначений червоним):

$$x_1 \rightarrow x_1 * \omega_1$$

$$x_2 \rightarrow x_2 * \omega_2$$

Потім усі зважені входи складаються разом зі зміщенням  $b$  (на схемі позначений зеленим):

$$(x_1 * \omega_1) + (x_2 * \omega_2) + b \quad (2.1)$$

Зрештою, сума передається через функцію активації (на схемі позначена жовтим):

$$y = \int (x_1 * \omega_1 * \omega_1 + x_2 * \omega_2 + b) \quad (2.2)$$

Функція активації використовується для підключення незв'язаних вхідних даних із висновком, у якого проста та передбачувана форма. Як правило, як функція активації, що використовується, береться функція сигмоїду:

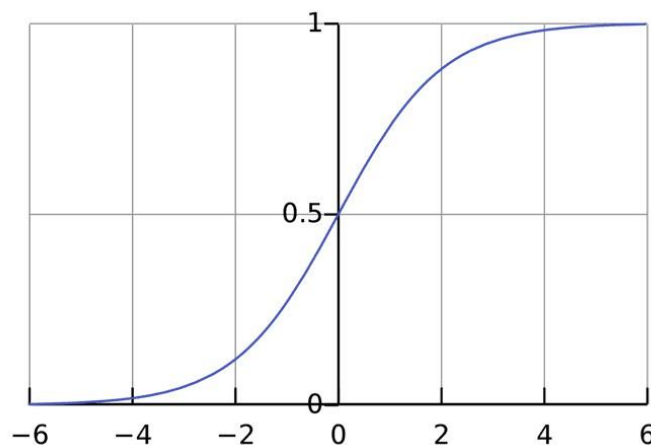


Рис. 2.6 Функція сигмоїду

Функція сигмоїду виводить лише числа в діапазоні  $(0, 1)$ . Ми сприймаємо це як компресію від  $(-\infty, +\infty)$  до  $(0, 1)$ . Великі негативні числа стають  $\sim 0$ , а великі позитивні числа стають  $\sim 1$ .

## 2.2. Приклади роботи з нейронами

Припустимо, у нас є нейрон із двома входами, який використовує функцію активації сигмоїду та має наступні параметри:

$$\omega = [0, 1]$$

$$b = 4$$

$w = [0, 1]$  – це просто один із способів написання  $w_1 = 0$ ,  $w_2 = 1$  у векторній формі. Привласним нейрону вхід зі значенням  $x = [2, 3]$ . Для більш компактного представлення використаємо скалярний добуток.

$$\begin{aligned} (\omega * x) + b &= ((\omega_1 * x_1) + (\omega_2 * x_2)) + b \\ &= 0 * 2 + 1 * 3 + 4 \\ &= 7 \end{aligned} \tag{2.3}$$

$$y = \int (\omega * x + b) = \int (7) = 0.999$$

З урахуванням, що вхід був  $x = [2, 3]$ , висновок дорівнює 0.999. Ось і все. Такий процес передачі вхідних даних для отримання висновку називається прямим поширенням, або feedforward.

Для цього потрібно використовувати NumPy. Це потужна обчислювальна бібліотека Python, яка використовує математичні операції.

Лістинг 2: Приступимо до імплементації нейрона.

```
1 import numpy as np
2
3
4 def sigmoid(x):
5     # Наша функція активації: f(x) = 1 / (1 + e^(-x))
6     return 1 / (1 + np.exp(-x))
7
8
9 class Neuron:
```

```

10 def __init__(self, weights, bias):
11     self.weights = weights
12     self.bias = bias
13
14 def feedforward(self, inputs):
15     # Вступні дані про вагу, додавання зміщення
16     # та подальше використання функції активації
17
18     total = np.dot(self.weights, inputs) + self.bias
19     return sigmoid(total)
20
21
22 weights = np.array([0, 1]) # w1 = 0, w2 = 1
23 bias = 4 # b = 4
24 n = Neuron (weights, bias)
25
26 x = np.array([2, 3]) # x1 = 2, x2 = 3
27 print(n.feedforward(x)) # 0.9990889488055994

```

Це той самий приклад, який розглядався раніше. Відповідь отримана цього разу також дорівнює 0.999.

Нейронна мережа по суті є групою зв'язаних між собою нейронів. Проста нейронна мережа виглядає так:

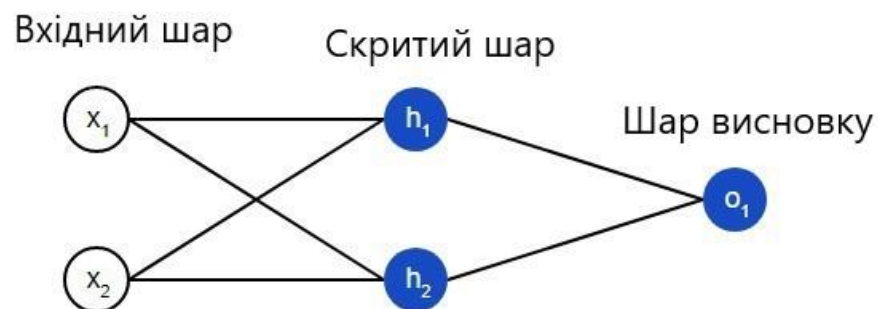


Рис. 2.6. Приклад збору нейронів у нейромережу

На вступному шарі мережі два входи –  $x_1$  та  $x_2$ . На прихованому шарі два нейтрони –  $h_1$  і  $h_2$ . На шарі виведення є один нейрон –  $o_1$ . Звернемо увагу, що вхідні дані для  $o_1$  є результатами виведення  $h_1$  і  $h_2$ . У такий спосіб і будується нейромережа.

Прихованим шаром називається будь-який шар між вступним шаром і шаром виведення, що є першим і останнім шарами відповідно. Прихованих шарів може бути кілька.

Використаємо продемонстровану вище мережу і уявімо, що всі нейрони мають однакову вагу  $w = [0, 1]$ , однакове зміщення  $b = 0$  і ту саму функцію активації сигмоїду. Нехай  $h_1$ ,  $h_2$  та  $o_1$  самі відзначають результати виведення представлених ними нейронів.

Як введення використаємо значення  $x = [2, 3]$ :

$$\begin{aligned} h_1 &= h_2 = \int (\omega * x + b) \\ &= \int ((0 * 2) + (1 * 3) + 0) \\ &= \int (3) \\ &= 0.9526 \end{aligned} \tag{2.4}$$

$$\begin{aligned} o_1 &= \int (\omega * [h_1, h_2] + b) \\ &= \int ((0 * h_1) + (1 * h_2) + 0) \\ &= \int (0.9526) \\ &= 0.7216 \end{aligned} \tag{2.5}$$

Результат виведення нейронної мережі для вхідного значення  $x = [2, 3]$  становить 0.7216. Все дуже просто.

Нейронна мережа може мати будь-яку кількість шарів з будь-якою кількістю нейронів у цих шарах.

Суть залишається такою самою: необхідно направити вхідні дані через нейрони в мережу для отримання в результаті вихідних даних.

Далі покажемо, як реалізувати пряме розповсюдження обумовлене по відношенню до нейронної мережі. Як опорна точка буде використана наступна схема нейронної мережі:



Рис. 2.7. Схема нейронної мережі

Лістинг 3: реалізація прямого розповсюдження заборгованості.

```

1  import numpy as np
2
3  # ... Тут код із попереднього розділу
4
5
6  class OurNeuralNetwork:
7      """
8      Нейронна мережа, яка має:
9      - 2 входи
10     - 1 прихований шар із двома нейронами (h1, h2)
11     - шар виведення з одним нейроном (o1)
12     У кожного нейрона однакові вага та зміщення:
13     - w = [0, 1]
14     - b = 0

```

```

15 """
16 def __init__(self):
17     weights = np.array([0, 1])
18     bias = 0
19
20 # Клас Neuron з попереднього розділу
21 self.h1 = Neuron(weights, bias)
22 self.h2 = Neuron(weights, bias)
23 self.o1 = Neuron (weights, bias)
24
25 def feedforward(self, x):
26     out_h1 = self.h1.feedforward(x)
27     out_h2 = self.h2.feedforward(x)
28
29 # Введення для  $o_1$  є висновками  $h_1$  та  $h_2$ 
30 out_o1 = self.o1.feedforward(np.array([out_h1, out_h2]))
31
32 return out_o1
33
34
35 network = OurNeuralNetwork()
36 x = np.array([2, 3])
37 print(network.feedforward(x)) # 0.7216325609518421

```

Ми знову здобули 0.7216.

### 2.3. Приклади тренування нейронної мережі

Припустимо, у нас є такі параметри:

|           |             |              |              |
|-----------|-------------|--------------|--------------|
| Ім'я/Name | Вага/Weight | Зріст/Height | Стать/Gender |
|-----------|-------------|--------------|--------------|

|         | (фунти) | (дюйми) |   |
|---------|---------|---------|---|
| Alice   | 133     | 65      | F |
| Bob     | 160     | 72      | M |
| Charlie | 152     | 70      | M |
| Diana   | 120     | 60      | F |

Натренуємо нейронну мережу таким чином, щоб вона передбачала стать заданої людини залежно від її ваги та зростання.



Рис. 2.8. Схема нейронної мережі

Чоловіки Male будуть представлені як 0, а жінки Female як 1. Для простоти подання дані також будуть дещо зміщені.

| Ім'я/Name | Вага/Weight<br>(мінус 135) | Зріст/Height (мінус 66) | Стать/Gender |
|-----------|----------------------------|-------------------------|--------------|
| Alice     | -2                         | -1                      | 1            |
| Bob       | 25                         | 6                       | 0            |
| Charlie   | 17                         | 4                       | 0            |
| Diana     | -15                        | -6                      | 1            |

Для оптимізації тут зроблено довільні усунення 135 і 66. Однак, зазвичай для усунення вибираються середні показники.

Перед тренуванням нейронної мережі потрібно вибрати спосіб оцінки того, наскільки добре мережа справляється із завданнями. Це необхідно для

подальших спроб виконувати поставлене завдання краще. Таким є принцип втрати.

В даному випадку буде використовуватися середньоквадратична помилка (MSE) втрати:

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_{true} - y_{pred})^2 \quad (2.6)$$

Давайте розберемося:

–  $n$  – число об'єктів, що розглядаються, в даному випадку дорівнює 4. Це Alice, Bob, Charlie і Diana;

–  $y$  – змінні, які будуть передбачені. У разі це стать людини;

–  $y_{true}$  – істинне значення змінної, тобто так звана правильна відповідь.

Наприклад, для Alice значення  $y_{true}$  буде 1, тобто Female;

–  $y_{pred}$  – передбачуване значення змінної. Це результат виведення мережі;

–  $(y_{true} - y_{pred})^2$  – називають квадратичною помилкою (MSE). Тут функція втрати просто бере середнє значення з усіх квадратичних помилок. Звідси й назва помилки. Чим краще передбачення, тим нижчі втрати.

Кращі прогнози = Найменші втрати.

Тренування нейронної мережі = Прагнення до мінімізації її втрат.

Наприклад, наша мережа завжди видає 0. Іншими словами, вона впевнена, що всі люди – чоловіки. Якою буде втрата?

| Ім'я/Name | $y_{true}$ | $y_{pred}$ | $(y_{true} - y_{pred})^2$ |
|-----------|------------|------------|---------------------------|
| Alice     | 1          | 0          | 1                         |
| Bob       | 0          | 0          | 0                         |
| Charlie   | 0          | 0          | 0                         |
| Diana     | 1          | 0          | 1                         |

$$MSE = \frac{1}{4} (1 + 0 + 0 + 1) = 0.5$$

Python код середньоквадратичної помилки (MSE)

Нижче наведено код для підрахунку втрат:

```

1 import numpy as np
2
3
4 def mse_loss(y_true, y_pred):
5     # y_true та y_pred є масивами numpy з однаковою довжиною
6     return ((y_true - y_pred) ** 2).mean()
7
8
9 y_true = np.array([1, 0, 0, 1])
10 y_pred = np.array([0, 0, 0, 0])
11
12 print(mse_loss(y_true, y_pred)) # 0.5

```

При виникненні складнощів з розумінням роботи коду варто ознайомитись з `quickstartNumPy` для операцій з масивами.

Тренування нейронної мережі – багатоваріантні обчислення.

Поточна мета зрозуміла – мінімізація втрат нейронної мережі. Тепер стало ясно, що вплинути на передбачення мережі можна за допомогою зміни її ваги та усунення. Однак як мінімізувати втрати?

Для простоти уявімо, що в наборі даних розглядається тільки Alice:

| Ім'я/Name | Вага/Weight (мінус 135) | Зріст/Height (мінус 66) | Стать/Gender |
|-----------|-------------------------|-------------------------|--------------|
| Alice     | -2                      | -1                      | 1            |

Потім втрата середньоквадратичної помилки буде просто квадратичною помилкою для Alice:

$$\begin{aligned}
 MSE &= \frac{1}{1} \sum_{l=1}^1 (y_{true} - y_{pred})^2 \\
 &= (y_{true} - y_{pred})^2 \\
 &= (1 - y_{pred})^2
 \end{aligned}
 \tag{2.7}$$

Ще один спосіб розуміння втрати – уявлення її як функції ваги та усунення. Давайте позначимо кожен вагу і зміщення в мережі, що

розглядається (рис. 2.9):

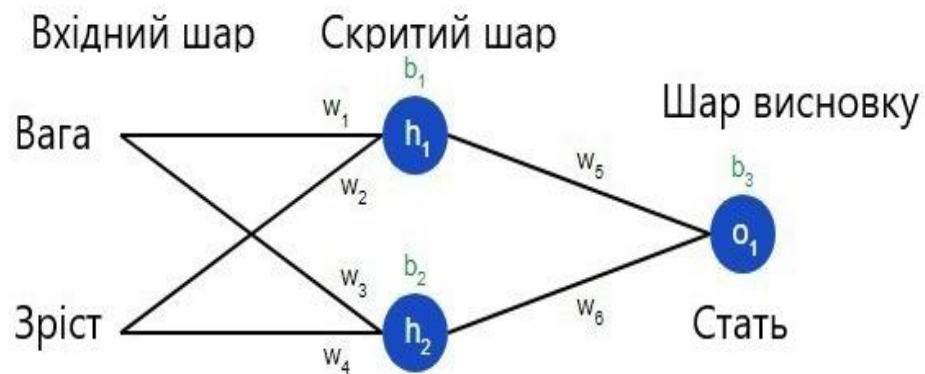


Рис. 2.9. Позначення кожної ваги і зміщення в мережі

Потім можна прописати втрату як багатоваріантну функцію:

$$L(\omega_1, \omega_2, \omega_3, \omega_4, \omega_5, \omega_6, b_1, b_2, b_3)$$

Припустимо, нам потрібно трохи відредагувати  $w_1$ . У такому разі, як зміниться втрата  $L$  після внесення поправок до  $w_1$ ?

На це запитання може відповісти приватна похідна  $\frac{\partial L}{\partial \omega_1}$ .

Тут математичні обчислення будуть набагато складнішими.

Для початку, давайте перепишемо приватну похідну в контексті  $\frac{\partial y_{pred}}{\partial \omega_1}$ :

$$\frac{\partial L}{\partial \omega_1} = \frac{\partial L}{\partial y_{pred}} * \frac{\partial y_{pred}}{\partial \omega_1} \quad (2.8)$$

Дані обчислення можливі завдяки диференціюванню складної функції.

Підрахувати  $\frac{\partial L}{\partial y_{pred}}$  можна завдяки обчисленій вище  $L = (1 - y_{pred})^2$ :

$$\frac{\partial L}{\partial y_{pred}} = \frac{\partial (1 - y_{pred})^2}{\partial y_{pred}} = -2(1 - y_{pred}) \quad (2.9)$$

Тепер, давайте визначимо, що робити з  $\frac{\partial y_{pred}}{\partial \omega_1}$ . Як і раніше, дозволимо  $h_1$ ,  $h_2$ ,  $o_1$  стати результатами виведення нейронів, які вони представляють. Подальші обчислення:

$$y_{pred} = o_1 = \int(\omega_5 h_1 + \omega_6 h_2 + b_3) \quad (2.10)$$

Як було зазначено раніше, тут  $f$  є функцією активації сигмоїду. Оскільки  $w_1$  впливає лише на  $h_1$ , а не на  $h_2$ , можна записати:

$$\frac{\partial y_{pred}}{\partial \omega_1} = \frac{\partial y_{pred}}{\partial h_1} * \frac{\partial h_1}{\partial \omega_1} \quad (2.11)$$

$$\frac{\partial y_{pred}}{\partial h_1} = \omega_5 * \int(\omega_5 h_1 + \omega_6 h_2 + b_3) \quad (2.12)$$

Використання диференціювання складної функції.

Ті самі дії проводяться для  $\frac{\partial h_1}{\partial \omega_1}$ :

$$h_1 = \int \omega_1 x_1 + \omega_2 x_2 + b_1 \quad (2.13)$$

$$\frac{\partial h_1}{\partial \omega_1} * x_1 * \int(\omega_1 x_1 + \omega_2 x_2 + b_1) \quad (2.14)$$

У разі  $x_1$  – вага, а  $x_2$  – зріст. Тут  $f'(x)$  як похідна функції сигмоїду зустрічається вдруге. Спробуємо вивести її:

$$\int x = \frac{1}{1+e^x} \quad (2.15)$$

$$\int(x) = \frac{e^x}{(1+e^x)^2} = \int(x) * (1 - \int(x)) \quad (2.16)$$

Функція  $f'(x)$  у такому вигляді буде використана дещо пізніше.

Ось і все. Тепер  $\frac{\partial L}{\partial \omega_1}$  розбита на кілька частин, які будуть оптимальними для підрахунку:

$$\frac{\partial L}{\partial \omega_1} = \frac{\partial L}{\partial y_{pred}} * \frac{\partial y_{pred}}{\partial h_1} * \frac{\partial h_1}{\partial \omega_1} \quad (2.17)$$

Ця система підрахунку приватних похідних при роботі у зворотному порядку відома як метод зворотного поширення помилки, або backprop.

У нас накопичилося чимало формул, у яких легко заплутатися. Для кращого розуміння принципу роботи розглянемо наступний приклад.

У цьому прикладі також буде задіяна тільки Alice:

| Ім'я/Name | Вага/Weight (мінус 135) | Зріст/Height (мінус 66) | Стать/Gender |
|-----------|-------------------------|-------------------------|--------------|
| Alice     | -2                      | -1                      | 1            |

Тут вага буде представлена як 1, а зміщення як 0. Якщо виконаємо пряме

поширення (feedforward) через мережу, отримаємо:

$$h_1 = \int \omega_1 x_1 + \omega_2 x_2 + b_1 \quad (2.18)$$

$$= \int (-2 \pm 1 + 0)$$

$$= 0.0474$$

$$h_2 = \int \omega_3 \omega_1 + \omega_4 x_2 + b_2 = 0.0474 \quad (2.19)$$

$$o_1 = \int \omega_5 \omega_1 + \omega_6 h_2 + b_3 \quad (2.20)$$

$$= \int 0.0474 + 0.0474 + 0$$

$$= 0.524$$

Видачі нейронної мережі  $y_{pred} = 0.524$ . Це дає нам слабке уявлення про те, чи розглядається чоловік Male (0), або жінка Female (1). Давайте підрахуємо  $\frac{\partial L}{\partial \omega_1}$ :

Ми вивели  $f'(x) = f(x) * (1 - f(x))$  раніше для нашої функції активації сигмоїду.

Результат свідчить, що якщо ми збираємося збільшити  $w_1$ ,  $L$  трохи збільшується в результаті.

У нас є всі необхідні інструменти для тренування нейронної мережі. Ми використовуємо алгоритм оптимізації під назвою стохастичний градієнтний спуск (SGD), який говорить нам, як саме поміняти вагу та усунення для мінімізації втрат. По суті це відображається в наступному рівнянні:

$$\omega_1 \leftarrow \omega_1 - n \frac{\partial L}{\partial \omega_1} \quad (2.21)$$

$\eta$  є константою під назвою оцінка навчання, що контролює швидкість навчання.

Все що ми робимо, так це віднімаємо  $\frac{\partial L}{\partial \omega_1}$  з  $w_1$ :

Якщо  $\frac{\partial L}{\partial \omega_1}$  позитивна,  $w_1$  зменшиться, що призведе до зменшення  $L$ .

Якщо  $\frac{\partial L}{\partial \omega_1}$  негативна,  $w_1$  збільшиться, що призведе до зменшення  $L$ .

Якщо ми застосуємо це на кожну вагу та зміщення в мережі, втрата

поступово знижуватиметься, а показники мережі сильно покращаться.

Наш процес тренування буде виглядати так:

Вибираємо один пункт із нашого набору даних. Це те, що робить його стохастичним спуском градієнта. Ми обробляємо лише один пункт за раз.

Підраховуємо всі приватні похідні втрати за вагою чи зміщенням. Це може бути  $\frac{\partial L}{\partial \omega_1}$ ,  $\frac{\partial L}{\partial \omega_2}$ , і таке інше.

Використовуємо рівняння оновлення для оновлення кожної ваги та зміщення;

Повертаємось до першого пункту.

Погляньмо, як це працює на практиці.

Нарешті, ми реалізуємо готову нейронну мережу (Додаток В).

Наші втрати постійно зменшуються в міру того, як навчається нейронна мережа:

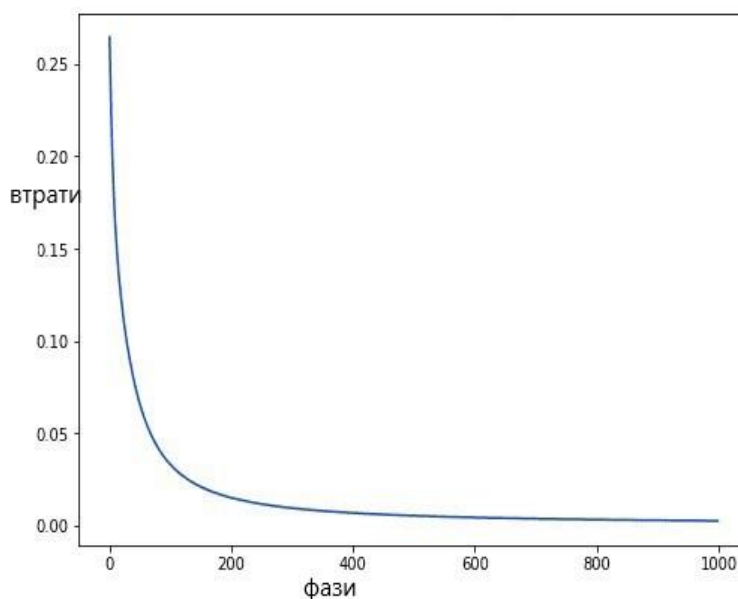


Рис. 2.11. Втрати нейронної мережі vs

Тепер ми можемо використовувати нейронну мережу для передбачення статей:

1 # Робимо передбачення

```
2 emily = np.array([-7, -3]) # 128 фунтів, 63 дюйми
3 frank = np.array([20, 2]) # 155 фунтів, 68 дюймів
4 print("Emily: %.3f" % network.feedforward(emily)) # 0.951 - F
5 print("Frank: %.3f" % network.feedforward(frank)) # 0.039 - M
```

### **РОЗДІЛ 3. ПРАКТИЧНЕ ЗАСТОСУВАННЯ ШТУЧНОЇ НЕЙРОННОЇ МЕРЕЖІ YOLOV3 ДЛЯ ДЕТЕКТУВАННЯ, ТРЕКІНГУ І ПІДРАХУНКУ ПІШОХОДІВ У РЕЖИМІ РЕАЛЬНОГО ЧАСУ**

#### **3.1. Алгоритми навчання та тестування нейронної мережі YOLOv3**

YOLOv3 складається з магістральної мережі DarkNet-53 і містить 106 шарів, серед яких 75 шарів згортки, залишкові блоки, з'єднання, що пропускаються, і шари з підвищеною дискретизацією (Додаток Г). Залишковий блок включає пару згорткових шарів розмірністю  $3 \times 3$  і  $1 \times 1$  з shortcut з'єднанням, необхідним для запобігання загасання градієнта.

На вхід мережі подаються зображення розмірністю  $992 \times 480$  для подальшої їх обробки та навчання нейронної мережі на них. DarkNet53 відповідає за модифікацію цих зображень, їх підготовку до наступних етапів та отримує на виході відповідні карти ознак. Після цього ці карти ознак йдуть на блоки YOLOv3, які займаються детекцією.

На шари YOLO надходять вже перетворені зображення їх подальшої обробки. На першому шарі YOLO роздільна здатність сітки становить  $1/32$  вхідного зображення, на останньому шарі –  $1/8$  вхідного зображення, що дозволяє виявляти як великі, так і дрібні об'єкти. Між шарами YOLO також знаходяться шари згортки, route шари, що поєднують масив попередніх шарів разом і upsampling шари, призначені для збільшення зображення.

Для аналізу точності одержаних результатів роботи нейронної мережі побудовано загальний алгоритм тестування нейронної мережі YOLOv3, представлений на рис. 3.1.

Спочатку відбувається поділ вхідних даних на валідаційну (V) і тренувальну (T) вибірки у співвідношенні 20% до 80% відповідно, таким чином тренувальна вибірка становить 67 200 зображень, валідаційна – 16 800 зображень.

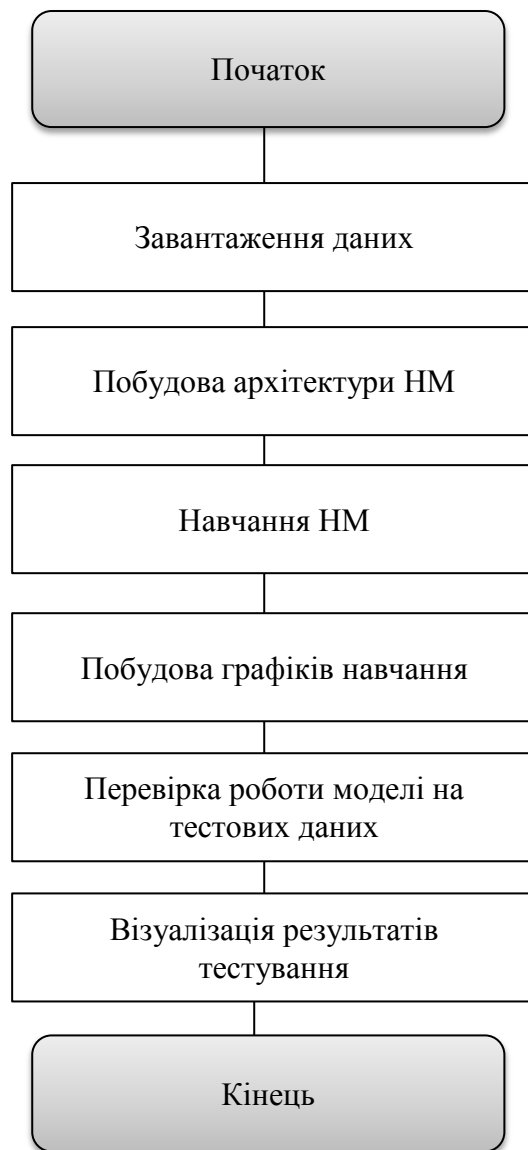


Рис. 3.1. Алгоритм тестування нейронної мережі

Після чого починається цикл навчання на епосі. За епохами відбувається обробка всіх зображень [5]. Усього епох – 15 000. У циклі відбувається поділ тренувальної вибірки на пакети (1), тобто. число зображень, що будуть оброблятися нейромережею одночасно,  $t \in [1, N]$ , де  $N$  – число пакетів. Один пакет містить 2 зображення.

Спочатку відбувається прямий прохід безлічі зображень з пакета  $t$  через нейронну мережу  $f^*$ . Нехай  $p = f^*(x)$  – безліч прогнозів нейронної мережі (прямокутники). Далі порівнюються набір значень (координати центру, висота та ширина прямокутників), отриманий після прямого проходу нейромережі з розміткою, що становить тренувальний набір, та обчислюється помилка мережі

(1) на пакеті. Потім відбувається усереднення помилки на всіх зображеннях пакета, відбувається оновлення ваги та обчислюється загальна помилка (2) на тренувальній вибірці, а також обчислюються метрики якості кожні 300 епох.

Після обробки всіх пакетів відбувається обчислення помилки (3) на валідаційній вибірці, а також кожні 300 епох розраховуються метрики якості на цій вибірці [6]. Кожні 300 епох зберігаються отримані ваги, щоб у разі перенавчання була можливість використовувати останні збережені ваги. Після того, як кількість епох досягне 15 000, навчання нейронної мережі закінчується.

### **3.2. Результати навчання, отримані метрики**

Для оцінки якості розпізнавання об'єкта пішохід, були використані такі метрики як точність (precision), точність (accuracy), повнота (recall) і локалізація об'єкта (IoU) [7].

Під час навчання нейронної мережі відбувався підрахунок метрик якості розпізнавання на валідаційній та навчальній вибірках.

Осередок матриці, що відповідає числу 153 266, показує, що 153 266 пішоходів були чітко визначені нейронною мережею як об'єкт «пішохід». Осередок матриці, що відповідає числу 8195, говорить про те, що 8195 пішоходів були класифіковані нейронною мережею, як «не пішохід». Осередок матриці, що відповідає числу 7454, показує, що нейронна мережа класифікувала 7454 об'єктів, як «пішохід», хоча ці об'єкти належали до категорії «не пішохід». Осередок матриці, що відповідає числу 0, говорить про те, що 0 об'єктів, що належать до категорії «не пішохід», були класифіковані нейронною мережею, як «не пішохід».

З матриці помилок можна подати значення метрик якості розпізнавання як в табл. 3.2.

Таблиця 3.2

Значення метрик для класу «пішохід»

| Клас    | Точність<br>(ассигасу) | Точність<br>(Precision) | Повнота | Локалізація<br>об'єкту |
|---------|------------------------|-------------------------|---------|------------------------|
| Пішохід | 90.7%                  | 95.4%                   | 94.8%   | 88.7%                  |

Після навчання було отримано графік зміни функції втрат, а також графіки, що відповідають метрикам якості розпізнавання.

Приклади зображень із розпізнаними пішоходами представлені на рис. 3.2-3.3. Код для запуску роботи нейронної мережі представлений у Додатку Д.



Рис. 3.2. Приклад результату роботи нейронної мережі у денний час



Рис. 3.3. Приклад результату роботи нейронної мережі у вечірній час

Для підрахунку пішоходів на перехресті виділено спеціальні зони, у яких ведеться підрахунок пішоходів [8]. На рис. 3.4 ці зони зображені синіми чотирикутниками.

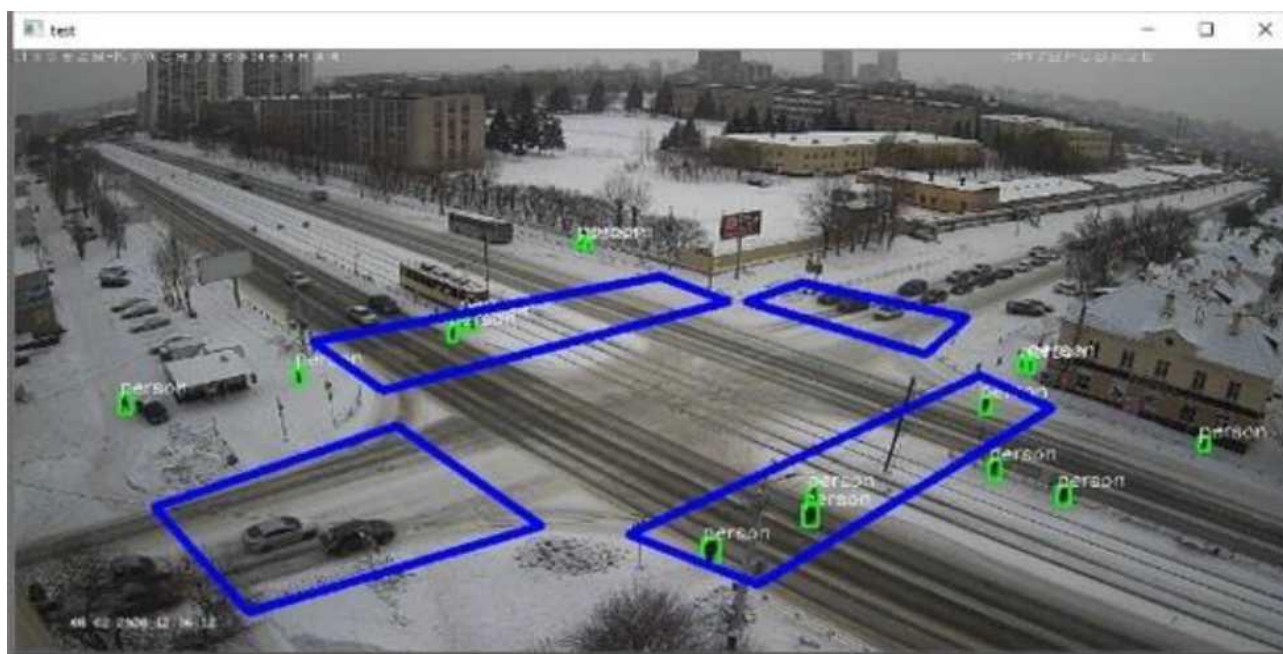


Рис. 3.4. Приклад результату роботи нейронної мережі

Таким чином, ведеться підрахунок пішоходів на кожному з пішохідних переходів:

```

#підрахунок пішоходів
import cv2
import math
import numpy as np
від sort import *
count_pedestrian = [0,0,0,0]
current_pedestrian=[[[]],[[]],[[]],[[]]]
lb      = [(408,379),(181,448),(108,364),(296,299),
(408,    379)]
lt      = [(553,200),(285,271),(231,232),(500,180),
(553,    200)]
rb      = [(804,285),(576,426),(476,386),(748,259),
(804,    285)]
rt      = [(738,213),(706,242),(567,200),(610,185),
(738,    213)]

def convertBack(x, y, w, h):
xmin=int(round(x -(w /    2)))
xmax=int(round(x +(w /    2)))
ymin=int(round(y -(h /    2)))
ymax=int(round(y +(h /    2)))
return xmin, ymin, xmax, ymax

def is_left(point,point1,point2):
res=((point[0]-point1[0])/(point2[0]-point1[0])-(point[1]-
point1[1])/(point2[1]-point1[1]))
return res

def in_rectangle(p):
global lb, lt, rb, rt
center=((p[2]-p[0])/2)+p[0],((p[3]-p[1])/2)+p[1])

```

```

    if is_left(center,lb[0],lb[1])>=0 and
    is_left(center,lb[2],lb[3])>=0 and
    is_left(center,lb[1],lb[2])<=0 and is_left(center,lb[3],lb[4])
    <=0:
        return 1
    if is_left(center,lt[0],lt[1])>=0 and
    is_left(center,lt[2],lt[3])>=0 and
    is_left(center,lt[1],lt[2])<=0 and is_left(center,lt[3],lt[4])
    <=0:
        return 2
    if is_left(center,rt[0],rt[1])>=0 and
    is_left(center,rt[2],rt[3])>=0 and
    is_left(center,rt[1],rt[2])<=0 and is_left(center,rt[3],rt[4])
    <=0:
        return 3
    if is_left(center,rb[0],rb[1])>=0 and
    is_left(center,rb[2],rb[3])>=0 and
    is_left(center,rb[1],rb[2])<=0 and is_left(center,rb[3],rb[4])
    <=0:
        return 4
    return 0 def count(trackers,img)

```

Таким чином, пішохід знаходиться у певній чотирикутній зоні (рис. 3.5), якщо координати центру прямокутника, що відповідає пішоходу, лежать ліворуч від прямої  $A$  і праворуч від прямої, протилежної прямої  $A$  [3].

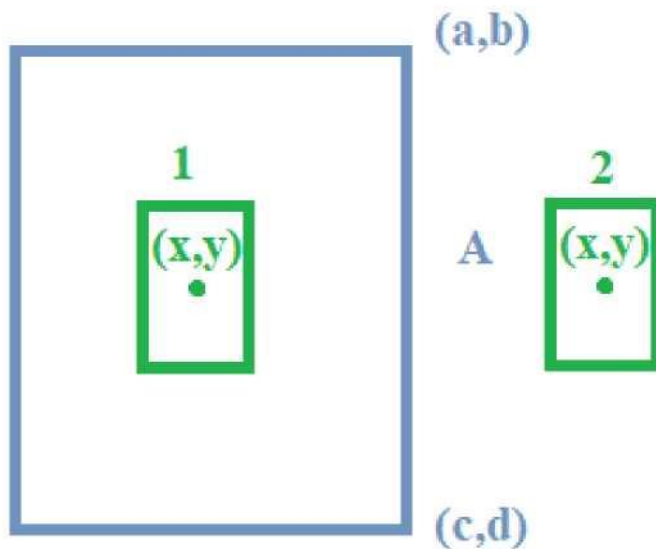


Рис. 3.5. Зона знаходження пішохода

Аналогічно з двома іншими сторонами чотирикутної зони. Отже, після того, як пішохід залишив зону підрахунку, ми збільшуємо кількість людей, які пройшли через цю зону.

Для оцінки точності підрахунку пішоходів було обрано тридцятихвилинний відеофрагмент. Оцінка точності проводилася окремо за кожним пішохідним переходом.

## ВИСНОВОК

У роботі були розроблені основні концепції, які становлять основу застосування штучних нейронних мереж для розв'язання прикладних задач.

1. Методи Data Mining допомагають вирішити багато завдань, із якими стикається аналітик. З них основними є: класифікація, регресія, пошук асоціативних правил та кластеризація.

Мета технології Data Mining – знаходження даних таких моделей, які можуть бути знайдені звичайними методами. Існують два види моделей: передбачувальні та описові.

2. Штучні нейронні мережі це клас моделей, заснованих на біологічній аналогії з мозком людини та призначених після проходження етапу так званого навчання на наявних даних для вирішення різноманітних завдань аналізу даних. При застосуванні цих методів насамперед постає питання вибору конкретної архітектури мережі. Розмір та структура мережі повинні відповідати суті досліджуваного явища.

Нейронна мережа, отримана в результаті навчання, виражає закономірності, присутні в даних. При такому підході вона виявляється функціональним еквівалентом деякої моделі залежностей між змінними, подібними до тих, що будуються в традиційному моделюванні.

3. Завдання чіткого поділу двох класів полягає у створенні правила, яке дозволяє віднести вектори властивостей об'єктів до одного з двох класів. Для цього використовується математичний апарат, який включає побудову лінійного вирішального правила та пошук такої функції  $f(x)$ , що при  $f(x) > 0$  вектор  $x$  відноситься до першого класу, а при  $f(x) < 0$  до другого.

4. Штучні нейронні мережі це потужний, але при цьому нетривіальний прикладний інструмент. Вони є важливим розширенням поняття обчислення. Нейронні мережі обіцяють створення автоматів, що виконують функції, які були раніше виключно прерогативою людини. Машини можуть виконувати рутинні, монотонні та небезпечні завдання, і з розвитком технології виникнуть нові програми.

Бібліотеки, такі як TensorFlow, Keras, Torch і Theano, спростили процес створення нейронних мереж, дозволяючи програмістам зосередитися на декларативному описі архітектури мережі.

Нейронна мережа була створена за допомогою Keras, з використанням функцій активації ReLU для прихованого шару та softmax для виходу. Модель була скомпільована з оптимізатором SGD і функцією втрат categorical\_crossentropy.

Нейронна мережа складається з нейронів, які виконують математичні операції над вхідними даними, використовуючи ваги та функції активації. Розглянуті приклади показують, як нейрони обробляють інформацію та як результати їхньої роботи об'єднуються в мережу.

5. У роботі було розроблено модель штучної нейронної мережі, яка в режимі реального часу виявляє та веде підрахунок пішоходів у кожному напрямку дорожнього вузла.

Для збору даних для навчання було взято 7 камер різних дорожніх вузлів. З них було зібрано близько 7000 зображень, на яких розтошавоно кожного з пішоходів. Після подальших перетворень, шляхом аугментації, кількість зображень зросла до 84 000. Серед них 16 800 зображення є валідаційною вибіркою, що залишилися – тренувальною. Підготовку даних здійснювали за допомогою платформи Supervisely.

Як фреймворк для навчання використовувався DarkNet. Це фреймворк із відкритим вихідним кодом, написаний мовою Сі. Як середовище розробки використовувався PyCharm із застосуванням мови програмування Python 3.6.

Для аналізу точності роботи нейронної мережі були використані метрики якості, такі як – локалізація об'єкта, точність (accuracy), точність (precision), повнота (recall), також візуалізується матриця помилок.

Докладно описано процес підрахунку пішоходів на кожному з пішохідних переходів. Наведено графіки точності підрахунку для кожної зони, що відповідає пішохідному переходу, з яких згодом можна побачити завантаженість кожної ділянки.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bernardin K. Evaluating Multiple Object Tracking Performance: The CLEAR MOT Metrics. International Conference on Computer Vision, 2008. P. 94-107.
2. Bewley A. Simple online and realtime tracking. URL : <https://arxiv.org/pdf/1602.00763.pdf>
3. Czichos H. Measurement, Testing and Sensor Technology. Fundamentals and Application to Materials and Technical Systems. Berlin: Springer, 2018. 213 p.
4. Diagnostics as a Reasoning Process: From Logic Structure to Software Design. Journal of Computing and Information Technology. 2018. Vol. 27 (1). P. 43-57.
5. Dicle C. The way they move: Tracking multiple targets with similar appearance. International Conference on Computer Vision, 2013. P. 73-84.
6. Girshick R. Fast R-CNN. Computer Vision and Pattern Recognition (CVPR'15). Boston, MA, USA: IEEE, 2015. 9 p.
7. Hatch S.V. Computerized Engine Controls. Boston: Cengage Learning, 2016. 688 p.
8. Lin T.-Y. Feature Pyramid Networks for Object Detection. Computer Vision and Pattern Recognition (CVPR'14). Washington, USA: IEEE, 2014. 9 p.
9. LISA lab, University of Montreal Deep Learning Tutorial. 2015. 173 p.
10. Machine Learning Techniques in Additive Manufacturing: a State of the Art Review on Design, Processes and Production Control / S. Saxena, T. & Gopi, N. Harikeerthana, M. Gupta, V. Gaur, G. Krolczyk, C.S. Wu. Journal of Intelligent Manufacturing. 2022. Vol. 1. P. 1-35.
11. Mariya Yao Applied Artificial Intelligence: A Handbook For Business Leaders Kindle Edition, Publisher: TOPBOTS, 2018. 246 p.
12. Neil Wilkins Artificial Intelligence: An Essential Beginner's Guide to AI, Machine Learning, Robotics, The Internet of Things, Neural Networks, Deep

Learning, Reinforcement Learning, and Our Future Paperback. Publisher: Bravex Publications, 2019. 112 p.

13. Redmon J. YOLOv3: An incremental improvement. Computer Vision and Pattern Recognition (CVPR'18). Washington, USA : IEEE, 2018. 6 p.

14. Redmon J. You Only Look Once: Unified, Real-Time Object Detection [Text]. Computer Vision and Pattern Recognition (CVPR'16). Las Vegas, NV, USA : IEEE, 2016. 9 p.

15. Ren S. Faster R-CNN: towards real-time object detection with region proposal networks. International Conference on Neural Information Processing Systems. MIT Press, 2015. P. 91-99.

16. Rezatofighi S. Joint Probabilistic Data Association Revisited. International Conference on Computer Vision, 2015. P. 74-87

17. Richie Dorsey Machine Learning for Beginners: A Complete Guide for Getting Started with Machine Learning Kindle Edition. Publisher: Amazon.com Services LLC, 2019. 162 p.

18. Rokach L. Ensemble Learning: Pattern Classification Using Ensemble Methods. Singapore : World Scientific Publishing Co Pte Ltd., 2019. 302 p.

19. Stephen Marsland. Machine Learning: An Algorithmic Perspective. 2015. 452 p.

20. Tso B. Classification Methods for Remotely Sensed Data. Boca Raton : CRC Press, 2016. 352 p.

21. Uijlings J. Selective Search for Object Recognition. International Journal of Computer Vision. 2013 № 104(2). P. 154-171.

22. What Is Data Mining? 2022. URL : <https://medium.com/rightdata/what-is-data-mining-70978aedef079>

23. Wieczorek A.N. Analysis of the Possibility of Integrating a Mining Right-Angle Planetary Gearbox with Technical Diagnostics Systems. Scientific Journal of Silesian University of Technology. Series Transport. 2016. Vol. 93. P. 149-163.

24. Yaser S. Abu-Mostafa. Learning from data. 2017. 215 p.

25. Безсонов О. О. Еволюційні штучні нейронні мережі прямого

розповсюдження : архітектури, навчання, застосування : дис. ... д-ра техн. наук : 05.13.23. Харків, 2017. 40 с.

26. Винокурова О. А. Гібридні еволюційні адаптивні вейвлет-нейрофаззі-системи для динамічного аналізу даних : автореф. дис. ... д-ра техн. наук : 05.13.23. Харків, 2012. 40 с.

27. Вікторов Є. О. Гібридні еволюційні нейронні мережі та їх навчання : автореф. дис. ... канд. техн. наук : 05.13.23. Харків, 2011. 19 с.

28. Дубровін В. І., Субботін С. О. Методи оптимізації та їх застосування в задачах навчання нейронних мереж : навч. посіб. для студ. вищ. навч. закл. Запоріжжя : ЗНТУ, 2003. 136 с.

29. Іванченко Г. Ф. Системи штучного інтелекту : навч. посіб. Київ : КНЕУ, 2011. 382 с.

30. Ключин Д. А. Непараметричні методи розпізнавання з гарантованим рівнем значущості : дис. ... д-ра фіз.-мат. наук : 01.05.01. Київ, 2008. 278 с.

31. Козін І. В., Селютін Є. К. Особливості пошуку оптимальних класифікацій: еволюційні алгоритми. Вісник Запорізького національного університету. Фізико-математичні науки. Запоріжжя, 2019. №2. С. 62–68.

32. Кривохата А. Г., Кудін О. В., Лісняк А. О. Методи глибинного навчання у задачах машинного слуху. XIX Міжнародна конференція з математичного моделювання, присвячена 250-річчю з дня народження Жана Батиста Фур'є : матеріали конференції. (Херсон, 17-21 вересня 2018). Херсон, 2018. С. 70–71.

33. Кривохата А. Г., Кудін О. В., Лісняк А. О. Огляд методів машинного навчання для класифікації акустичних даних. Вісник Херсонського національного технічного університету. Херсон, 2018. Т. 1, № 3 (66). С. 327–331.

34. Кудін О. В., Кривохата А. Г. Генетичні алгоритми оптимізації ансамблів згорткових нейронних мереж. Комп'ютерні науки, інформаційні технології та системи управління : матеріали Міжнародної науково-технічної конференції здобувачів вищої освіти та молодих вчених. (Івано-Франківськ 27-

29 листопада 2019). Івано-Франківськ, 2019. С. 28.

35. Лебьодкіна А. Ю. Методи та моделі прискореної нейромережевої обробки даних у розподіленому обчислювальному середовищі : автореф. дис. ... канд. техн. наук : 05.13.23. Харків, 2012. 20 с.

36. Новотарський М. А., Нестеренко Б. Б. Штучні нейронні мережі : обчислення. Праці Інституту математики НАН України. Київ, 2004. Т. 50. 408 с.

37. Перепелиця В. О., Козін І. В., Терещенко Е. В. Задачі класифікації: підходи, методи, алгоритми. Запоріжжя : Поліграф, 2008. 188 с.

38. Руденко О. Г., Бодянський Є. В. Штучні нейронні мережі : навчальний посібник. Київ : Компанія СМІТ, 2006. 404 с.

39. Системи штучного інтелекту: нечітка логіка, нейронні мережі, нечіткі нейронні мережі, генетичний алгоритм : монографія / В. П. Лисенко, В. М. Решетюк, В. М. Штепа та ін. Київ : НУБіП України, 2014. 332 с.

40. Шоломицький О. Л. Застосування штучних нейронних мереж для розв'язання прикладних задач інженерної механіки. Перший крок у науку: матеріали ІХ студентської конференції, м. Суми, 25 лютого 2018 р. Суми: СумДУ, 2018. С. 279.

## Додаток А

### Лістинг 1. Навчання python розпізнати цифри.

Спочатку потрібно підключити необхідні бібліотеки, у нашому випадку це тенорф low. Також відключаємо виведення налагоджувальних повідомлень та роботу з GPU, вони нам не стануть у нагоді. Для роботи з масивами нам знадобиться бібліотека numpy.

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
os.environ['CUDA_VISIBLE_DEVICES'] = "-1"

від tensorflow import keras
від tensorflow.keras import layers
від tensorflow.keras import Sequential
з tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout,
Activation, BatchNormalization, AveragePooling2D
from tensorflow.keras.optimizers import SGD, RMSprop, Adam
import tensorflow_datasets as tfds # pip install tensorflow-datasets
import tensorflow as tf
import logging
import numpy as np

tf.logging.set_verbosity(tf.logging.ERROR)
tf.get_logger().setLevel(logging.ERROR)
```

Тепер ми готові створити неймережу. Завдяки Tensorflow на це знадобиться лише чотири рядки коду.

```
model = Sequential()
model.add(Dense(2, input_dim=2, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer=SGD(lr=0.1))
```

Ми створили модель нейронної мережі – клас Sequential – і додали до неї два шари: вхідний та вихідний. Така мережа називається багатошаровий перцептрон (multilayer perceptron).

У нашому випадку мережа має два входи (зовнішній шар), два нейрони у внутрішньому шарі та один вихід.

Можна подивитися, що в нас вийшло:

```
print(model.summary())
```

```
Neural network:
Model: "sequential"
```

| Layer (type)    | Output Shape | Param # |
|-----------------|--------------|---------|
| dense (Dense)   | (None, 2)    | 6       |
| dense_1 (Dense) | (None, 1)    | 3       |

```

Total params: 9
Trainable params: 9
Non-trainable params: 0

```

Навчання нейромережі полягає у знаходженні значень параметрів цієї мережі.

Наша мережа має дев'ять параметрів. Щоб навчити її, нам знадобиться вихідний набір даних, у разі це результати роботи функції XOR.

```
X = np.array([[0,0], [0,1], [1,0], [1,1]])
```

```
y = np.array([[0], [1], [1], [0]])
```

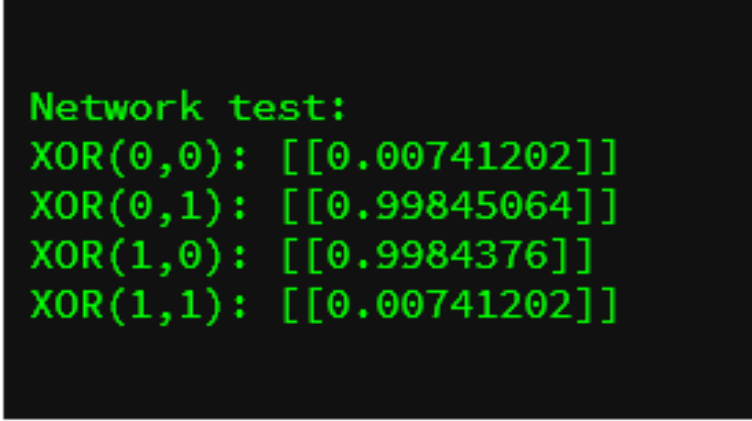
```
model.fit(X, y, batch_size=1, epochs=1000, verbose=0)
```

Функція fit запускає алгоритм навчання, яке у нас виконуватиметься тисячу разів, на кожній ітерації параметри мережі коригуватимуться. Наша мережа невелика, тому навчання пройде швидко. Після навчання мережею вже можна скористатися:

```
print("Network test:")
```

```
print("XOR(0,0):", model.predict_proba(np.array([[0, 0]])))
print("XOR(0,1):", model.predict_proba(np.array([[0, 1]])))
print("XOR(1,0):", model.predict_proba(np.array([[1, 0]])))
print("XOR(1,1):", model.predict_proba(np.array([[1, 1]])))
```

Результат відповідає тому, чому мережа навчалася.



```
Network test:
XOR(0,0): [[0.00741202]]
XOR(0,1): [[0.99845064]]
XOR(1,0): [[0.9984376]]
XOR(1,1): [[0.00741202]]
```

Лістинг: Ми можемо вивести всі значення знайдених коефіцієнтів на екран.

```
## Parameters layer 1
W1 = model.get_weights()[0]
b1 = model.get_weights()[1]
## Parameters layer 2
W2 = model.get_weights()[2]
b2 = model.get_weights()[3]

print("W1:", W1)
print("b1:", b1)
print("W2:", W2)
print("b2:", b2)
print()
```

```

W1: [[ 2.8668058 -2.904025 ] [-2.871452  2.9036295]]
b1: [-0.00128211 -0.00191825]
W2: [[3.9633768] [3.9168582]]
b2: [-4.897212]

```

Внутрішня реалізація функції `model.predict_proba` виглядає приблизно так:

```

x_in = [0, 1]
## Input
X1 = np.array([x_in], "float32")
## First layer calculation
L1 = np.dot(X1, W1) + b1
## Relu activation function: y = max(0, x)
X2 = np.maximum(L1, 0)
## Second layer calculation
L2 = np.dot(X2, W2) + b2
## Sigmoid
output = 1 / (1 + np.exp(-L2))

```

Розглянемо ситуацію, коли на вхід мережі подали значення `[0,1]`:

```

L1 = X1W1 + b1 = [02.8668058 + 1-2.871452 + -0.0012821, 0-2.904025 + 1*2.9036295 +
-0.00191825] = [-2.8727343  2.9017112]

```

Функція активації ReLu (rectified linear unit) – це заміна негативних елементів нулем.

```
X2 = np.maximum(L1, 0) = [0. 2.9017112]
```

Тепер знайдені значення попадають на другий шар.

```
L2 = X2W2 + b2 = 03.9633768 + 2.9017112*3.9633768 + -4.897212 = 6.468379
```

Нарешті, як вихід використовується функція Sigmoid, яка призводить значення до діапазону 0...1:

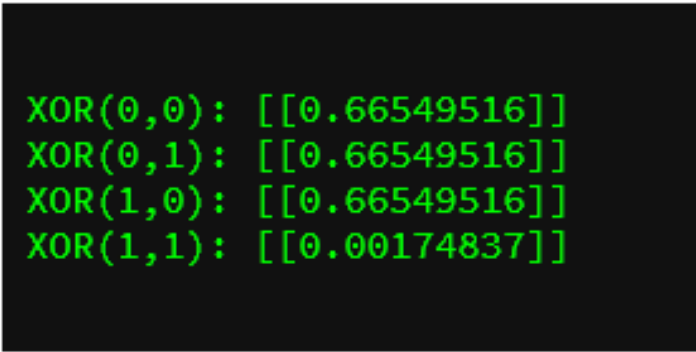
```
output = 1 / (1 + np.exp(-L2)) = 0.99845064
```

Ми здійснили звичайні операції множення та додавання матриць і отримали відповідь:  $XOR(0,1) = 1$ .

З цим прикладом на Python раджу поекспериментувати самостійно. Наприклад, ти можеш змінювати число нейронів у внутрішньому шарі.

Але алгоритм навчання, який використовується в Keras, не є ідеальним: нейромережі не завжди вдається навчитися за 1000 ітерацій, і результати не завжди вірні. Так, Keras ініціалізує початкові значення випадковими величинами і при кожному запуску результат може відрізнятись. Моя мережа з двома нейронами успішно навчалася лише у 20% випадків.

Неправильна робота мережі виглядає приблизно так:



```
XOR(0,0): [[0.66549516]]  
XOR(0,1): [[0.66549516]]  
XOR(1,0): [[0.66549516]]  
XOR(1,1): [[0.00174837]]
```

Але це не страшно. Якщо бачиш, що нейронна мережа під час навчання не видає правильних результатів, алгоритм навчання можна запустити вдруге. Правильно навчену мережу можна використовувати без обмежень.

## Навчання нейронної мережі

```
#команда для запуску навчання
./darknet detector train data/train.data cfg/yolov3.cfg darknet53.conv.74

#конфіг файл з архітектурою нейронної мережі [net]
# Testing
#batch=1
#subdivisions=1
# Training batch=64 subdivisions=32 width=992 height=480 channels=3
momentum = 0.9
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5 hue=.1
letter_box=1
learning_rate=0.001
burn_in=1000
max_batches = 20000
policy=steps
steps=16000,18000
scales=.1,.1
[Convolutional]
batch_normalize=1
filters=32
size=3
stride=1
pad=1
activation=leaky
# Downsample
[Convolutional]
batch_normalize=1
filters=64
size=3
stride=2
```

```
pad=1
activation=leaky
[convolutional] batch_normalize=1 filters=32
size=1
stride=1
pad=1
activation=leaky
[convolutional] batch_normalize=1 filters=64
size=3
stride=1
pad=1
activation=leaky
[shortcut]
from=-3
activation=linear
# Downsample
[convolutional] batch_normalize=1 filters=128
size=3
stride=2
pad=1
activation=leaky
[convolutional] batch_normalize=1 filters=64
size=1
stride=1
pad=1
activation=leaky
[convolutional] batch_normalize=1 filters=128
size=3
stride=1
pad=1
activation=leaky
[convolutional] batch_normalize=1 filters=64
size=1
stride=1
pad=1
```

```
activation=leaky
[convolutional] batch_normalize=1 filters=128
size=3
stride=1
pad=1
activation=leaky
[shortcut] from=-3
activation=linear
# Downsample
[convolutional] batch_normalize=1 filters=256
size=3
stride=2
pad=1
activation=leaky
[convolutional] batch_normalize=1 filters=128
size=1
stride=1
pad=1
activation=leaky
[convolutional] batch_normalize=1 filters=256
size=3
stride=1
pad=1
activation=leaky
[convolutional] batch_normalize=1 filters=128
size=1
stride=1
pad=1
activation=leaky
[convolutional] batch_normalize=1 filters=256
size=3
stride=1
pad=1
activation=leaky
[shortcut] from=-3
```

```

activation=linear
[convolutional] batch_normalize=1 filters=128
size=1
stride=1
pad=1
activation=leaky
[convolutional] batch_normalize=1 filters=256
size=3
stride=1
pad=1
activation=leaky
[shortcut]
from=-3
activation=linear
[convolutional] batch_normalize=1 filters=128
size=1
stride=1
pad=1
activation=leaky [convolutional] batch_normalize=1 filters=256
[shortcut]
from=-3
activation=linear
[convolutional] batch_normalize=1 filters=128 size=1
stride=1
pad=1
activation=leaky
[convolutional] batch_normalize=1 filters=256
size=3
stride=1
pad=1
activation=leaky
[shortcut] from=-3
activation=linear
[convolutional] batch_normalize=1 filters=128
size=1

```

```

stride=1
pad=1
activation=leaky
[convolutional] batch_normalize=1 filters=256
size=3
stride=1
pad=1
activation=leaky
[Convolutional]
batch_normalize=1 filters=128 size=1 stride=1
pad=1
activation=leaky
[convolutional] batch_normalize=1 filters=256 size=3
stride=1
pad=1
activation=leaky
[shortcut]
from=-3
activation=linear
[convolutional] batch_normalize=1 filters=128 size=1
stride=1
pad=1
activation=leaky
[convolutional] batch_normalize=1 filters=256 size=3
stride=1
pad=1
activation=leaky
[shortcut]
from=-3 activation=linear
# Downsample
[convolutional] batch_normalize=1 filters=512 size=3
stride=2
pad=1
activation=leaky [convolutional]
batch_normalize=1

```

```

filters=256
size=1
stride=1
pad=1
activation=leaky
[convolutional] batch_normalize=1 filters=512 size=3 stride=1
pad=1
activation=leaky
[shortcut]
from=-3
activation=linear
[convolutional] batch_normalize=1 filters=256 size=1
stride=1
pad=1
activation=leaky
[convolutional] batch_normalize=1 filters=512
size=3
stride=1
pad=1
activation=leaky
[shortcut] from=-3
activation=linear
[convolutional] batch_normalize=1 filters=256
size=1
stride=1
pad=1
activation=leaky [convolutional] batch_normalize=1 filters=512 [shortcut]
from=-3
activation=linear
[convolutional] batch_normalize=1 filters=256 size=1
stride=1
pad=1
activation=leaky
[convolutional] batch_normalize=1 filters=512
size=3

```

stride=1  
pad=1  
activation=leaky  
[shortcut] from=-3  
activation=linear  
[convolutional] batch\_normalize=1 filters=256  
size=1  
stride=1  
pad=1  
activation=leaky  
[convolutional] batch\_normalize=1 filters=512  
size=3  
stride=1  
pad=1  
activation=leaky  
[convolutional] batch\_normalize=1 filters=256  
size=1  
stride=1  
pad=1  
activation=leaky  
[convolutional] batch\_normalize=1 filters=512  
size=3  
stride=1  
pad=1  
activation=leaky  
[shortcut] from=-3  
activation=linear  
[convolutional] batch\_normalize=1 filters=256 size=1  
stride=1  
pad=1  
activation=leaky  
[convolutional] batch\_normalize=1 filters=512  
size=3  
stride=1  
pad=1

```

activation=leaky
[shortcut] from=-3
activation=linear
[convolutional] batch_normalize=1 filters=256
size=1
stride=1
pad=1
activation=leaky [convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky
[shortcut]
from=-3 activation=linear
# Downsample
[convolutional] batch_normalize=1 filters=1024 size=3
stride=2
pad=1
activation=leaky
[convolutional] batch_normalize=1 filters=512 size=1
stride=1
pad=1
activation=leaky
[convolutional] batch_normalize=1 filters=1024 size=3
stride=1
pad=1
activation=leaky
[shortcut]
from=-3
activation=linear
[convolutional] batch_normalize=1 filters=512 size=1
stride=1
pad=1

```

```

activation=leaky [convolutional] batch_normalize=1 filters=1024
[shortcut]
from=-3
activation=linear
[convolutional] batch_normalize=1 filters=512
size=1
stride=1
pad=1
activation=leaky
[convolutional] batch_normalize=1 filters=1024
size=3
stride=1
pad=1
activation=leaky
[shortcut]
from=-3
activation=linear
[convolutional] batch_normalize=1 filters=512
size=1
stride=1
pad=1
activation=leaky
[convolutional] batch_normalize=1 filters=1024
size=3
stride=1
pad=1
activation=leaky
[shortcut]
from=-3
activation=linear
#####
[convolutional] batch_normalize=1 filters=512 size=1
stride=1
pad=1
activation=leaky

```

```

[convolutional] batch_normalize=1 size=3
stride=1
pad=1
filters=1024 activation=leaky
[convolutional] batch_normalize=1 filters=512
size=1
stride=1
pad=1
activation=leaky
[convolutional] batch_normalize=1 size=3
stride=1
pad=1
filters=1024 activation=leaky
[convolutional] batch_normalize=1 filters=512
size=1
stride=1
pad=1
activation=leaky
[convolutional] batch_normalize=1 size=3
stride=1
pad=1
filters=1024 activation=leaky [convolutional]
size=1
stride=1
pad=1
filters=18
activation=linear
[yolo]
mask = 6,7,8
anchors = 4,7      3,6,   2,5   3,5   2,4
classes=1
num=5
jitter=.3
ignore_thresh = .7
truth_thresh = 1

```

```

random=1
[route]
layers =      -4
[convolutional] batch_normalize=1 filters=256
size=1
stride=1
pad=1
activation=leaky
[upsample] stride=2
[route]
layers =      -1,  61
[convolutional] batch_normalize=1 filters=256
size=1
stride=1
pad=1
activation=leaky
[convolutional] batch_normalize=1 size=3
stride=1
pad=1
filters=512
activation=leaky
[convolutional] batch_normalize=1 filters=256
size=1
stride=1
pad=1
activation=leaky
[convolutional] batch_normalize=1 size=3
stride=1
pad=1
filters=512
activation=leaky
[convolutional] batch_normalize=1 filters=256
size=1
stride=1
pad=1

```

```

activation=leaky
[convolutional] batch_normalize=1 size=3
stride=1
pad=1
filters=512
activation=leaky
[Convolutional]
size=1
stride=1
pad=1
filters=18
activation=linear
[yolo]
mask = 3,4,5
anchors = 4,7      3,6,   2,5   3,5   2,4
classes=1
num=5
jitter=.3
ignore_thresh =    .7
truth_thresh = 1
random=1
[route]
layers =    -4
[convolutional] batch_normalize=1 filters=128
size=1
stride=1
pad=1
activation=leaky
[upsample] stride=2
[route]
layers =    -1,  36
[convolutional] batch_normalize=1 filters=128 size=1
stride=1
pad=1
activation=leaky

```

```

[convolutional] batch_normalize=1 size=3
stride=1
pad=1
filters=256 activation=leaky
[convolutional] batch_normalize=1 filters=128
size=1
stride=1
pad=1
activation=leaky
[convolutional] batch_normalize=1 size=3
stride=1
pad=1
filters=256 activation=leaky
[convolutional] batch_normalize=1 filters=128 size=1
stride=1
pad=1
activation=leaky
[convolutional] batch_normalize=1 size=3
stride=1
pad=1
filters=256 activation=leaky
[Convolutional] size=1
stride=1
pad=1
filters=18
activation=linear
[yolo]
mask = 0,1,2
anchors = 4,7      3,6,   2,5   3,5   2,4
classes=1
num=5
jitter=.3
ignore_thresh =    .7
truth_thresh = 1
random=0

```

## Додаток В

**Лістинг 2. Реалізація нейронної мережі**

| Ім'я/Name | Вага/Weight (мінус 135) | Зріст/Height (мінус 66) | Стать/Gender |
|-----------|-------------------------|-------------------------|--------------|
| Alice     | -2                      | -1                      | 1            |
| Bob       | 25                      | 6                       | 0            |
| Charlie   | 17                      | 4                       | 0            |
| Diana     | -15                     | -6                      | 1            |

```

1  import numpy as np
2
3
4  def sigmoid(x):
5      # Функція активації sigmoid:  $f(x) = 1 / (1 + e^{-x})$ 
6      return 1 / (1 + np.exp(-x))
7
8
9  def deriv_sigmoid(x):
10     # Похідна від sigmoid:  $f'(x) = f(x) * (1 - f(x))$ 
11     fx = sigmoid(x)
12     return fx * (1 - fx)
13
14
15 def mse_loss(y_true, y_pred):
16     # y_true та y_pred є масивами numpy з однаковою довжиною
17     return ((y_true - y_pred) ** 2).mean()
18
19
20 class OurNeuralNetwork:
21     """
22     Нейронна мережа, яка має:
23     - 2 входи
24     - прихований шар із двома нейронами ( $h_1, h_2$ )
25     - шар виведення з одним нейроном ( $o_1$ )
26
27     *** ВАЖЛИВО ***:
28     Код нижче написаний як простий, освітній. НЕ оптимальний.
29     Цей код нейронної мережі виглядає не так. НЕ ВИКОРИСТОВУЙТЕ цей

```

```

30 код.
31 Натомість прочитайте/запустіть його, щоб зрозуміти, як працює ця мережа.
32 """
33 def __init__(self):
34     # Ваги
35     self.w1 = np.random.normal()
36     self.w2 = np.random.normal()
37     self.w3 = np.random.normal()
38     self.w4 = np.random.normal()
39     self.w5 = np.random.normal()
40     self.w6 = np.random.normal()
41
42     # Зміщення
43     self.b1 = np.random.normal()
44     self.b2 = np.random.normal()
45     self.b3 = np.random.normal()
46
47     def feedforward(self, x):
48         # x є масивом numpy із двома елементами
49         h1 = sigmoid(self.w1 * x[0] + self.w2 * x[1] + self.b1)
50         h2 = sigmoid(self.w3 * x[0] + self.w4 * x[1] + self.b2)
51         o1 = sigmoid(self.w5 * h1 + self.w6 * h2 + self.b3)
52         return o1
53
54     def train(self, data, all_y_trues):
55         """
56         - data is a (nx 2) numpy array, n = # samples in the dataset.
57         - all_y_trues is a numpy array with n elements.
58           Elements in all_y_trues відповідають those in data.
59         """
60         learn_rate = 0.1
61         epochs = 1000 # кількість циклів у всьому наборі даних
62
63         for epoch in range(epochs):
64             для x, y_true в zip(data, all_y_trues):
65                 # --- Виконуємо зворотний зв'язок (нам знадобляться ці значення надалі)
66                 sum_h1 = self.w1 * x [0] + self.w2 * x [1] + self.b1
67                 h1 = sigmoid(sum_h1)

```

```

68
69 sum_h2 = self.w3 * x [0] + self.w4 * x [1] + self.b2
70 h2 = sigmoid(sum_h2)
71
72 sum_o1 = self.w5 * h1 + self.w6 * h2 + self.b3
73 o1 = sigmoid(sum_o1)
74 y_pred = o1
75
76 # --- Підрахунок приватних похідних
77 # --- Найменування: d_L_d_w1 є "частково L / частково w1"
78 d_L_d_ypred = -2 * (y_true - y_pred)
79
80 # Нейрон o1
81 d_ypred_d_w5 = h1 * deriv_sigmoid(sum_o1)
82 d_ypred_d_w6 = h2 * deriv_sigmoid(sum_o1)
83 d_ypred_d_b3 = deriv_sigmoid(sum_o1)
84
85 d_ypred_d_h1 = self.w5 * deriv_sigmoid(sum_o1)
86 d_ypred_d_h2 = self.w6 * deriv_sigmoid(sum_o1)
87
88 # Нейрон h1
89 d_h1_d_w1 = x[0] * deriv_sigmoid(sum_h1)
90 d_h1_d_w2 = x[1] * deriv_sigmoid(sum_h1)
91 d_h1_d_b1 = deriv_sigmoid(sum_h1)
92
93 # Нейрон h2
94 d_h2_d_w3 = x[0] * deriv_sigmoid(sum_h2)
95 d_h2_d_w4 = x[1] * deriv_sigmoid(sum_h2)
96 d_h2_d_b2 = deriv_sigmoid(sum_h2)
97
98 # --- Оновлюємо вагу та зміщення
99 # Нейрон h1
100 self.w1 -= learn_rate * d_L_d_ypred * d_ypred_d_h1 * d_h1_d_w1
101 self.w2 -= learn_rate * d_L_d_ypred * d_ypred_d_h1 * d_h1_d_w2
102 self.b1 -= learn_rate * d_L_d_ypred * d_ypred_d_h1 * d_h1_d_b1
103
104 # Нейрон h2
105 self.w3 -= learn_rate * d_L_d_ypred * d_ypred_d_h2 * d_h2_d_w3

```

```

106 self.w4 -= learn_rate * d_L_d_ypred * d_ypred_d_h2 * d_h2_d_w4
107 self.b2 -= learn_rate * d_L_d_ypred * d_ypred_d_h2 * d_h2_d_b2
108
109 # Нейрон  $o_1$ 
110 self.w5 -= learn_rate * d_L_d_ypred * d_ypred_d_w5
111 self.w6 -= learn_rate * d_L_d_ypred * d_ypred_d_w6
112 self.b3 -= learn_rate * d_L_d_ypred * d_ypred_d_b3
113
114 # --- Підраховуємо загальну втрату наприкінці кожної фази
115 if epoch % 10 == 0:
116     y_preds = np.apply_along_axis(self.feedforward, 1, data)
117     loss = mse_loss(all_y_trues, y_preds)
118     print("Epoch %d loss: %.3f" % (epoch, loss))
119
120
121 # Визначення набору даних
122 data = np.array([
123     [-2, -1], # Alice
124     [25, 6], # Bob
125     [17, 4], # Charlie
126     [-15, -6], # Diana
127 ])
128
129 all_y_trues = np.array([
130     1, # Alice
131     0, # Bob
132     0, # Charlie
133     1, # Diana
134 ])
135
136 # Тренуємо нашу нейронну мережу!
137 network = OurNeuralNetwork()
    network.train(data, all_y_trues)

```

### Конфігурація нейронної мережі YOLOv3

| Count | Type          | Filters | Size/Stride | Input           | Output          |
|-------|---------------|---------|-------------|-----------------|-----------------|
| 1     | Convolutional | 32      | 3 x 3/1     | 992 x 480 x 3   | 992 x 480 x 32  |
|       | Convolutional | 64      | 3 x 3/2     | 992 x 480 x 32  | 992 x 480 x 64  |
|       | Convolutional | 32      | 1x 1/1      | 992 x 480 x 64  | 496 x 240 x 32  |
|       | Convolutional | 64      | 3 x 3/1     | 496 x 240 x 32  | 496 x 240 x 64  |
|       | Shortcut      |         |             |                 |                 |
| 1     | Convolutional | 128     | 3 x 3/2     | 496 x 240 x 64  | 248 x 120 x 128 |
| 2     | Convolutional | 64      | 1x 1/1      | 248 x 120 x 128 | 248 x 120 x 64  |
|       | Convolutional | 128     | 3 x 3/1     | 248 x 120 x 64  | 248 x 120 x 128 |
|       | Shortcut      |         |             |                 |                 |
| 1     | Convolutional | 256     | 3 x 3/2     | 248 x 120 x 128 | 124 x 60 x 256  |
| 8     | Convolutional | 128     | 1x 1/1      | 124 x 60 x 256  | 124 x 60 x 128  |
|       | Convolutional | 256     | 3 x 3/1     | 124 x 60 x 128  | 124 x 60 x 256  |
|       | Shortcut      |         |             |                 |                 |
| 1     | Convolutional | 512     | 3 x 3/2     | 124 x 60 x 256  | 62 x 30 x 512   |
| 8     | Convolutional | 256     | 1x 1/1      | 62 x 30 x 512   | 62 x 30 x 256   |
|       | Convolutional | 512     | 3 x 3/1     | 62 x 30 x 256   | 62 x 30 x 512   |
|       | Shortcut      |         |             |                 |                 |
| 1     | Convolutional | 1024    | 3 X 3/2     | 62 X 30 X 512   | 31 X 15 X 1024  |
| 4     | Convolutional | 512     | 1X 1/1      | 31 X 15 X 1024  | 31 X 15 X 512   |
|       | Convolutional | 1024    | 3 X 3/1     | 31 X 15 X 512   | 31 X 15 X 1024  |
|       | Shortcut      |         |             |                 |                 |
| 3     | Convolutional | 512     | 1X 1/1      | 31 X 15 X 1024  | 31 X 15 X 512   |
|       | Convolutional | 1024    | 3 X 3/1     | 31 X 15 X 512   | 31 X 15 X 1024  |
| 1     | Convolutional | 68      | 1X 1/1      | 31 X 15 X 1024  | 31 X 15 X 68    |

|   |               |     |         |                |                |
|---|---------------|-----|---------|----------------|----------------|
|   | Yolo          |     |         |                |                |
|   | Route         |     |         |                |                |
|   | Convolutional | 256 | 1X 1/1  | 31 X 15 X 68   | 31 X 15 X 256  |
|   | Upsample      |     | X 2     | 31 X 15 X 256  | 62 X 30 X 256  |
|   | Route         |     |         |                |                |
| 3 | Convolutional | 256 | 1X 1/1  | 62 X 30 X 256  | 62 X 30 X 256  |
|   | Convolutional | 512 | 3 X 3/1 | 62 X 30 X 256  | 62 X 30 X 512  |
| 1 | Convolutional | 68  | 1X 1/1  | 62 X 30 X 512  | 62 X 30 X 68   |
|   | Yolo          |     |         |                |                |
|   | Route         |     |         |                |                |
|   | Convolutional | 256 | 1X 1/1  | 62 X 30 X 68   | 62 X 30 X 128  |
|   | Upsample      |     | X 2     | 62 X 30 X 128  | 124 X 60 X 128 |
|   | Route         |     |         |                |                |
| 3 | Convolutional | 128 | 1X 1/1  | 124 X 60 X 128 | 124 X 60 X 128 |
|   | Convolutional | 256 | 3 X 3/1 | 124 X 60 X 128 | 124 X 60 X 256 |
| 1 | Convolutional | 68  | 1X 1/1  | 124 X 60 X 256 | 124 X 60 X 68  |
|   | Yolo          |     |         |                |                |

### Код для запуску роботи нейронної мережі

```
#запуск роботи нейронної мережі
import operator
від ctypes import *
import math
import random
import os
from queue import Queue
від PIL import Image, ImageDraw
від os.path import join

import cv2
import numpy as np
import time
import darknet
від accidents import AccidentCounter
from database import UpdateVisionData, fill_dict, fill_speeddict,
TaskThreadQueue, InsertAccidentsCount, \
    UpdateAccidentsCount, UpdateSpeedData
import settings

від sort import *

від shapely.geometry import Point
від shapely.geometry.polygon import Polygon

class_to_color = {
    1: (26, 66, 180),
}
def draw_rect(image, space_x, space_y, c):
    cv2.rectangle(image, (space_x[0], space_y[0]), (space_x[1],
space_y[1]), class_to_color[c], 1)
    return image

def convertBack(x, y, w, h):
    xmin = int(round(x - (w / 2)))
    xmax = int(round(x + (w / 2)))
    ymin = int(round(y - (h / 2)))
    ymax = int(round(y + (h / 2)))
    return xmin, ymin, xmax, ymax

def parse_yolo_data(path, ext):
    files = os.listdir(path)
    ext_files = [i for i in files if ext in i]
    if len(ext_files) != 1:
        raise Exception("File *%s must be a single" % ext)
```

```

    return path + ext_files[0]

YOLO_DATA_PATH = "yolo_data/people/"

def YOLO():
    Global metaMain, netMain, altNames, YOLO_DATA_PATH, mask configPath =
    parse_yolo_data(YOLO_DATA_PATH, '.cfg')
    metaPath = parse_yolo_data(YOLO_DATA_PATH, '.data')
    weightPath = parse_yolo_data(YOLO_DATA_PATH, '.weights')

    if not os.path.exists(configPath):
        raise ValueError("Invalid config path" + os.path.abspath(configPath) + )
    if not os.path.exists(weightPath):
        raise ValueError("Invalid weight path" + os.path.abspath(weightPath) + )
    if not os.path.exists(metaPath):
        raise ValueError("Invalid data file path" + os.path.abspath(metaPath) + )
    if netMain is None:
        netMain = darknet.load_net_custom(configPath.encode(
            "ascii"), weightPath.encode("ascii"), 0, 1) # batch size
= 1
    if metaMain is None:
        metaMain = darknet.load_meta(metaPath.encode("ascii"))
    if altNames is None:
        try:
            with open(metaPath) as metaFH:
                metaContents = metaFH.read() import re
                match = re.search("names *= *(.*)$", metaContents, re.IGNORECASE |
                    re.MULTILINE)
                if match:
                    result = match.group(1)
                else:
                    result = None
            try:
                if os.path.exists(result):
                    with open(result) as namesFH:
                        namesList =
namesFH.read().strip().split("\n")
                        altNames = [x.strip() for x in namesList]
            except TypeError:
                pass
            except Exception:
                pass
        url = "rtsp://tdstream.is74.ru/p/pobkras"
        cap = cv2.VideoCapture(url)
        darknet_image = darknet.make_image(darknet.network_width(netMain),

darknet.network_height(netMain), 3)
        mot_tracker = Sort()

        while True:
            try:
                ret, frame_read = cap.read()

```

```

        if frame_read is None or not ret:
            cap = cv2.VideoCapture(url)
            continue
        frame_rgb = cv2.cvtColor(frame_read,
cv2.COLOR_BGR2RGB)
        frame_resized = cv2.resize(frame_rgb,

(darknet.network_width(netMain),

darknet.network_height(netMain)),

interpolation=cv2.INTER_LINEAR)
        frame_src = frame_resized.copy()
    except:
        cap = cv2.VideoCapture(url)
        continue
    darknet_frame = frame_resized

    darknet.copy_image_from_bytes(darknet_image, darknet_frame.tobytes()) detections =
        darknet.detect_image(netMain, metaMain,
darknet_image, thresh = 0.1)
    det = []
    dets = np.array(det)
    for detection in detections:
        x, y, w, h, score = detection[2][0], \
                                detection[2][1], \
                                detection[2][2], \
                                detection[2][3], \
                                detection[1]
        xmin, ymin, xmax, ymax = convertBack(
            float(x), float(y), float(w), float(h))
        pt1 = (xmin, ymin)
        pt2 = (xmax, ymax)
        det.append([xmin, ymin, xmax, ymax, score])

    dets = np.array(det)
    trackers = mot_tracker.update(dets)
    for x1, y1, x2, y2, obj_id, dx1, dy1, dx2, dy2 in trackers:
        __x1, __y1, __x2, __y2 = x1, y1, x2, y2
        draw_rect(frame_resized, [int(__x1), int(__x2)], [int(__y1),
int(__y2)], det_full[indexx[0], 3])
        image = frame_resized
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        cv2.imshow('frame', image)
        ch = 0xFF & cv2.waitKey(1)
        if ch == 27:
            break

    cap.release()

if __name__ == "__main__":
    try:
        YOLO()

```

```
except Exception as e: print(str(e)) exit(1)
```