Міністерство освіти і науки України
Криворізький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерних систем та мереж

Пояснювальна записка
до кваліфікаційної роботи магістра
за спеціальністю 123 «Комп'ютерна інженерія»

на тему: МЕТОДИ МАШИННОГО НАВЧАННЯ ДЛЯ ОБРОБКИ ВЕЛИКИХ МАСИВІВ ДАНИХ

Проектував	_____	Д. С. Кондратьєв
Керівник роботи	_____	О. Г. Руденко
Консультант	_____	А. О. Сенько
Нормконтроль	_____	Д. І. Кузнецов
Завідувач кафедри	_____	А. І. Купін

Кривий Ріг
2022

Криворізький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерних систем та мереж

Ступінь освіти магістр

Спеціальність 123 «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ

Завідувач кафедри, голова циклової комісії

_____ А. І. Купін

"___" _____ 20__ року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Кондратьєв Данило Станіславович
(прізвище, ім'я, по батькові)

1. Тема роботи методи машинного навчання для обробки великих масивів даних
керівник роботи Руденко О. Г., проф.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
2. Строк подання студентом роботи 25.11.2022
3. Вихідні дані до роботи Бінарна класифікація електроних листів на спам
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити).
Проаналізувати існуючи алгоритми в машинному навчанні. Розібрати найпопулярніші метрики навчання. Практично перевірити різні підходи до обробки навчальних даних. Реалізувати модель глибокого навчання для бінарної класифікації.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень). Презентація (Х слайдів).
6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1			
2			
3			

7. Дата видачі завдання 02.02.2021.

Календарний план

№	Назва етапів роботи	Строк виконання етапів роботи
1		
2		
3		
4		
5		
6		
7		

Студент _____ _____

(підпис) (призвище та ініціали)

Керівник роботи _____ _____

(підпис) (призвище та ініціали)

Реферат

Пояснювальна записка: 95 сторінок, 90 рисунків, 5 таблиць, 22 використаних джерел.

Проєкт складається з шістьох розділів.

У першому розділі розглянуто теоретичні аспекти машинного навчання та глибокого навчання. Розібрано найпопулярніші метрики для різних типів задач. Проговорено про типи нейронних мереж, їх особливості та відповідне використання.

У другому розділі розглянуто сучасні інструменти для розробки моделей машинного навчання. Найпопулярніша мова програмування Python та одна з її найпопулярніших бібліотек Keras/Tensorflow для Машиного навчання, надають усі необхідні інструменти для швидкої розробки моделей придатних для використання у великих проєктів.

У третьому розділі реалізовано програму моделі глибокого навчання для бінарної класифікації поштового спаму. Розібрано альтернативні підходи до побудови архітектури моделі для даної задачі. Виявлено сильні та слабкі сторони використаного набору даних.

Abstract

Explanatory note: 95 pages, 90 figures, 5 tables, 22 used sources.

The project consists of six sections.

In the first section, we have examined the theoretical aspects of the concept of machine and deep learning. The most popular metrics for different types of tasks are analyzed. We talked about the types of neural networks, their features and appropriate use.

The second section presents modern tools for developing machine learning models. The most popular programming language, Python, and one of its most popular machine learning libraries, Keras/Tensorflow, provide all the tools to quickly develop models suitable for use in large projects.

In the third section, the program of the deep learning model for the binary classification of email spam is implemented. Alternative approaches to building the model architecture for this task are analyzed. The strengths and weaknesses of the used data set were identified.

Зміст

Перелік скорочень

1. ІН — істинно негативні
2. ІП — істинно позитивні
3. ГНМ — глибокі нейронні мережі
4. ПЗШ — повнозв’язні шари мережі

Вступ

Використання комп’ютерів для полегшення праці людини вже триває багато років, але все ще залишаються області в яких людину не замінити. З розвитком штучного інтелекту (artificial intelligence) все більше задач можна буде покласти на потужності комп’ютеризованого обладнання. Так, наприклад, за останній час дуже швидко прогресує глибоке навчання (deep learning). Штучний інтелект виграє нагороди у людських конкурсах мистецтва; з’являються технології підвищення якості та роздільної здатності зображень і відео в реальному часі, і це тільки декілька прикладів.

Майбутнє це машинне навчання. Інтерес до машинного навчання стрімко зростає з кожним днем із застосуванням майже в усіх галузях, від медицини до банківської справи, від безпілотних автомобілів до замовлення кави.

Штучний інтелект — загально кажучи, це коли комп’ютер приймає рішення, імітуючи способи прийняття рішень людиною. В інших випадках він може імітувати еволюційні процеси, генетичні процеси або фізичні процеси. Але загалом, щоразу, коли ми бачимо, як комп’ютер сам вирішує проблему, будь то керування автомобілем, пошук маршруту між двома точками, діагностування пацієнта чи рекомендація фільму то це штучний інтелект.

Машинне навчання — це підрозділ штучного інтелекту, який фокусується на прийнятті рішень базуючись на даних.

Глибоке навчання — це підрозділ машинного навчання який фокусується на використанні спеціальних об’єктів, нейронних мереж.

Актуальність роботи. За останні 10 років найефективніші системи штучного інтелекту, такі як розпізнавання мовлення на смартфонах або найновіший автоматичний перекладач Google, були створені завдяки техніці під назвою «глибоке навчання».

Глибоке навчання — це фактично нова назва для підходу до штучного інтелекту під назвою нейронні мережі, які то входили, то виходили з моди вже понад 70 років. Нейронні мережі вперше були запропоновані в 1944 році Уорреном Маккалоу та Волтером Піттсом, двома дослідниками Чиказького університету, які переїхали до Массачусетського технологічного інституту в 1952 році як члени-засновники того, що іноді називають першим відділом когнітивної науки.

Нейронні мережі були основною сферою досліджень як у нейронауці, так і в інформатиці до 1969 року. Потім ця техніка відродилася у 1980-х роках, знову зазнала занепаду в першому десятилітті двухтисячних і повернулася у другому, головним чином завдяки збільшенню процесорної потужності графічних чіпів.

З розквітом глибокого навчання задачі які раніше були нерозв’язаними або існуючи рішення були недостатньо хорошими, стали вирішуватися з допомогою цієї технології. З популяризацію інетрента, а потім і створенню електроної пошти почалися проблеми з небажаними листами. Це була перша форма спаму, спам на електронну пошту. В ті часи не було ефективних методів боротьби зі спамом.

У ретроспективі, наявні зараз технології можуть легко вирішити проблему текстового спаму, що буде показано в даній роботі.

Мета і завдання дослідження. Основною метою дослідження є огляд математичної бази нейронних мереж, їх типів принципів функціонування та навчання.

Серед головних завдань дослідження є такі: зробити детальний опис коду для практичного рішення проблеми та інструментів створення цього коду; опис екосистеми пакетів та інструментів навколо машинного навчання та глибокого навчання; визначення ступені задоволення від програмного вирішення з точки зору кінцевого користувача за допомогою метрик.

Об’єкт дослідження. Процеси навчання глибоких нейронних мереж, внутрішне представлення тензорів і те як вони навчаються новим даним.

Предмет дослідження. Моделі та засоби створення штучних нейронних мереж, їх вплив на метрики точності, влучності, повноти.

Методи досліджень. Використовуючи системний аналіз, аналітичні методи та реалізовану програму встановити параметри оптимізації нейронної мережі за такими факторами: кількість навчальних епох алгоритму навчання; змінна функції активації; зміна архітектури мережі.

Наукова новизна одержаних результатів. Визначення впливу параметрів і гіперпараметрів моделі на її ефективність та інші метрики на прикладі бінарної класифікації.

Практичне значення отриманих результатів. Результати досліджень дозволяють правильно скоригувати параметри і гіперпараметри моделі для пришвидшення її навчання, а також збільшення її ефективності.

1 Теоретичні відомості про глибоке навчання

Машинне навчання всюди. Це твердження стає більш правдивим з кожним днем. Важко уявити хоч один аспект життя, який не може бути покращений тим чи іншим способом штучного інтелекту. Для будь-якої роботи, яка потребує повторення або аналізу даних і висновків, машинне навчання може допомогти. Протягом останніх кількох років машинне навчання досягло величезних успіхів зростання завдяки прогресу в обчислювальній потужності та великої кількості даних.

1.1 Машинне навчання

Машинне навчання інколи плутають з штучним інтелектом, але це не так. Машинне навчання це частина штучного інтелекту.

Рисунок 1.1 — Машинне навчання як розділ штучного інтелекту

Машинне навчання працює роблячи рішення базуючись на даних. Це не дуже зрозуміле пояснення, тому, як приклад можна навести як люди роблять рішення.

Людина може щось вирішити використовуючи:

- логіку та міркування;
- досвід.

Наприклад, якщо купувати телефон, то можна дослідити функції телефонів такі як: ціна; потужність процесору; потужність графічного процесору; частота та роздільна здатність екрану; тощо. А потім обрати таку комбінацію функцій яка б задовольняла бюджет на покупку. Це використання логіки та міркування.

В іншому випадку, можна подивитися огляди від інших людей, та створити лист з найбільш популярними моделями телефонів, їх плюсами та мінусами. Це буде використання досвіду, але в машинному навчанні, термін досвід означає данні.

Це може здатися як не значний спосіб вирішення проблем. В такому випадку подивимося як зазвичай вирішують проблеми за допомогою комп'ютера. Якщо треба заставити комп'ютер зробити те що хоче людина, хтось повинен написати програму для цього, тобто набір інструкцій які комп'ютер виконає для отримання результату. Але деякі проблеми занадто складно виразити звичайними алгоритмами, наприклад, задача ідентифікації об'єкта на картинці.

Для того щоб навчитися відрізняти об'єкти, нам показували щось та казали його назву поки ми не запам'ятали що об'єкт з такими особливостями має таку назву. Майже те саме відбувається і при машинному навчанні. Ми показуємо комп'ютеру багато картинок і кажемо на яких є потрібний нам об'єкт, а на яких немає, і так до тих пір коли поки комп'ютерний алгоритм не вивчить набір особливостей потрібного нам об'єкту. Звісно писати програму все одно треба, та є багато різних алгоритмів[9].

1.2 Глибоке навчання

Подібно до того, як машинне навчання є частиною штучного інтелекту, глибоке навчання є частиною машинного навчання.

Нейронні мережі, також звані багатошаровими перцептронами. Нейронні мережі є однією з найпопулярніших, якщо не найпопулярніших, варіантів машинного навчання. Вони настільки корисні, що ця сфера має власну назву: глибоке навчання. Глибоке навчання має численні застосування в найсучасніших сферах машинного навчання, включаючи розпізнавання зображень, обробку мов та медицину. Нейронні мережі покликані, у широкому сенсі цього слова, імітувати роботу людського мозку.

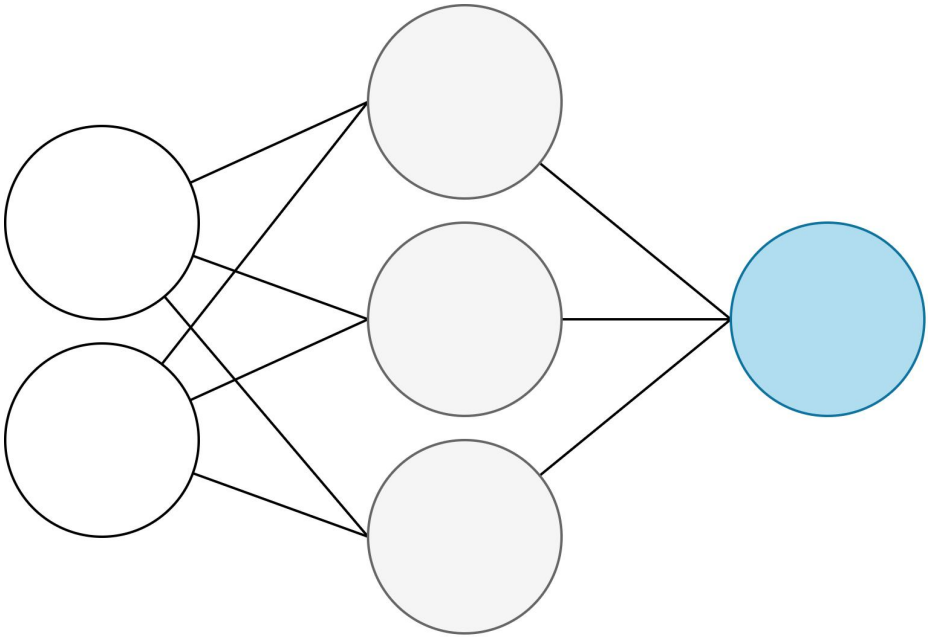

Рисунок 1.2 — Умовне позначення нейронної мережі

Існує два види машинного навчання, а так як нейроні мережі це підмножина машинного навчання, то це стосується їх теж.

1.3 Машинне навчання під наглядом

Машинне навчання під наглядом (supervised) — це імітація відношення або кореляції між двома наборами даних. Цей тип завжди намагається взяти вхідний набір даних і перетворити його на вихідний набір даних. Цей тип може бути неймовірно потужним та корисним інструментом. Розглянемо наступні приклади (вхідні набори даних виділено жирним шрифтом, вихідні набори даних виділено курсивом):

- Використання **пікселів зображення** для виявлення *присутності* чи *відсутності* кота;
- Використовуючи **фільми, які вам сподобалися**, передбачте більше *фільмів, які вам можуть сподобатися;*
- Використання **даних датчиків** погоди для прогнозування *ймовірності дощу*;
- Використання **датчиків двигуна** автомобіля для прогнозування *оптимальних налаштувань налаштування;*
- Використання **даних новин** для прогнозування завтрашньої *ціни* акцій;
- Використання введеного **числа** для передбачення подвоєння *числа*;
- Використання необробленого **аудіо файлу** для прогнозування *транскрипції* аудіо;

Усе це приклади машинного навчання під наглядом. У всіх випадках алгоритм машинного навчання намагається імітувати шаблон між двома наборами даних таким чином, щоб він міг використовувати один набір даних для прогнозування іншого [11].

1.4 Машинне навчання без нагляду

Такий вид має спільну властивість із навчанням з наглядом: воно перетворює один набір даних в інший. Але набір даних, у який він перетворюється, раніше не відомий і не зрозумілий. На відміну від навчання під наглядом, немає «правильної відповіді», яку, модель, намагаєтеся змусити дублювати. Він просто шукає за алгоритмом «знайти шаблони в цих даних і показати їх».

Наприклад, кластеризація набору даних у групи є типом навчання без нагляду. Кластеризація перетворює вхідні дані у набори груп кластера. Кожному рядку вхідних даних буде присвоєно номер залежно від того, в якому кластері він знаходиться. Таким чином, набір даних перетворюється з купи рядків даних на купу груп. Алгоритм не говорить що таке групи, тому, вони нумеруються просто цифрами. Та й звідки алгоритм міг це знати? Він просто говорить: “Гей, вчений! Я знайшов якусь кореляцію. Схоже, у ваших даних є групи. Ось вони!”.

Рисунок 1.3 — Машинне навчання без нагляду групує дані

Ідея кластеризації загалом розповсюджується на все навчання без нагляду. Незважаючи на те, що існує багато форм такого навчання, усі їх можна розглядати як форму кластеризації.

Незважаючи на те, що алгоритм, не може сказати назву кластеру, можна розділити вхідні дані (слова) на групи, а тобто тварини та їжа.

1.4 Параметричне та не параметричне навчання

Параметричне навчання характеризується фіксованою кількістю параметрів, тоді як кількість параметрів непараметричного навчання є нескінченною, тобто визначається даними.

Стандартна глибока нейронна мережа (Deep Neural Network) є параметричною, оскільки має фіксовану кількість параметрів. Однак більшість DNN мають стільки параметрів, що їх можна інтерпретувати як непараметричні; було доведено, що в межах нескінченної ширини, глибоку нейронну мережу можна розглядати як процес Гауса, який є непараметричною моделлю[1].

1.5 Розповсюдження вперед

Для того щоб уявити нейронну мережу, можна взяти за приклад рисунок 1.x. На вхід мережі подається масив даних, зазвичай в матричній формі, тобто така форма що складається зі стовпчиків та рядків. Стовпчики означають різні змінні, так наприклад, показники кожного сенсора будуть в окремому стовпчику. Рядки ж пов’язують стовпчики. Частіше за все один рядок це виміри виконані в один і той же момент часу.

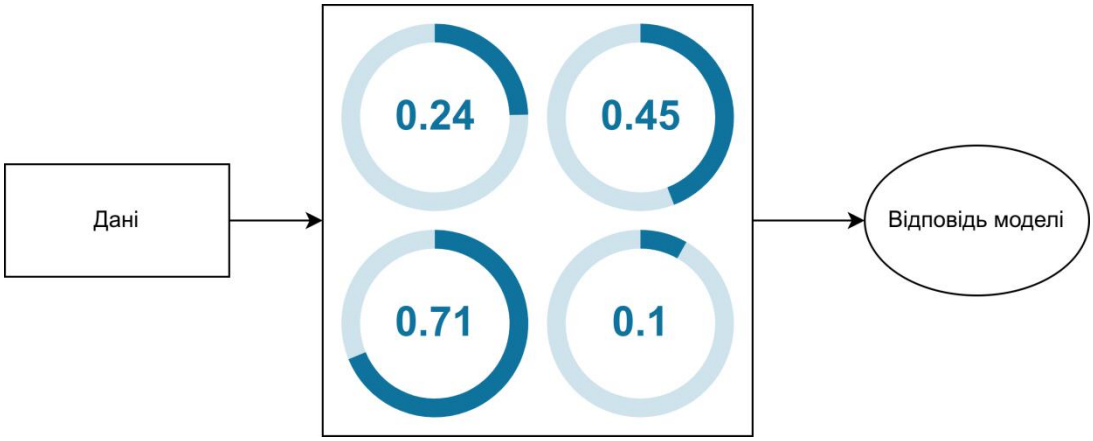

Рисунок 1.4 — Ваги нейронної мережі

Найпростіша нейрона мережа складається з одного нейрона. Для того щоб отримати передбачення, тобто відповідь моделі, вхідне число множиться на вагу нейрона, а так як в даному випадку нейрон один, він одразу посилає на вихід результат множення — число 1.8.

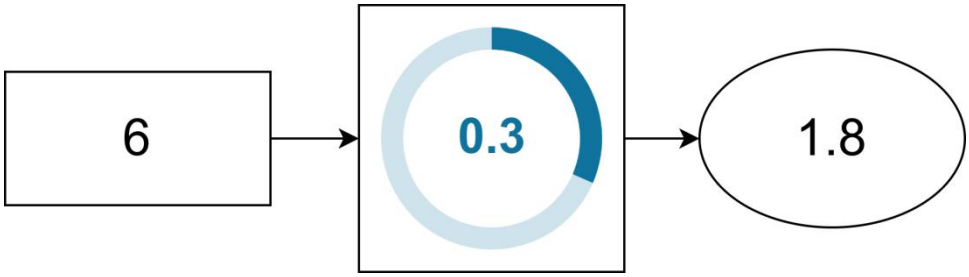

Рисунок 1.5 — Найпростіша нейронна мережа

Таку модель можна представити у вигляді звичайної функції

$$F(input,\ weight) = input \cdot weight \tag{1.1}$$

Тому, передбачення усіх нейронних мереж це лише результат математичних дій над вхідними даними. Далі розглянемо як формується передбачення коли вхідні дані це вектор із більше ніж одного значення.

На наступному рисунку показано нейронну мережу в більш поширеному візуальному форматі. На кожен вхід (білі кола) подається одне значення, або один вектор (рядок) вхідних даних. Кожне з'єднання з вихідним нейроном має вагу, підписану над лінією з'єднання. Значення на вихідному нейроні це скалярний добуток (dot product) вхідного вектору.

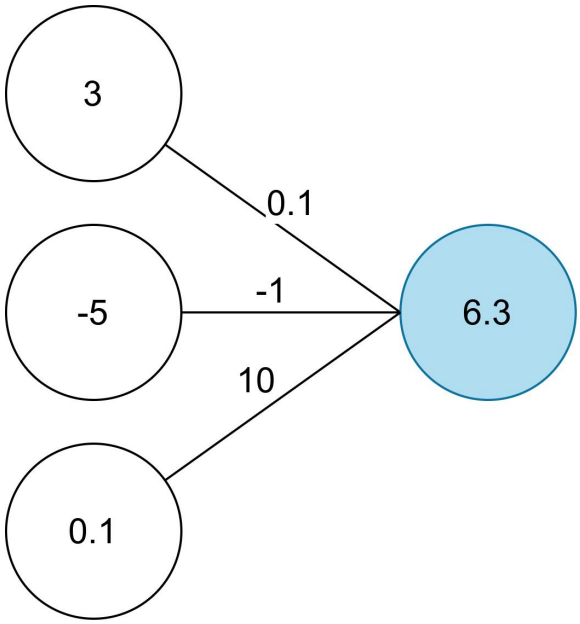

Рисунок 1.6 — Нейронна мережа на вхід якої подається вектор

Виражаючись мовою математичних формул, отримання скалярного добутку, числа 6.3 вимагає такі дії:

$$F(\overline{A},\ \overline{W}) = \overline{A} \cdot \overline{B} = A_1 \cdot W_1 + A_2W_2 + A_3W_3 \tag{1.2}$$

$$F(\overline{A},\ \overline{W}) = (3 \cdot 0.1) + (-5 \cdot - 1) + (0.1 \cdot 10) = 0.3 + 5 + 1 = 6.3$$

Нейронна мережа з одним входом та декількома виходами зображена на рисунку 1.7. Так як на вході лише одне число, то формула передбачення буде такою:

$$F(number,\ weight) = number \cdot weight \tag{1.3}$$

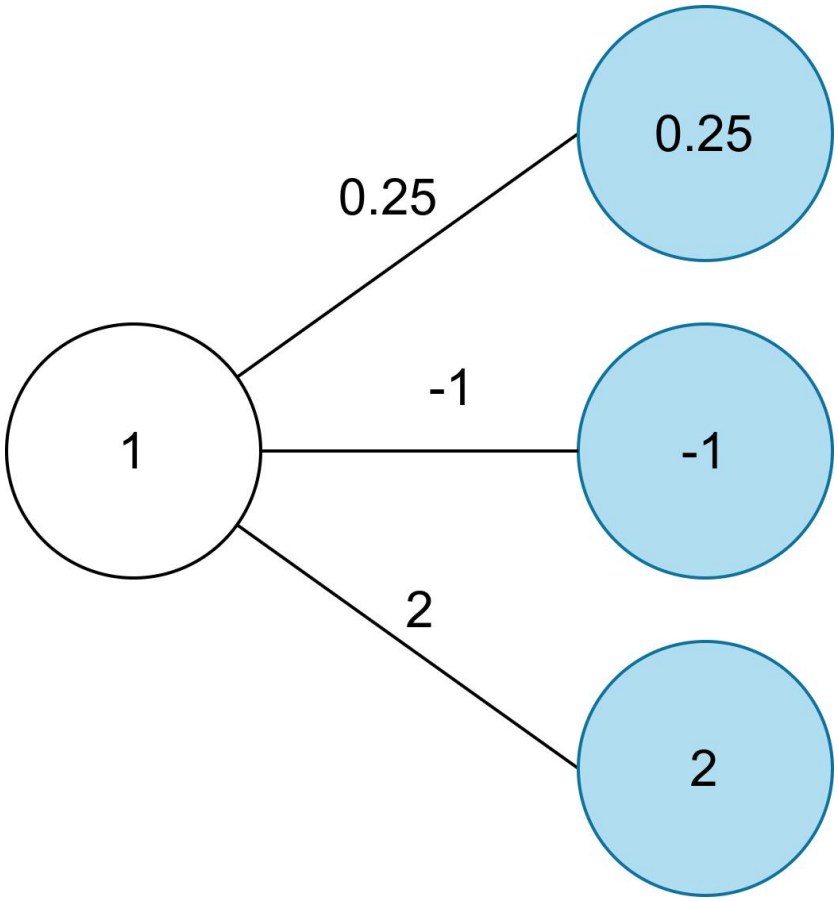

Рисунок 1.7 — Нейронна мережа з одним входом та трьома виходами

Математична формула трохи ускладнюється коли нейронна мережа має декільки входів і виходів. На рисунку 1.8 зображено таку мережу.

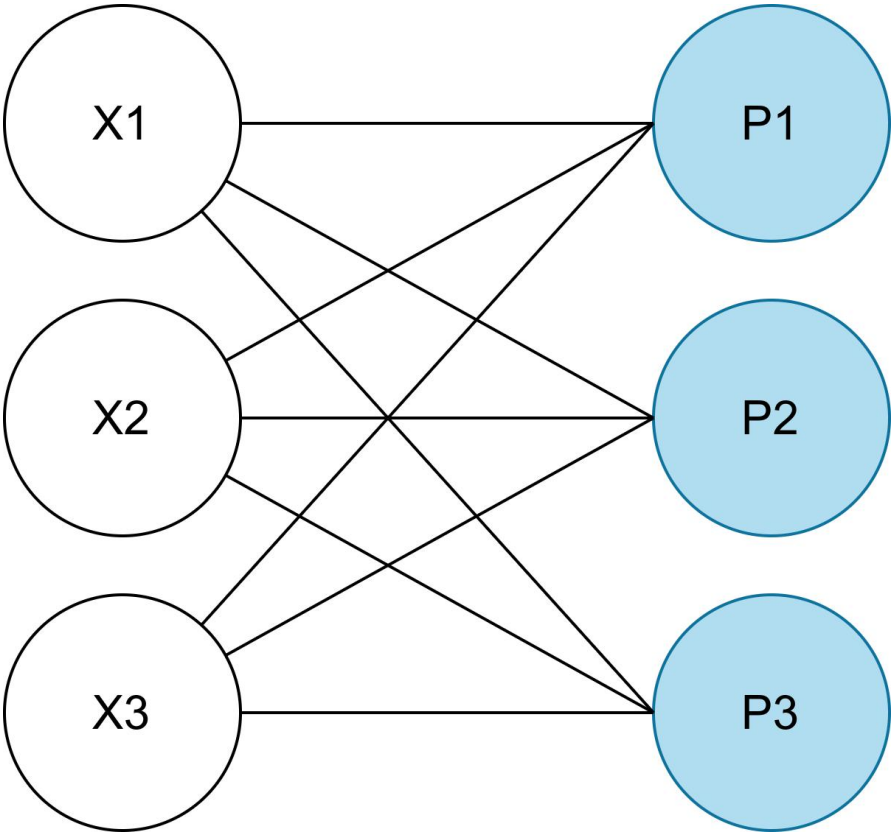

Рисунок 1.8 — Нейрона мережа з трьома входами та виходами

Дивлячись на цю мережу можна дійти висновку що вона складається з трьох менших мереж. Тобто, кожен вихідний нейрон це окрема нейронна мережа. Формула підрахунку передбачення, тепер приймає аргументи у вигляді вектора, для вхідних нейронів, і у вигляді матриці, для вагів мереж.

$$\begin{aligned} F(\overline{X}, \overline{W}) = \overline{X} \cdot \overline{W} &= |X_1 \quad X_2 \quad X_3| \cdot \begin{vmatrix} W_{X1,1} & W_{X2,1} & W_{X3,1} \\ W_{X1,2} & W_{X2,2} & W_{X3,2} \\ W_{X1,3} & W_{X2,3} & W_{X3,3} \end{vmatrix} \\ &= |X_1 * W_{X1,1} + X_2 * W_{X1,2} + X_3 * W_{X1,3} \quad \cdots \quad \cdots| \\ &= |P_1 \quad P_2 \quad P_3| \end{aligned} \tag{1.4}$$

Операція яка відбувається між матрицями, це звичайне множення матриць. Кожен елемент у вихідному векторі це передбачення отримане операцією — зважена сума.

Під час прямого розповсюдження попередня активація та активація відбувається на кожному прихованому рівні та вузлі вихідного рівня нейронної мережі.

Попередня активація – це зважена сума вхідних даних, яка є лінійним перетворенням ваг щодо доступних вхідних даних. Нейрон

приймає рішення про те, передавати інформацію далі чи ні, на основі суми та функції активації під час прямого поширення.

Активація – це обчислена зважена сума вхідних даних, переданих до функції активації. Функція активації – це математична функція, яка додає мережі нелінійність.

$$F(\overline{X}, \overline{W}) = \overline{X} \cdot \overline{W} = \ldots = A(|\cdots \quad \cdots \quad \cdots| = |A(P_1) \quad A(P_2) \quad A(P_3)| \qquad (1.5)$$

Де A(x) це якась функція від якої можна отримати першу похідну. Найбільш розповсюджена функція активації це relu [12, 13].

1.6 Шари нейронних мереж

Як видно з наступного рисунка, також можна взяти вихідні дані однієї мережі та передати їх як вхідні дані іншої мережі. Це призводить до двох послідовних векторно-матричних множень. Можливо, ще не зрозуміло, чому роблять такі моделі; але деякі набори даних (такі як класифікація зображень) містять кореляції, які є надто складними для одновагової матриці.

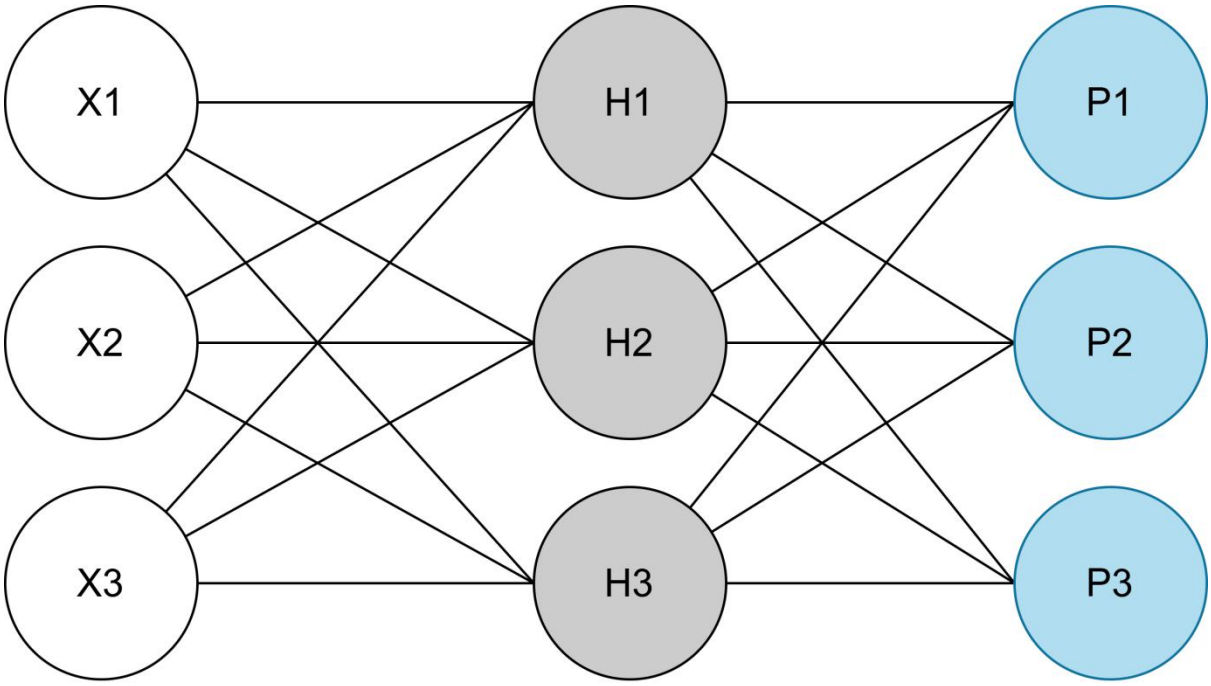

Рисунок 1.9 — Нейронна мережа з одним прихованим шаром

Зі збільшенням кількості шарів будуть з’являться нові проблеми. Існують методи для боротьби з цими побічними ефектами збільшення складності моделі, про них буде йтися в наступних розділах.

На рисунку 1.9 показано нейронну мережу в якій кожен нейрон зв’язаний з кожним нейроном із наступного шару. Існують інші види

архітектур, в яких зв’язаність не є повною або навіть деякі зв’язки формуються через один шар [14].

1.7 Градієнтний спуск

Для того щоб зрозуміти як можна “навчити” модель видавати правильний результат, треба розуміти який цикл проходить в нейронній мережі. Як відомо з минулих розділів, на вхід мережі подається якесь число, а потім воно множаться на відповідну вагу, після чого вихідне значення передається функції активації, це і є вихідним значенням одного нейрона.

Інша важлива частина градієнтного спуску це функція помилки або loss function. Вона вимірює на скільки передбачене значення відрізняється від правильної відповіді. Найпростіша функція помилки це різниця між правильним значенням та передбаченим в квадраті. Квадрат в такій формулі збільшує похибку коли вона і так велика, і зменшує її коли вона наближається до 0, це допомагає пришвидшити навчання.

Градієнтний спуск — це алгоритм оптимізації, який зазвичай використовується для навчання моделей машинного навчання та нейронних мереж. Навчальні дані допомагають цим моделям навчатися з часом, а функція помилки в градієнтному спуску діє як барометр, вимірюючи свою точність з кожною ітерацією оновлення параметрів. До тих пір, поки функція не буде близькою або дорівнювати нулю, модель буде продовжувати коригувати свої параметри, щоб дати найменшу можливу похибку[2].

Як було сказано раніше, функція помилки може виглядати як квадратична функція, але бувають і більш складні. На рисунку 1.10 показано графік квадратичної функцій помилки.

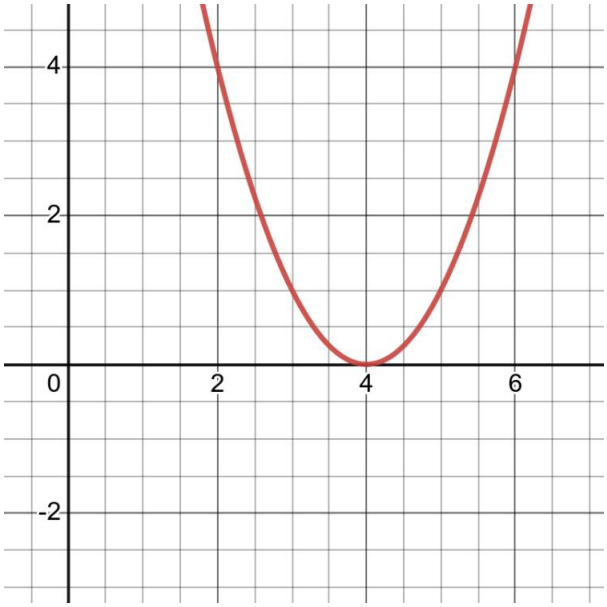

Рисунок 1.10 — Графік функції (x-4)^2

На квадратичній функції дуже леко показати як проходять кроки мінімізації. Так, на рисунку 1.11 показано графік тієї ж самої функції помилки що і раніше, але додаткова позначено стрибки або переходи на на епохах навчання моделі. З момента ініціалізації моделі, помилка буде десь з лівої або правої частини з великим значенням Y, а задача навчання знайти мінімум функції, в точці чотири нуль. Кожен стрибок це зазвичай одна епоха навчання яка наближає модель до точки мінімума.

Рисунок 1.11 — Графік функції (x-4)^2

Зі збільшенням швидкості навчання, або простіше кажучи зі збільшенням коефіцієнту зміни ваг можна пришвидшити находження мінімума, але це також може діяти у шкоду. Якшо один крок буде

настільки великим що проміжний мінімум попаде з однієї гілки графіка до іншої то це спричиняє ефект відстрибування від гілок графіка [15].

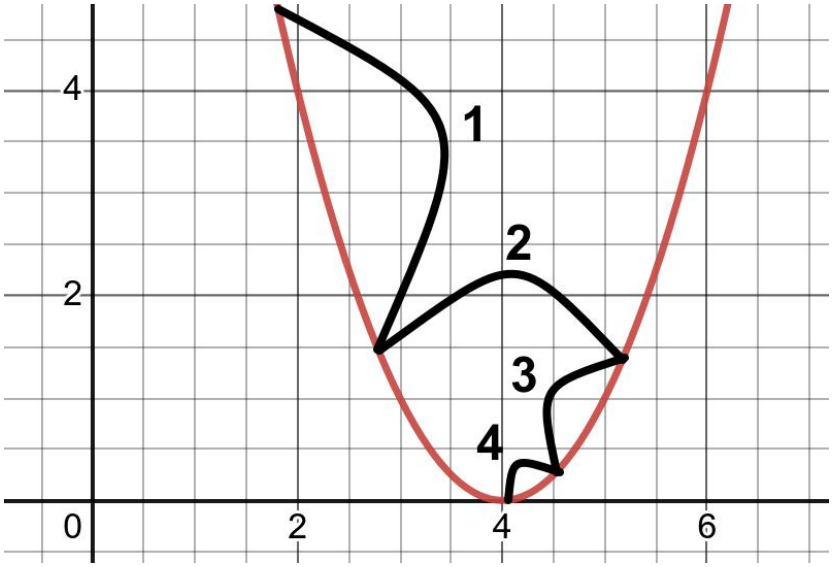

Рисунок 1.12 — Графік функції помилки з великим коефіцієнтом швидкості навчання

1.8 Що вивчають нейрони

Розглянемо реальний набір даних для тренування. Так як галузь машино навчання швидко розвивається і вже є одною з найпопулярніших галузей, то існує багато готових наборів даних для тренування своїх моделей. Один з таких наборів це MNIST.

Повна назва MNIST - це Modified National Institute of Standards and Technology і складається з цифр, які старшокласники та співробітники бюро перепису населення США написали від руки у 1998 році. Цікаво те, що ці рукописні цифри є чорно-білими зображеннями почерку людей. До зображення кожної цифри вказано фактичне число, яке воно означає (0–9). Протягом останніх кількох десятиліть люди використовували цей набір даних, щоб навчати нейронні мережі читати людський почерк.

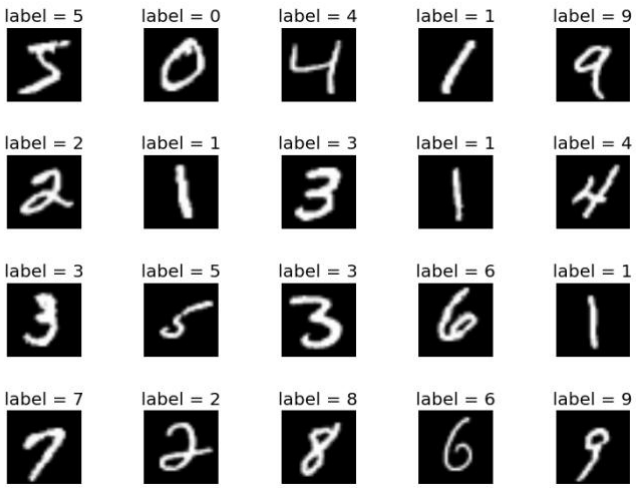

Рисунок 1.13 — Частина набору даних MNIST

Кожне зображення має лише 784 пікселя (28 × 28). Враховуючи, що одне зображення це 784 пікселя та 10 можливих станів, можна уявити форму нейронної мережі: кожен навчальний приклад містить 784 значення (по одному для кожного пікселя), тому нейронна мережа повинна мати 784 вхідних значення. Така модель повинна передбачити 10 ймовірностей: по одній для кожної цифри, як на рисунку 1.14.

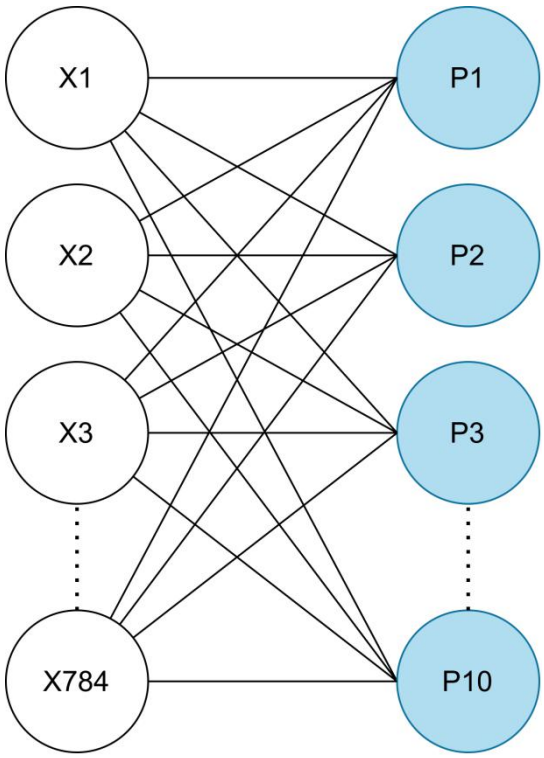

Рисунок 1.14 — Модель нейронної мережі для набору даних MNIST

Якщо натренувати модель з такою архітектурою як на рисунку 1.14, і візуалізувати ваги мережі, ми отримаємо щось схоже як на рисунку 1.15.

Мережа запам’ятовує специфічну форму цифри, а якщо написання цифри трохи відрізняється, то утворюється такий ефект розмиття, що дуже добре видно на цифрах два, п’ять та сім [16].

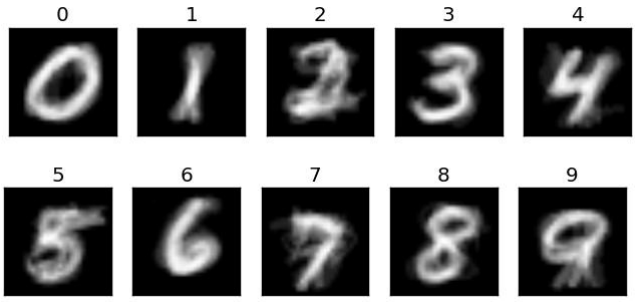

Рисунок 1.15 — Візуалізація матриці вагів моделі натренованої на MNIST набору даних з архітектурою один шар

1.9 Зворотнє розповсюдження

Зворотне розповсюдження полягає в передачі значення функції помилки назад таким чином, щоб ми могли точно налаштувати вагові коефіцієнти, на основі яких ця помилка і отримана. Функція оптимізації (градієнтний спуск) допоможе знайти вагові коефіцієнти, які приведуть до менших втрат у наступній ітерації навчання.

Якщо розповсюдження вперед відбувається з допомогою функції актвації та фунція гіпотези, яка в даному випадку була просто множенням ваг на входи нейрона. То для розповсюдження назад треба взяти часткові похідні цих функцій, тому при виборі функції активації в першу чергу звертають увагу на можливість взяти похідну.

Тепер потрібно знайти помилку в кожному нейроні нейронної мережі. Чому? Бо кожна помилка, до якої приходить модель глибокого навчання, насправді була сформована, тим, що всі вузли зібралися в одне число. Отже, потрібно з’ясувати, який вузол відповідає за більшу частину втрат на кожному шарі, щоб можна було надати йому менше значення ваги і таким чином зменшивши загальну помилку моделі [3, 17].

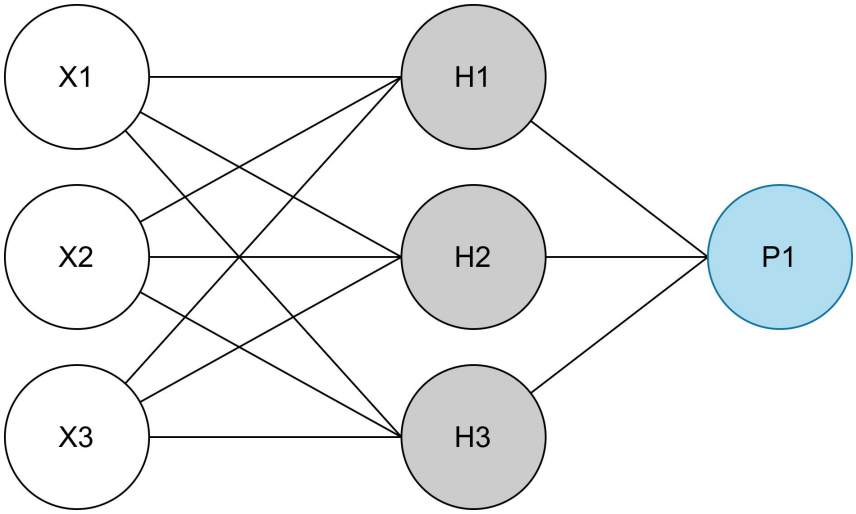

Рисунок 1.16 — Модель з одним виходом і трьома входами

1.10 Типи нейронних мереж

1.10.1 Згорткові нейронні мережі (комп'ютерна бачення)

Комп'ютерне бачення є першою та найбільшою історією успіху глибокого навчання. Щодня кожен із нас взаємодіє з моделями глибокого зору — через Google Photos, пошук зображень Google, YouTube, відеофільтри в програмах камери, програмне забезпечення OCR та багато іншого. Ці моделі також є центром передових досліджень автономного водіння, робототехніки, медичної діагностики за допомогою ШІ, автономних систем роздрібної торгівлі та навіть автономного землеробства.

Комп'ютерне бачення є областю, яка призвела до початкового зростання глибокого навчання між 2011 і 2015 роками. Тип моделі глибокого навчання, який називається згортковими нейронними мережами, почав отримувати надзвичайно хороші результати на змаганнях з класифікації зображень приблизно в той час, спочатку Ден Сіресан, який виграв дві ніші конкурси (конкурс з розпізнавання китайських ієрогліфів ICDAR 2011 та конкурс з розпізнавання дорожніх знаків у Німеччині IJCNN 2011), а потім, особливо, восени 2012 року, коли група Хінтона виграла гучний масштабний конкурс ImageNet з візуального розпізнавання[4, 5].

Цікаво, що цих ранніх успіхів було недостатньо, щоб зробити глибоке навчання мейнстрімом у той час — на це знадобилося кілька років. Спільнота дослідників комп'ютерного бачення витратила багато років, інвестуючи в методи, відмінні від нейронних мереж, і вона не була готова

відмовитися від них лише тому, що на сцені з’явився новачок. У 2013 і 2014 роках глибоке навчання все ще викликало сильний скептицизм багатьох дослідників комп’ютерного зору. Лише у 2016 році ГН нарешті стало домінуючим.

Фундаментальна відмінність між повноз'єднанними шарами і шаром згортки полягає в наступному: ПШ вивчають глобальні шаблони у своєму вхідному просторі ознак (наприклад, для цифри MNIST, шаблони, що включають всі пікселі), тоді як шари згортки вивчають локальні шаблони — у випадку зображень, візерунків, знайдених у маленьких двовимірних вікнах входів, див. рис. 1.17. Ця ключова характеристика надає ЗМ дві цікаві властивості.

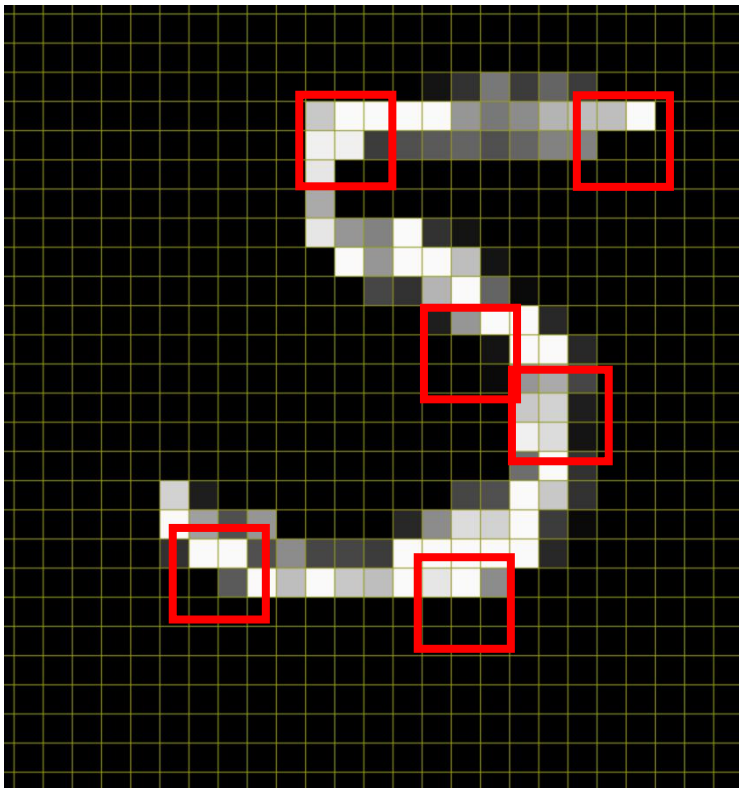

Рисунок 1.17 — Зображення можна розбити на локальні патерни з допомогою вікна згорки

Патерни, які вони вивчають, є інваріантними до переміщення. Вивчивши певний шаблон у нижньому правому куті картинки, ЗМ може розпізнати його будь-де: наприклад, у верхньому лівому куті. Повноз'єднана модель повинна була б вивчати шаблон заново, якби вона з’явилася в новому місці. Це робить ЗМ ефективними щодо даних під час обробки зображень (оскільки візуальний світ фундаментально інваріантний щодо переміщення): їм потрібно менше навчальних зразків, щоб вивчати представлення, які мають силу узагальнення.

Вони можуть вивчати просторову ієрархію патернів. Перший шар згортки вивчатиме невеликі локальні шаблони, такі як ребра фігур, другий шар згортки вивчатиме більші шаблони, створені з особливостей перших шарів, і так далі див. рис. 1.18. Це дозволяє мережам ефективно вивчати дедалі складніші та абстрактніші візуальні поняття, оскільки візуальний світ принципово просторово ієрархічний.

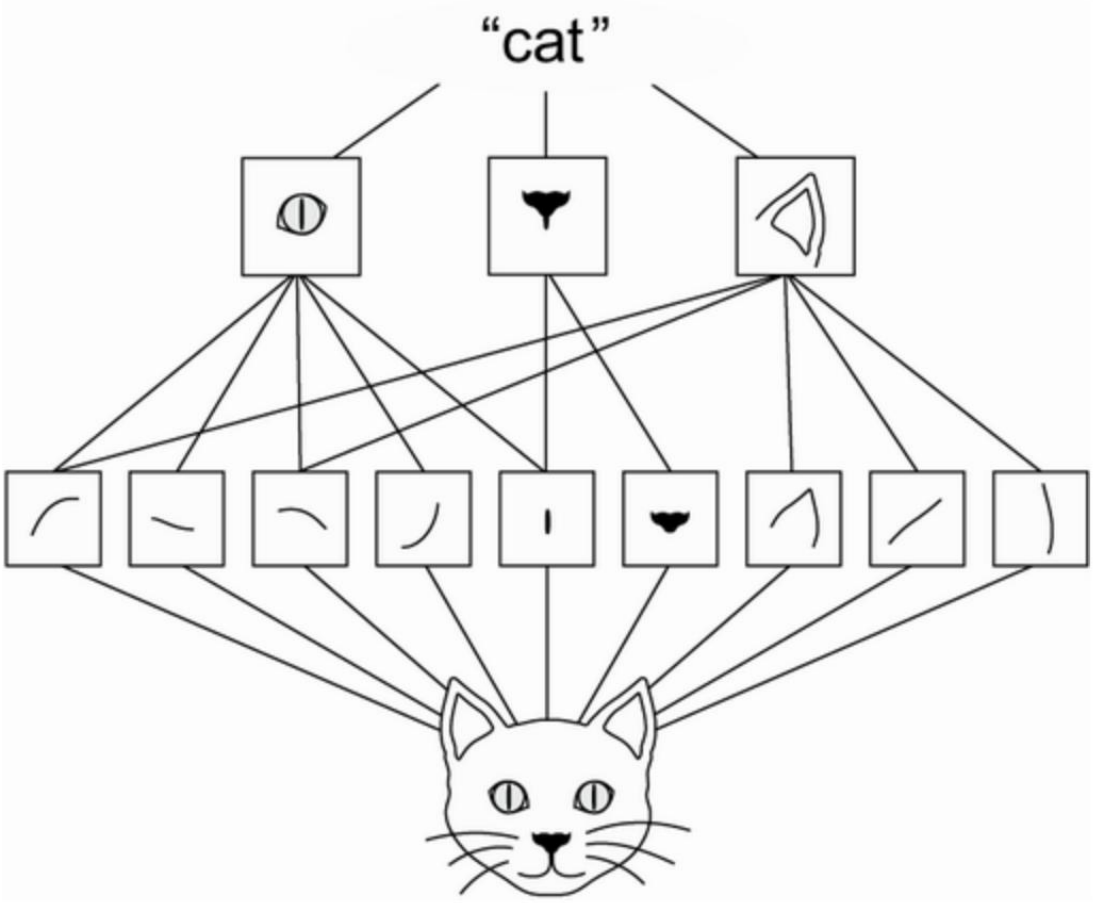

Рисунок 1.18 — Виділення локальних особливостей

ЗМ працюють над тензорами рангу 3, які називаються мапами ознак, з двома просторовими осями (висота та ширина), а також віссю глибини (також називається віссю каналів). Для зображення RGB розмір осі глибини дорівнює 3, оскільки зображення має три кольорові канали: червоний, зелений і синій. Для чорно-білого зображення, як і для цифр MNIST, глибина дорівнює 1 (рівні сірого). Операція згортки витягує вікна заданого розміру з мапи ознак і застосовує те саме перетворення до всіх осей, створюючи вихідну карту функцій. Ця вихідна мапа ознак все ще є тензором рангу 3, вона має ширину та висоту. Її глибина може бути довільною, оскільки вихідна глибина є параметром шару, а різні канали на

цій осі глибини більше не позначають певні кольори, як у вхідних даних RGB; скоріше, вони означають фільтри. Фільтри кодують певні аспекти вхідних даних, наприклад, на високому рівні один фільтр може кодувати концепцію «присутності обличчя у вхідних даних»[6].

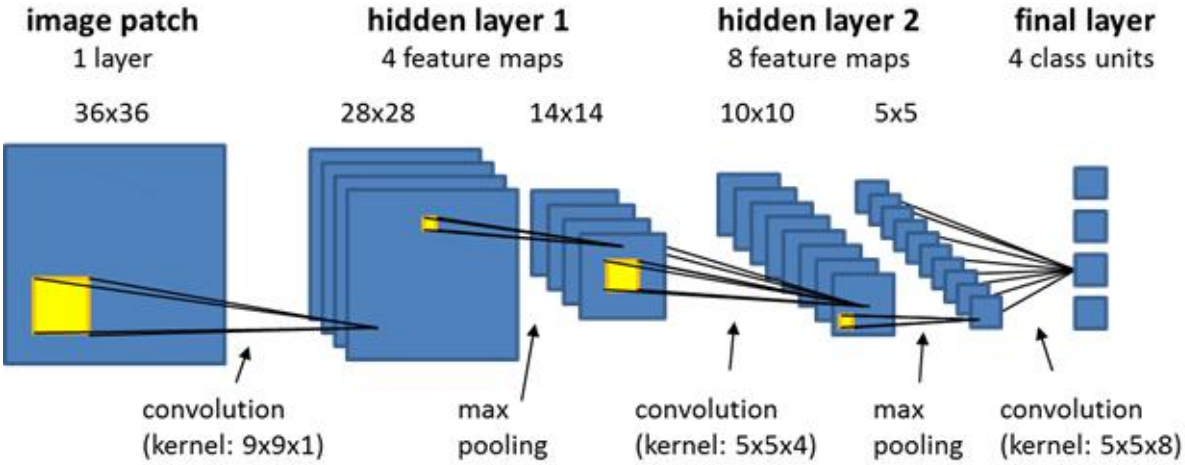

Рисунок 1.19 — Зміна даних у згорткових нейроних мережах

1.10.1.1 Ефекти межі та відступи

Розглянемо карту об’єктів 5 × 5 (загалом 25 клітинок). Є лише 9 клітинок, навколо яких можна центрувати вікно 3 × 3, утворюючи сітку 3 × 3, див. рисунок 1.20. Отже, вихідна мапа об’єктів матиме 3 × 3. Вона трохи зменшується: у цьому випадку рівно на дві клітинки поряд з кожним виміром.

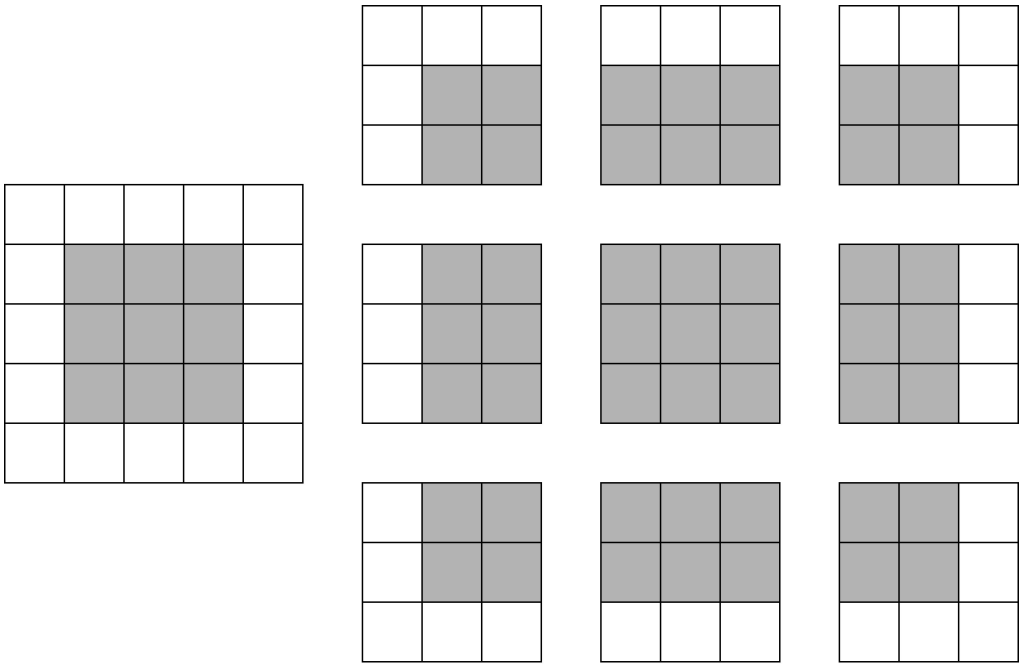

Рисунок 1.20 — Ефект межі

Якщо треба отримати вихідну мапу об’єктів з тими самими просторовими розмірами, що й вхідні дані, то треба використовувати відступи. Додавання відступів це додавання відповідної кількості рядків і стовпців з кожного боку мапи вхідних об’єктів, щоб зробити можливим розміщення центральних вікон згортки навколо кожної вхідної клітинки [19, 20]. Для вікна 3 × 3 додається один стовпець праворуч, один стовпець ліворуч, один рядок угорі та один рядок унизу. Для вікна 5 × 5 додається два ряди див. рисунок 1.21.

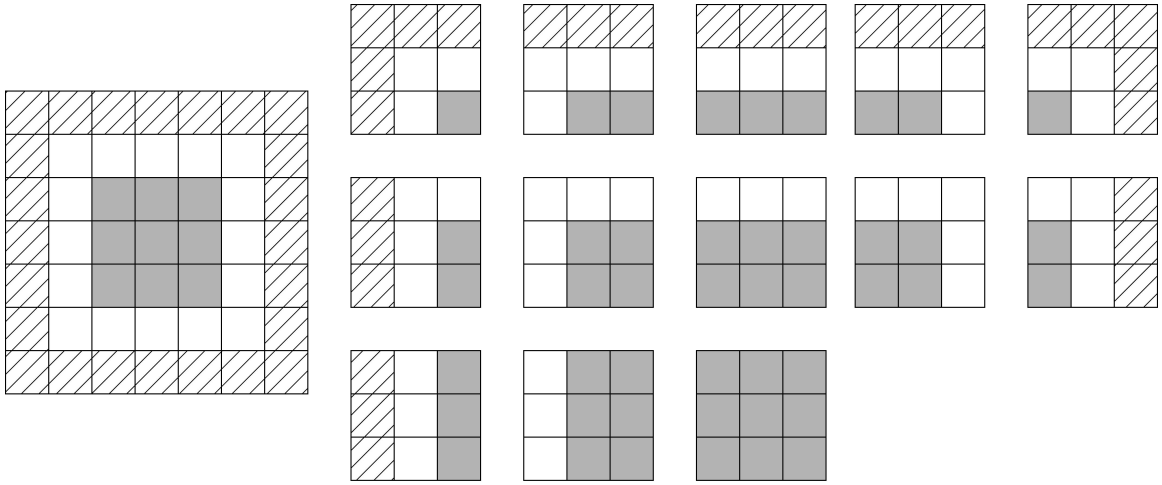

Рисунок 1.21 — Відступи

Іншим фактором, який може вплинути на розмір вихідних даних, є поняття кроків. Відстань між двома послідовними вікнами є параметром згортки, який називається її кроком, який за замовчуванням дорівнює 1. Можливі поступові згортки: згортки з кроком, вищим за один. На рисунку 1.22 можна побачити вікна згортки 3 × 3 із кроком 2 з вхідними даними 5 × 5.

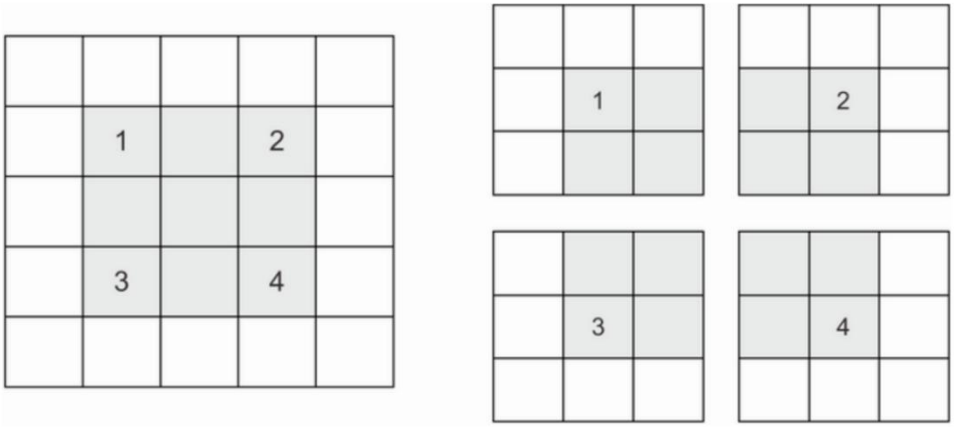

Рисунок 1.22 — Крок у дві клітинки між ітераціями

1.10.1.2 Аугментація даних

Перенавчання спричиняється надто малою кількістю даних для навчання, через що дуже складно або інколи неможливо навчити модель, яка може узагальнювати нові дані. Якщо б для навчання була доступна нескінчена кількість даних, модель би мала змогу побачити всі можливі аспекти наявні в даних, тобто вона ніколи б не перенавчилася. Аугментація даних використовує підхід створення більшої кількості навчальних даних із наявних навчальних даних шляхом доповнення зразків за допомогою ряду випадкових перетворень, які дають правдоподібні зображення. Мета полягає в тому, щоб під час навчання модель ніколи не побачила однакове зображення двічі. Це допомагає надати моделі більше кількості аспектів даних, щоб вона могла краще узагальнювати.

Серед найрозповсюджених функцій для аугментації даних є випадкове відображення; випадковий переворот на декілька градусів; випадкове приближення зображення, див рисунок 1.23.

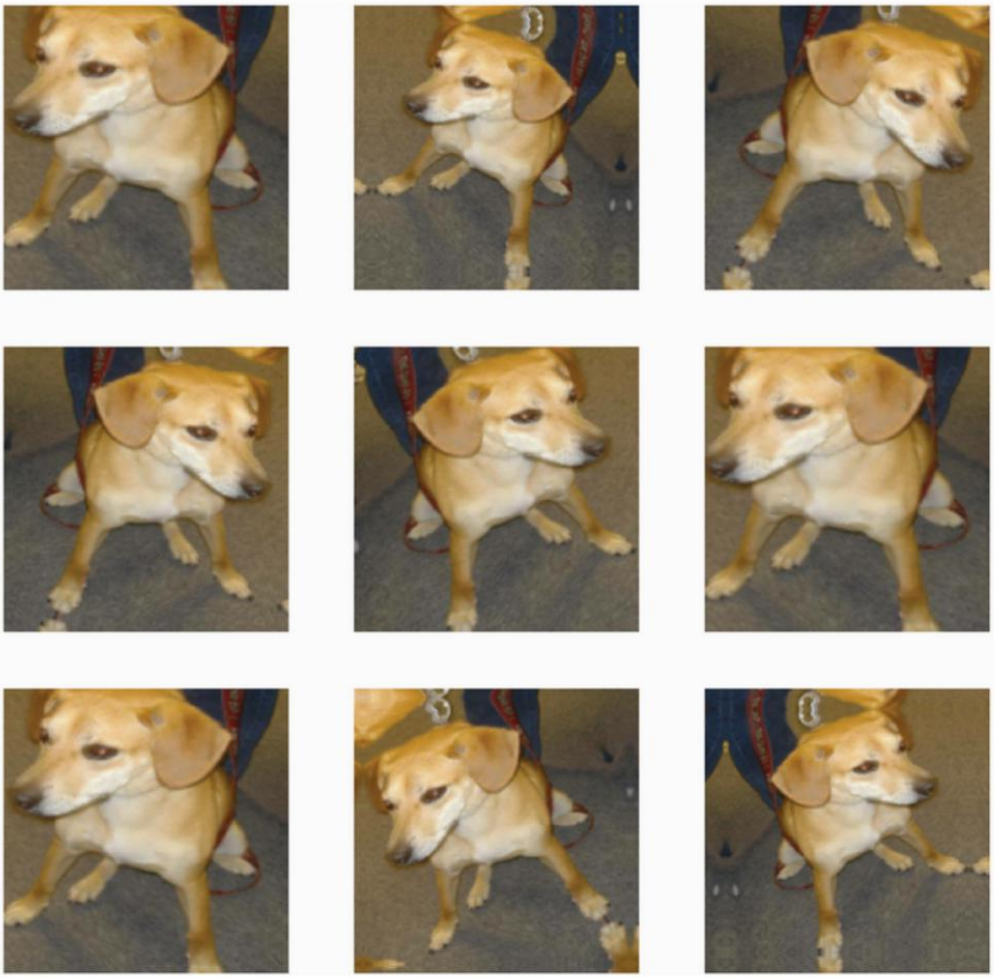

Рисунок 1.23 — Випадкові перетворення зображення, аугментація

1.10.2 Рекуренті нейронні мережі

Рекурентна нейронна мережа (RNN) — це тип штучної нейронної мережі, яка використовує послідовні дані або дані часових рядів. Ці алгоритми глибокого навчання зазвичай використовуються для порядкових або часових проблем, таких як переклад мови, обробка природної мови, розпізнавання мовлення та підписи до зображень; вони включені в такі популярні програми, як Siri, голосовий пошук і Google Translate. РНМ відрізняються своєю «пам'яттю», оскільки вони беруть інформацію з попередніх вхідних даних, щоб впливати на поточні вхідні та вихідні дані. Хоча традиційні глибокі нейронні мережі припускають, що входи та виходи незалежні один від одного, вихід рекурентних нейронних мереж залежить від попередніх елементів у послідовності. Хоча майбутні події також були б корисними для визначення результату даної послідовності, односпрямовані рекурентні нейронні мережі не можуть врахувати ці події у своїх прогнозах.

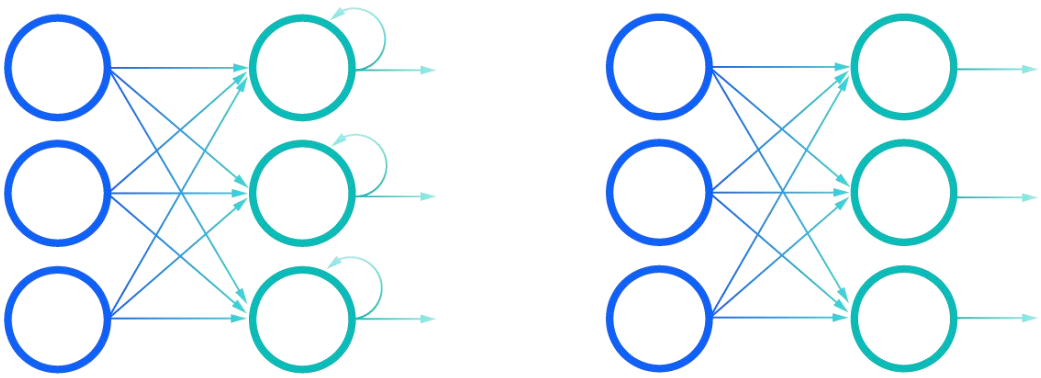

Рисунок 1.24 — Рекурента модель зліва та звичайна модель зправа

Часовий ряд може являти собою будь-які дані, отримані за допомогою вимірювань через регулярні проміжки часу, як-от щоденна ціна акцій, погодинне споживання електроенергії в місті або тижневі продажі в магазині. Часові ряди присутні скрізь, незалежно від того, чи це природні явища (наприклад, сейсмічну активність, еволюцію популяцій риб у річці чи погоду в певному місці) чи моделі людської діяльності (наприклад, відвідувачі веб-сайту, ВВП країни чи кредит операції з карткою). На відміну від типів даних, про які розповідалося вище, робота з часовими рядами передбачає розуміння динаміки системи — її періодичних циклів, її тенденції з часом, її регулярний режим і раптові сплески.

Безумовно, найпоширенішим завданням, пов’язаним із часовими рядами, є прогнозування, тобто передбачення того, що станеться далі в серії. Прогноз споживання електроенергії на кілька годин наперед; прогнозування доходу на кілька місяців вперед, для планування бюджету; прогнозування погоди на кілька днів наперед, для планування графіку.

Класифікація часових рядів — призначає одну або кілька категорійних міток для часового ряду. Наприклад, враховуючи часовий ряд активності відвідувача на веб-сайті, класифікує чи є відвідувач ботом чи людиною.

Виявлення подій часових рядів — визначає виникнення певної очікуваної події в безперервному потоці даних. Особливо корисним додатком є «виявлення гарячих слів», коли модель відстежує аудіопотік і розпізнає такі висловлювання, як «Окей Google» або «Агов Alexa».

Виявлення аномалій в часових рядах — виявлення будь-яких незвичних подій у безперервному потоці даних. Незвичайна діяльність у корпоративній мережі; незвичайні показання на виробничій лінії. Виявлення аномалій зазвичай виконується за допомогою навчання без учителя, оскільки часто не можна знати, яку аномалію треба шукати, тому неможливо тренуватися на конкретних прикладах аномалій.

Повторювані нейронні мережі використовують алгоритм зворотного поширення в часі (BPTT) для визначення градієнтів, який дещо відрізняється від традиційного зворотного поширення, оскільки він специфічний для даних послідовності. Принципи BPTT такі ж, як і традиційного зворотного поширення, коли модель навчається, обчислюючи помилки від вихідного до вхідного рівня. Ці розрахунки дозволяють правильно налаштувати та підігнати параметри моделі. BPTT відрізняється від традиційного підходу тим, що він підсумовує помилки на кожному кроці часу, тоді як у звичайних мережах не потрібно підсумовувати помилки, оскільки вони не поділяють параметри на кожному рівні.

Через цей процес RNN, як правило, стикаються з двома проблемами, відомими як вибухові градієнти та зникнення градієнтів. Ці проблеми визначаються розміром градієнта, який є нахилом функції втрат уздовж кривої помилок. Коли градієнт занадто малий, він продовжує зменшуватися, оновлюючи вагові параметри, поки вони не стануть незначними, тобто. 0. Коли це відбувається, алгоритм більше не навчається. Вибухові градієнти виникають, коли градієнт занадто великий, що створює нестабільну модель. У цьому випадку ваги моделі виростуть занадто великими, і в кінцевому підсумку вони будуть представлені як

NaN. Одним із рішень цих проблем є зменшення кількості прихованих шарів у нейронній мережі, усунення частини складності в моделі RNN.

1.11 Метрики машинного навчання

Вибір правильної метрики має вирішальне значення під час оцінки моделей машинного навчання. Для різних областей задач використовують різні метрики.

Також варто згадати, що метрика відрізняється від функції помилки. Функції помилки — це функції, які показують міру продуктивності моделі та використовуються для навчання моделі машинного навчання (з використанням певної оптимізації), і зазвичай диференціюються за параметрами моделі. З іншого боку, метрики використовуються для моніторингу та вимірювання ефективності моделі (під час навчання та тестування), і не повинні бути диференційованими. Однак, якщо для деяких завдань метрика продуктивності є диференційованою, її можна використовувати і як функцію втрат (можливо, з деякими регуляризаціями, доданими до неї), і як метрику, таку як MSE.

1.11.1 Матриця невідповідності

Це не метрика, але дуже важливо розуміти як інтерпретувати цю матрицю.

Однією з ключових концепцій продуктивності класифікації є матриця невідповідності (так відома як матриця помилок), яка є табличною візуалізацією прогнозів моделі порівняно з мітками базової істинності. Кожен рядок матриці невідповідності представляє екземпляри в передбаченому класі, а кожен стовпець представляє екземпляри в фактичному класі.

Припустімо, що треба збудувати бінарну класифікацію, щоб класифікувати зображення котів від зображень, які не є котами. І припустімо, що тестовий набір містить 1100 зображень (1000 зображень не котів і 100 зображень котів) із наведеною нижче матрицею плутанини.

Таблиця 1.1 - Матриця невідповідності

	Фактичний клас	
	Кіт	Не кіт

Передбачений клас	Кіт	90	60
	Не кіт	10	940

Зі 100 зображень котів модель правильно передбачила 90 і неправильно класифікувала 10. Якщо назвати клас «кота» позитивним, а клас не-кота – негативним, тоді 90 зразків, передбачуваних як коти, вважаються істинно-позитивними, а 10 зразків, передбачуваних як не-кішки, є хибно негативними.

З 1000 зображень, які не стосуються котів, модель класифікувала 940 правильно, а 60 неправильно. 940 правильно класифікованих зразків називають істинно негативними, а ті 60 називають хибно позитивними.

Тепер можна побачити що діагональні елементи цієї матриці позначають правильний прогноз для різних класів, тоді як недіагональні елементи позначають зразки, які неправильно класифіковані.

1.11.2 Точність

Точність класифікації є, мабуть, найпростішим показником, який можна собі уявити, і визначається як кількість правильних прогнозів, поділена на загальну кількість прогнозів, помножену на 100. Отже, у наведеному вище прикладі з 1100 зразків 1030 прогнозовано правильно, в результаті чого точність класифікації:

$$\mathrm{A} = \frac{\sum_i P_T}{\sum_i P_F} \tag{1.6}$$

$$\frac{90 + 940}{1000 + 100} * 100\% = 93.6\%$$

1.11.3 Влучність

Існує багато випадків, коли точність класифікації не є хорошим показником ефективності моделі. Одним із таких сценаріїв є незбалансований розподіл класів (один клас зустрічається частіше за інші). У цьому випадку, навіть якщо передбачити всі зразки як найчастіший клас, то модель отримаєте високий рівень точності, що взагалі не має сенсу (оскільки модель нічого не вивчає, а лише передбачає все як найвищий клас). Наприклад, у наведеній вище класифікації котів і не котів, якщо

модель передбачає, що всі зразки не є котами, це призведе до 1000/1100= 90,9%.

$$\frac{1000}{1100} * 100\% = 90.9\%$$

Тому також потрібно дивитися на показники продуктивності для конкретного класу. Влучність є одним із таких показників, який визначається як:

$$\text{Влучність} = \frac{\text{ІП}}{\text{ІП} + \text{ХП}} \tag{1.7}$$

Точність класів котів і не котів у прикладі вище можна обчислити як:

$$\text{Влучність котів} = \frac{90}{90 + 60} * 100\% = 60\%$$

$$\text{Влучність не котів} = \frac{940}{950} * 100\% = 98.9\%$$

Як ми бачимо, модель має набагато вищу точність у прогнозуванні не котячих зразків порівняно з котами. Це не дивно, оскільки під час навчання модель бачила більше прикладів зображень без котів, завдяки чому вона краще класифікувала цей клас.

1.11.4 Повнота

Повнота — ще один важливий показник, який визначається як частка зразків із класу, які правильно передбачені моделлю. Більш формально:

$$\text{Повнота} = \frac{\text{ІП}}{\text{ІП} + \text{ХН}} \tag{1.8}$$

Повноту класів котів і не котів у прикладі вище можна обчислити як:

$$\text{Повнота котів} = \frac{90}{100} * 100\% = 90\%$$

$$\text{Повнота не котів} = \frac{940}{1000} * 100\% = 94\%$$

1.11.5 F1 міра

Залежно від цілей, можна надати вищий пріоритет влучності чи повноті. Але є багато застосувань, у яких важливі як влучність, так і повнота. Тому придумали спосіб об’єднати ці два показники в один показник. Одна популярна метрика, яка поєднує точність і

запам’ятовування, називається F1 міра, що є гармонійним середнім значенням влучності та повноти, яке визначається як:

$$\mathrm{F}_1 = 2 \cdot \frac{\text{влучність} \cdot \text{повнота}}{\text{влучність} + \text{повнота}} \tag{1.9}$$

або для загального випадку

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{влучність} \cdot \text{повнота}}{\beta^2 \cdot \text{влучність} + \text{повнота}} \tag{1.10}$$

Варто зазначити, що завжди існує компроміс між влучністю та повнотою моделі. Якщо треба зробити влучність занадто високою, то в кінцевому підсумку рівень повноти впаде, і навпаки

1.11.6 ROC-крива

ROC-крива або робоча характеристика приймача є графіком, який показує продуктивність двійкового класифікатора як функцію його порогового значення. По суті, він показує істинну позитивну частоту (TPR) проти помилкової позитивної частоти (FPR) для різних порогових значень.

Багато моделей класифікації є імовірнісними, тобто вони передбачають ймовірність того, що вибірка є міткою. Потім вони порівнюють цю ймовірність виходу з деяким пороговим значенням і, якщо воно перевищує порогове значення, вони прогнозують його мітку як 0, інакше як 1. Як приклад, модель може передбачити такі ймовірності для 4 зразків зображень: [0,45, 0,6, 0,7, 0,3]. Тоді залежно від порогового значення виходять різні мітки:

Таблиця 1.2 — Зміна міток від порогового значення

Порог	Передбачення
0.5	[0,1,1,0]
0.2	[1,1,1,1]
0.8	[0,0,0,0]

Як видно з таблиці, змінюючи порогові значення, виходять абсолютно різні мітки. Кожен із цих сценаріїв призведе до різних показників влучності та повноти.

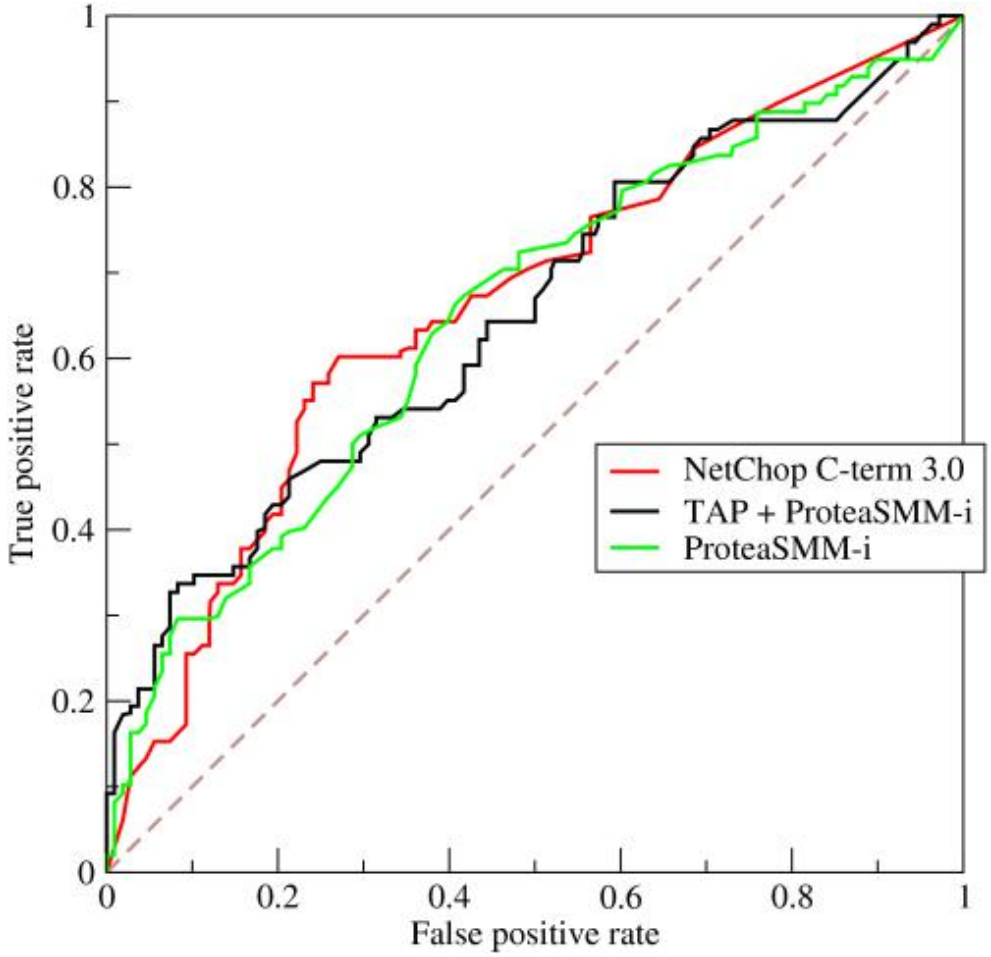

Рисунок 1.25 — ROC крива трьох методів

Чим нижчий поріг відсікання для позитивного класу, тим більше зразків прогнозується як позитивний клас, тобто вищий істинно позитивний рівень, а також вищий хибно позитивний рівень (що відповідає правій частині цієї кривої). Тому існує компроміс між тим, наскільки високим може бути влучність, і тим, наскільки треба обмежити помилку.

Крива ROC — це популярна крива, яка дає змогу оцінити загальну продуктивність моделі та вибрати порогове значення для моделі.

1.11.7 AUC або площа під ROC-кривою

Площа під кривою (AUC) є агрегованою мірою ефективності бінарного класифікатора для всіх можливих порогових значень (і тому вона є інваріантною до порогу).

AUC обчислює площу під кривою ROC, тому вона знаходиться між 0 і 1. Один із способів інтерпретації AUC – це ймовірність того, що модель

оцінює випадковий позитивний приклад вище, ніж випадковий негативний приклад.

Щоб вирішити, як оцінити ефективність моделі класифікації, треба добре зрозуміти вимоги до бізнесу/проблеми та вплив низької влучності та низької повноти, а також вирішити, за яким показником оптимізувати.

З практичної точки зору, модель класифікації, яка виводить ймовірності, є кращою, ніж вихід з одною міткою, оскільки вона забезпечує гнучкість налаштування порогу таким чином, щоб він відповідав мінімальним вимогам до влучності/повноти. Однак не всі моделі забезпечують такі хороші ймовірнісні результати.

1.11.8 СКП

Середня квадратична помилка є, мабуть, найпопулярнішим показником, який використовується для задач регресії. Він знаходить середню квадратичну помилку між прогнозованим і фактичним значеннями.

Припустімо, що є регресійна модель, яка передбачає ціни на будинки в районі Києва ($\hat{y}_i$), і припустімо, що для кожного будинку також є фактична ціна, за яку будинок було продано (позначено як y_i). Тоді СКП можна розрахувати як:

$$\text{СКП} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{1.11}$$

Іноді використовується КСКП, щоб отримати метрику з масштабом як цільові значення, що є є квадратним коренем із СКП.

З прикладу про прогноз цін на житло, КСКП показує, яке середнє відхилення в прогнозованих моделлю цін на житло від цільових значень (ціни, за якою будинки продаються).

1.11.9 САП

Середня абсолютна похибка (або середнє абсолютне відхилення) є ще одним показником, який визначає середню абсолютну відстань між прогнозованим і цільовим значеннями. САП визначається так:

$$\text{САП} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|^2 \tag{1.12}$$

САП більш стійкий до відхилень, ніж СКП. Основна причина полягає в тому, що в САП шляхом зведення помилок у квадрат, відхилення (які зазвичай мають вищі похибки, ніж інші зразки) отримують більше уваги та домінування в остаточній помилці та впливають на параметри моделі [22].

1.12 Висновки за розділом

Комп’ютерне навчання працює за суворими математичними законами, тому не треба плутати це, коли інколи у медіа кажуть що це вирішення усіх проблем людства. Нейроні мережі так само не зможуть досягти само усвідомлення без кардинальних змін у алгоритмах та продуктивності обладнання для навчання або запуску таких моделей.

При тренуванні слід завжди використовувати відповідні метрики продуктивності моделі, під конкретну задачу. Завжди є компроміси на які доведеться йти, тому треба свідомо обирати що важливіше.

2 Сучасні технології розробки глибоких мереж

2.1 Мова програмування з підтримкою нейронних мереж

Коли ми кажемо підтримка нейронних мереж то треба розуміти що майже будь який математичний алгоритм можна реалізувати на будь-якій мові програмування. Підтримка побудови нейронних мереж в даному випадку означає просте створення моделей та їх відлагодження; вбудовані швидкі алгоритми або їх акселерація за допомогою спеціалізованого обладнання так і обладнання звичайних споживачів, таких як відеокарти.

2.2 Мова програмування Python

Якість програмного забезпечення є важливою складовою успіху в промисловості та науці. IT-системи контролюють бізнес-процеси глобальної економіки. Все більш потужні та складні комп'ютери алгоритми забезпечують платформу для нових наукових відкриттів. А глобальне спілкування немислиме без інтелектуального програмного забезпечення.

У боротьбі за покупців головне місце займають ті, хто виходить на ринок швидше за своїх конкурентів. Кращі та креативніші рішення в поєднанні з можливістю миттєво реагувати на нове виклики керують перегонами. Написання безпечних і надійних програм за короткий проміжок часу, дозволить першим досягнути успіху.

Комплексна стандартна бібліотека та тисячі додаткових пакетів Python надають розробникам високоякісні рішення, які вони можуть легко інтегрувати у свої програми для задоволення практично будь-яких вимог. Таким чином Python звільняє величезні ресурси, які можна виділити для більш продуктивного використання в іншому місці[7].

Якщо перелічувати плюси та мінуси Python порівнянно з іншими мовами, то серед плюсів є:

+ швидкість розробки;
+ легкий та зрозумілий синтаксис;
+ велика екосистема сторонніх пакетів;
+ доступний майже на всіх пристроях.

А серед мінусів:

- низька швидкодія Python коду, що обходять використанням C/C++/Rust;
- складно писати багатопоточні додатки;
- стара мова, але вбирає в себе новітні просування у сфері мов програмування.

2.3 Anaconda

Anaconda — це дистрибутив пакетів, створених для Data Science. Він поставляється з conda, менеджером середовища. Зазвичай conda використовується для ізоляції проектів, які використовують різні версії Python та/або різні версії пакетів. Також conda вміє встановлювати, видаляти та оновлювати пакети у середовищах проектів. Anaconda поставляється з conda, Python і понад 150 науковими пакетами та їхніми залежностями. Це досить велике завантаження (~500 МБ), оскільки воно постачається з найпоширенішими пакетами наукових бібліотек на Python. Для людей, які консервативно ставляться до дискового простору, є також Miniconda, менший дистрибутив, який включає лише conda та Python.

2.4 Бібліотека Tensorflow

TensorFlow — це платформа все в одному з відкритим кодом для машинного навчання. Вона має комплексну, гнучку екосистему інструментів, бібліотек і ресурсів спільноти, що дозволяє дослідникам просувати найсучасніші технології машинного навчання, а розробникам – легко створювати й розгортати програми на базі машинного навчання.

Спочатку TensorFlow був розроблений дослідниками та інженерами, які працюють у команді Google Brain в організації Google Machine Intelligence Research для проведення машинного навчання та досліджень нейронних мереж. Система є достатньо загальною, щоб її також можна було застосовувати в багатьох інших областях.

TensorFlow забезпечує стабільні API Python і C++, а також API для інших мов але без зворотньої сумісності[8].

Можливості Tensorflow:

- він може автоматично обчислювати градієнт будь-якого диференційованого виразу, що робить його придатним для машинного навчання;
- він може працювати не тільки на процесорах, а й на графічних і TPU, високопаралельних апаратних прискорювачах;

- обчислення, визначені в TensorFlow, можна легко розподілити між багатьма машинами;
- програми TensorFlow можна експортувати в інші середовища виконання, такі як C++, JavaScript (для додатків на основі браузера) або TensorFlow Lite (для додатків, що працюють на мобільних пристроях або вбудованих пристроях) тощо. Це спрощує розгортання додатків TensorFlow на практиці.

2.5 Бібліотека Keras

Keras — це API глибокого навчання для Python, створений на основі TensorFlow, який забезпечує зручний спосіб визначення та навчання будь-якої моделі глибокого навчання. Keras спочатку був розроблений для досліджень з метою забезпечення швидких експериментів із глибоким навчанням.

Завдяки TensorFlow Keras може працювати на різних типах апаратного забезпечення — GPU, TPU або звичайному CPU — і його можна плавно масштабувати до тисяч машин.

Keras відомий тим, що надає перевагу якості розробки. Це API для людей, а не для машин. Він дотримується найкращих практик для зменшення когнітивного навантаження: пропонує узгоджені та прості робочі процеси, мінімізує кількість дій, необхідних для типових випадків використання, і забезпечує чіткий і дієвий зворотний зв’язок у разі помилки користувача. Це робить Keras простим у вивченні для початківців і дуже продуктивним у використанні для експертів.

Станом на кінець 2021 року Keras має більше мільйона користувачів, починаючи від академічних дослідників, інженерів і дослідників даних у стартапах і великих компаніях до аспірантів і любителів. Keras використовується в Google, Netflix, Uber, CERN, NASA, Yelp, Instacart, Square і сотнях стартапів, які працюють над широким спектром проблем у кожній галузі. Рекомендації YouTube походять від моделей Keras. Безпілотні автомобілі Waymo розроблені з моделями Keras. Keras також є популярним фреймворком на Kaggle, веб-сайті змагань з машинного навчання, де більшість змагань із глибокого навчання вигравали за допомогою Keras.

Оскільки Keras має велику та різноманітну базу користувачів, він не змушує користувача дотримуватися єдиного «справжнього» способу створення та навчання моделей. Навпаки, він забезпечує широкий спектр різних робочих процесів, від дуже високого до дуже низького рівня, що

відповідає різним профілям користувачів. Наприклад, є безліч способів створення моделей і безліч способів їх навчання, кожен з яких представляє певний компроміс між зручністю використання та гнучкістю. Ви можете використовувати Keras так само, як Scikit-learn — просто викликати fit() і дозволити фреймворку робити свою справу — або ви можете використовувати його як NumPy — повністю контролювати кожен аспект.

Ця філософія не відрізняється від філософії самого Python. Деякі мови пропонують лише один спосіб написання програм — наприклад, об'єктно-орієнтоване програмування або функціональне програмування. Водночас Python є мультипарадигмальною мовою: вона пропонує низку можливих шаблонів використання, які добре працюють разом. Це робить Python придатним для широкого діапазону дуже різних випадків використання: системне адміністрування, наука про дані, машинне навчання, веб-розробка. Подібним чином можна думати про Keras як про Python глибокого навчання: зручну мову глибокого навчання, яка пропонує різноманітні робочі процеси для різних профілів користувачів.

2.6 Історія Tensorflow і Keras

Keras випередив TensorFlow на вісім місяців. Він був випущений у березні 2015 року, а TensorFlow — у листопаді 2015 року. Виникає питання, якщо Keras побудований на основі TensorFlow, як він міг існувати до випуску TensorFlow? Keras спочатку був побудований на основі Theano, іншої бібліотеки для обробки тензорів, яка забезпечувала автоматичне диференціювання та підтримку графічного процесора — найперша у своєму роді. Theano, розроблений Монреальським інститутом навчання алгоритмів (MILA) при Університеті Монреаля, багато в чому був попередником TensorFlow. Він започаткував ідею використання графів статичних обчислень для автоматичної диференціації та компіляції коду як для CPU, так і для GPU.

Наприкінці 2015 року, після випуску TensorFlow, Keras було перетворено на мультібекендну архітектуру: стало можливим використовувати Keras або з Theano, або з TensorFlow. До вересня 2016 року TensorFlow досяг рівня технічної зрілості, коли стало можливим зробити його варіантом за замовчуванням для Keras. У 2017 році до Keras було додано два нових додаткових параметри серверної частини: CNTK (розроблений Microsoft) і MXNet (розроблений Amazon). Зараз і Theano, і CNTK вийшли з розробки, а MXNet не використовується широко за межами Amazon. Keras повертається до єдиного API — поверх TensorFlow.

Keras і TensorFlow багато років підтримували симбіотичні відносини. Протягом 2016 і 2017 років Keras став добре відомий як зручний спосіб розробки додатків TensorFlow, залучаючи нових користувачів до екосистеми TensorFlow. До кінця 2017 року більшість користувачів TensorFlow використовували його через Keras або в поєднанні з Keras. У 2018 році керівництво TensorFlow обрало Keras офіційним API високого рівня TensorFlow. Як наслідок, Keras API є головним у TensorFlow 2.0, випущеному у вересні 2019 року — масштабному оновленому дизайні TensorFlow і Keras, який враховує чотири роки відгуків користувачів і технічний прогрес.

2.7 Середовище для розробки глибокого навчання

Перш ніж розпочати розробку програм глибокого навчання, потрібно налаштувати середовище розробки. Дуже рекомендовано, хоча це і не обов'язково, запускати код глибокого навчання на сучасному графічному процесорі NVIDIA, а не на центральному процесорі комп'ютера. Деякі програми, зокрема обробка зображень із згортковими мережами, будуть нестерпно повільними для ЦП, навіть для швидкого багатоядерного процесора. І навіть для додатків, які реально можна запускати на центральному процесорі, як правило, збільшення швидкості буде в 5 або 10 разів за допомогою останнього графічного процесора.

Для глибокого навчання на GPU є три варіанти:

- придбати та встановити графічний процесор NVIDIA на свій комп'ютер (якщо відеокарта без підтримки CUDA);
- використати хмарні GPU в Google Cloud або AWS EC2;
- використати безкоштовне середовище від Colaboratory, яку пропонує Google, з підтримкою GPU і TPU прискорювачів.

Colaboratory — це найпростіший спосіб розпочати роботу, оскільки для цього не потрібно купувати обладнання та встановлювати програмне забезпечення — просто відкрити вкладку у браузері буде достаньо. Однак безкоштовна версія Colaboratory підходить лише для невеликих навантажень. Якщо потрібно збільшити масштаб, то доведеться платити за ресурси або ж використовувати перший або другий варіант.

Експерименти з глибоким навчанням у хмарі є простим і недорогим способом переходу до більших робочих навантажень без необхідності купувати додаткове обладнання. Розробка в хмарі проходить з допомогою Jupyter notebooks, тому досвід роботи в хмарі нічим не відрізняється від роботи локально.

Але якщо активно використовувати платне хмарне середовище для глибокого навчання, у довгостроковій перспективі або навіть протягом кількох місяців рахунок за послуги може стати дуже великим. Хмарні ресурси недешеві: у середині 2021 року вони коштували б 2,48 доларів США за годину за GPU V100 у Google Cloud. У той же час графічний процесор споживчого класу коштуватиме десь від 1500 до 2500 доларів США — ціна, яка з часом залишалася досить стабільною, навіть якщо характеристики цих графічних процесорів постійно вдосконалюються. Якщо ви активно використовуєте глибоке навчання, подумайте про налаштування локальної робочої станції з одним або кількома графічними процесорами.

Крім того, незалежно від того, працюєте ви локально чи в хмарі, краще операційну систему на базі Unix. Хоча технічно можливо запустити Keras у Windows безпосередньо, таке налаштування нерекомендується. Якщо необхідно провести глибоке навчання на Windows, найпростішим рішенням для того, щоб усе запустилося, є встановлення подвійного завантаження (dual boot) якогось Linux дистрибутиву на коп'ютер або використання підсистеми WSL, рівень сумісності, який дозволяє запускати програми Linux з Windows. Це може здатися клопотом, але в довгостроковій перспективі це заощадить багато часу та проблем.

2.8 Google Colabaratory середовище

Colaboratory (або скорочено Colab) — це безкоштовна служба Jupyter notebooks, яка не потребує інсталяції та повністю працює в хмарі. По суті, це веб-сторінка, яка дозволяє одразу писати та виконувати код Python з попередньо встановленими бібліотеками. Вони надають доступ до безкоштовного (але обмеженого) середовища виконання з GPU і навіть середовища виконання з TPU, тож не доведеться купувати власний GPU.

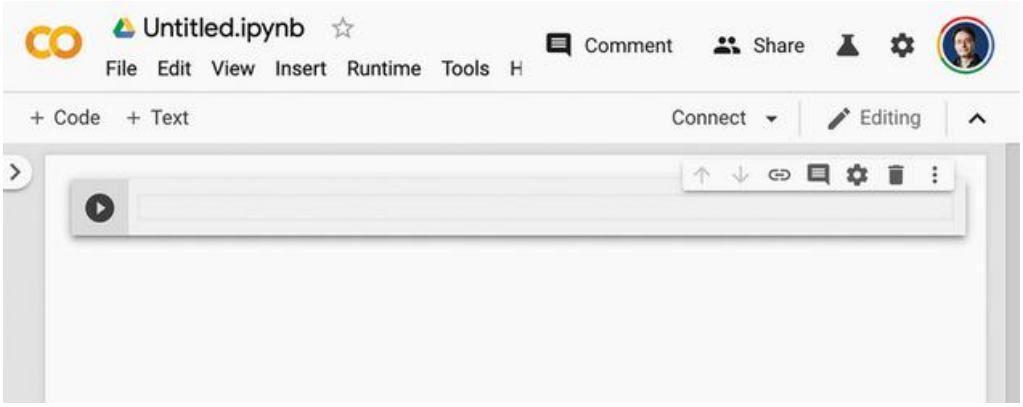

Рисунок 2.1 — Інтерфейс веб сторінки

Є дві кнопки на панелі інструментів: + Code і + Text. Вони призначені для створення комірок виконуваного коду Python і комірок тексту відповідно. Після введення коду в комірку коду, натискання Shift-Enter виконає його, рисунок 2.2.

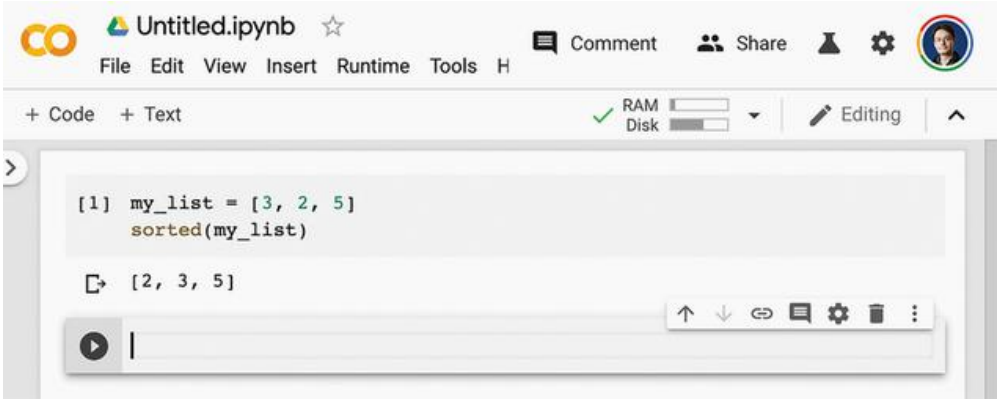

Рисунок 2.2 — Створення комірки з кодом

У текстовій комірці можна використовувати синтаксис Markdown. Натискання Shift-Enter на текстовій комірці відобразить її.

Текстові комірки корисні для надання читабельної структури Jupyter документам. Вони потрібні щоб анотувати код заголовками розділів і довгими абзацами з поясненнями або для відображеення рисунків.

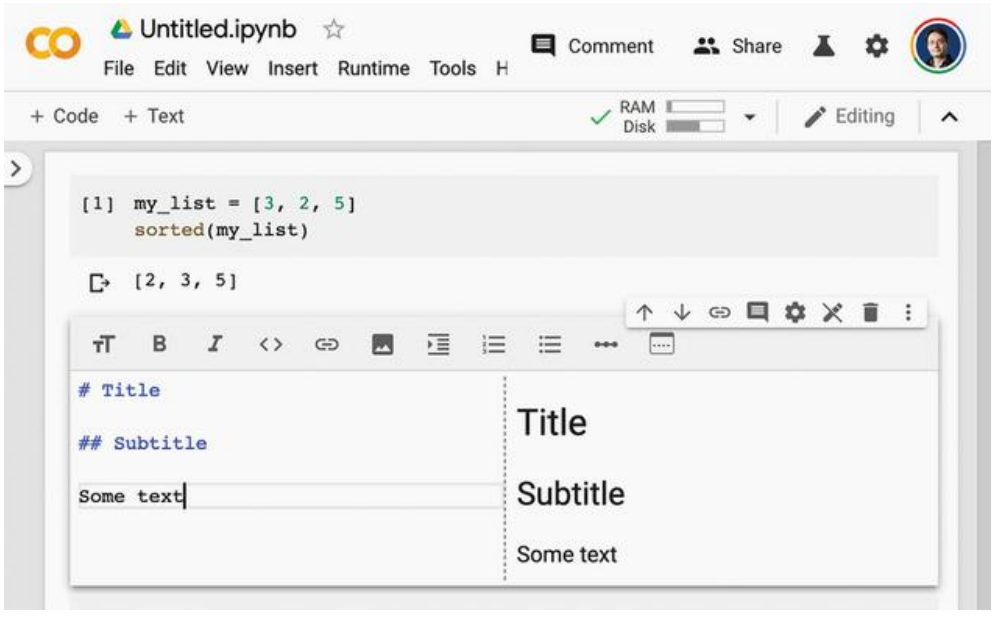

Рисунок 2.3 — Створення комірки з текстом

Середовище Colab за замовчуванням уже постачається з інстальованими TensorFlow і Keras, тож його можна використовувати одразу, без жодних кроків встановлення. Але якщо знадобиться інсталювати щось за допомогою pip, це можна зробити, використовуючи такий синтаксис у комірці коду рисунок 2.4.

```
!pip install package_name
```

Рисунок 2.4 — Встановлення пакетів з допомогою pip

Щоб використовувати середовище виконання з GPU у Colab, треба обрати у меню Runtime > Change Runtime Type і обрати GPU для Hardware Accelerator, рисунок 2.5.

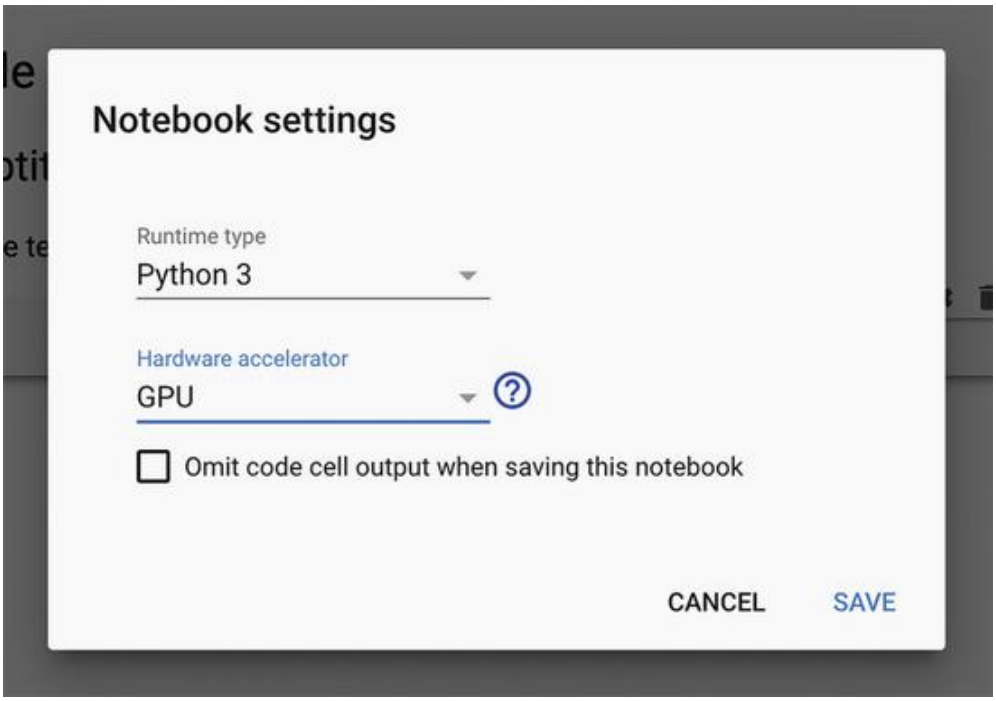

Рисунок 2.5 — Налаштування типу середовища

2.9 Узагальнення в машинному навчані

Причина не оцінювати моделі на тих самих даних, на яких вони навчалися, очевидна: лише через кілька епох, продуктивність моделі на даних, які раніше не бачилися, починає відрізнятися від продуктивності моделі на даних навчання, яка завжди покращується в міру навчання. Моделі починають перенавчатися.

Фундаментальною проблемою машинного навчання є суперечність між оптимізацією та узагальненням. Оптимізація стосується процесу коригування моделі для отримання найкращої можливої продуктивності

на навчальних даних, тоді як узагальнення стосується того, наскільки добре навчена модель працює з даними, яких вона ніколи раніше не бачила. Звичайно, мета — отримати хороше узагальнення, але ми не можемо контролювати узагальнення; ми можемо лише адаптувати модель до її навчальних даних. Якщо зробити це надто добре, спрацьовує перенавчання, а узагальнення страждає.

На початку навчання оптимізація та узагальнення співвідносяться: чим менше похибка на навчальних даних, тим менше похибка на тестових даних. Поки це відбувається, модель недонавчається, мережа ще не змоделювала всі релевантні кореляції в навчальних даних. Але після певної кількості ітерацій з навчальними даними узагальнення перестає покращуватися, показники валідації перестають покращюватися, а потім починають погіршуватися — модель починає перенавчатися. Тобто модель починає вивчати кореляції, які є специфічними для навчальних даних, але які вводять в оману або не мають значення, коли йдеться про нові дані, рисунок 2.6.

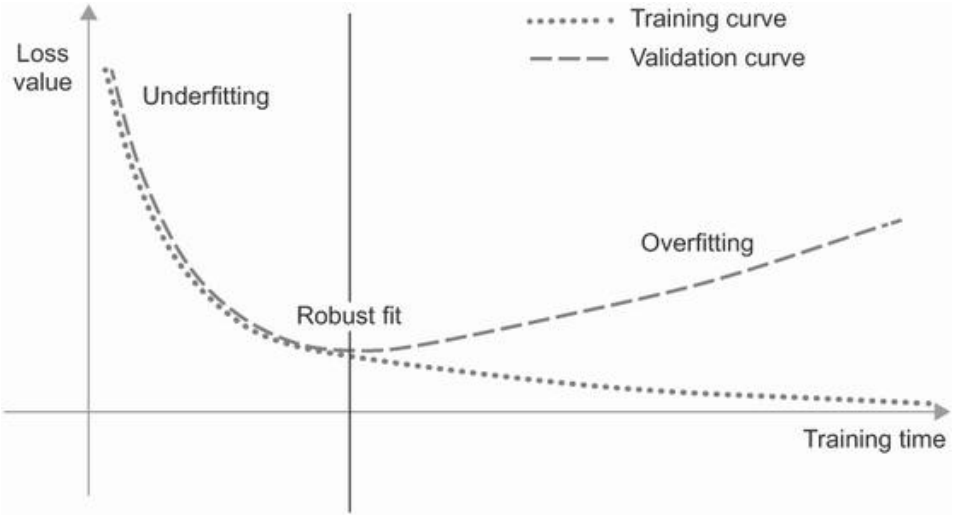

Рисунок 2.6 — Графік навчання моделі

Одна з проблем що впливає на перенавчання це набір даних з “шумами”. Якщо модель знаходить шлях щоб включити такі викиди, її ефективність узагальнення погіршиться, як показано на рисунку 2.7.

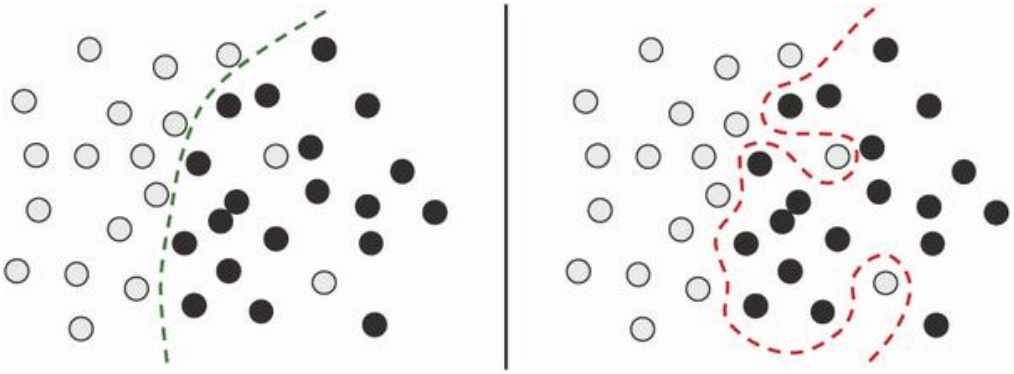

Рисунок 2.7 — Викиди в наборі даних; правильний розподіл і перенавчання

Не всі шуми в даних виникають через неточності — навіть ідеально чисті та акуратно позначені дані можуть бути шумними, якщо проблема пов’язана з невизначеністю та двозначністю. У завданнях класифікації часто буває, що деякі області простору вхідних ознак пов’язані з кількома класами одночасно. Припустімо, розробляється модель, яка бере зображення банана та передбачає, чи є банан незрілим, стиглим чи гнилим. Ці категорії не мають об’єктивних меж, тому одна й та сама картина може бути класифікована як недозріла або стигла різними людьми, які ставлять етикетки. Подібним чином багато проблем пов’язані з випадковістю. Наприклад, використання даних атмосферного тиску, щоб передбачити, чи буде завтра дощ, але за тими самими вимірюваннями іноді може йти дощ, а іноді — ясне небо, з певною ймовірністю.

Модель може бути занадто адаптованою до таких імовірнісних даних, якщо бути занадто впевненою щодо неоднозначних областей простору ознак, як на рисунку 2.7. Краща модель би ігнорувала окремі точки даних і розглядала ширшу картину.

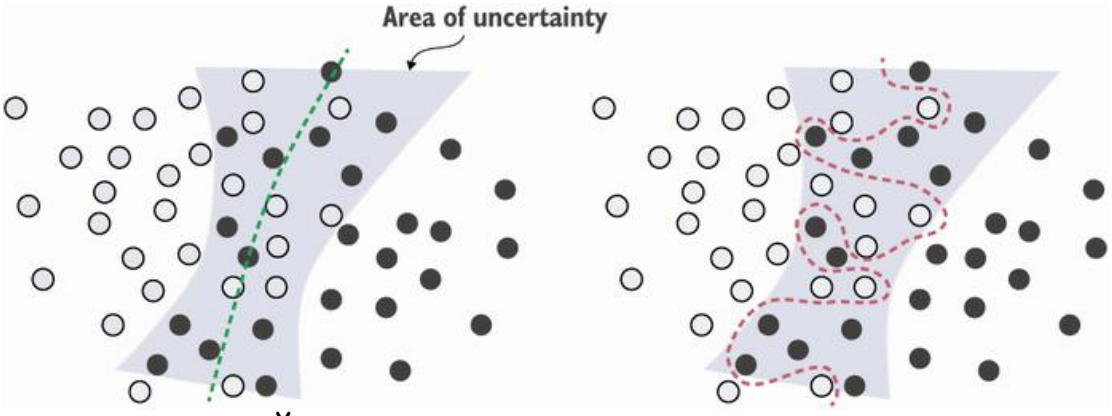

Рисунок 2.8 — Ймовірності в наборі даних; правильний розподіл і перенавчання

Якщо модель бачила всього лише двох помаранчевих смугастих котів за навчання, і обидва вони виявилися надзвичайно асоціальними, то вона зробить висновок, що помаранчеві смугасті коти, як правило, є асоціальними. Це перенавчання. Якби модель побачила більшу різноманітність котів, у тому числі з більшою кількістю помаранчевих, вона б зрозуміла, що колір котів погано корелює з характером.

Подібним чином моделі машинного навчання, навчені на наборах даних, які включають рідкісні ознаки — дуже сприйнятливі до перенавчання. У завданні класифікації тональності, якщо слово «баляндраси» з'являється лише в одному тексті в навчальних даних, і цей текст має негативний настрій, погана модель може надати дуже високу вагу на це слово і завжди класифікувати нові тексти, в яких згадується баляндраси, як негатив, тоді як, об'єктивно, нічого негативного в баляндрасах немає.

Важливо, що значення ознаки не обов'язково має з'являтися лише кілька разів, щоб призвести до помилкових кореляцій. Наприклад слово, яке зустрічається в 100 зразках тренувальних даних і пов'язане з позитивним настроєм у 54% випадків і з негативним у 46% випадків. Ця різниця цілком може бути повною статистичною випадковістю, але модель, швидше за все, навчиться використовувати цю кореляцію для завдання класифікації. Це одне з найпоширеніших джерел перенавчання.

Дані з викидами неминуче призводять до перенавчання. Таким чином, у випадках, коли немає впевненості, чи є функції інформативними чи відволікаючими, потрібно зробити вибірку ознак перед навчанням. Наприклад, обмеження текстових наборів даних найпоширенішими словами, наприклад топ десять тисяч, є грубою формою відбору ознак. Типовий спосіб вибору ознак полягає в тому, щоб обчислити певну оцінку корисності для кожної доступної ознаки — міру того, наскільки інформативна ознака щодо завдання, наприклад взаємну інформацію між ознакою та мітками — і зберегти лише ознаки, які вище деякого порогу.

2.10 Оцінка моделей машинного навчання

Ми можемо контролювати лише те, що можемо спостерігати. Оскільки мета машинного навчання — розробити моделі, які можуть успішно узагальнювати нові дані, дуже важливо мати можливість надійно виміряти узагальнюючу здатність моделі.

Оцінка моделі завжди зводиться до поділу доступних даних на три набори: навчання, перевірка та тестування. Модель тренується на даних

навчання, а оцінюється на даних перевірки. Коли модель буде готова до використання, залишається остання перевірка на тестових даних, які повинні бути максимально схожими на виробничі дані. Потім можна розгортати модель у виробництві.

Чому не використати лише два набори? Причина полягає в тому, що розробка моделі завжди передбачає налаштування її конфігурації: наприклад, вибір кількості шарів або розміру шарів (так звані гіперпараметри моделі, щоб відрізнити їх від параметрів, які є вагами мережі). Розробник виконує цю настройку, використовуючи як сигнал зворотного зв'язку продуктивність моделі на даних перевірки. По суті, це налаштування є формою навчання: пошук гарної конфігурації в деякому просторі параметрів. Як результат, налаштування конфігурації моделі на основі її продуктивності в наборі перевірки може швидко призвести до перенавчання набору перевірки, навіть якщо модель ніколи не навчали безпосередньо на ньому.

Центральним у цьому явищі є поняття витоку інформації. Щоразу, коли інжинер налаштовує гіперпараметр моделі на основі продуктивності моделі в наборі перевірки, деяка інформація про дані перевірки просочується в модель. Якщо зробити це лише один раз для одного параметра, тоді вийде дуже мало бітів інформації, і набір перевірки залишиться надійним для оцінки моделі. Але якщо повторювати це багато разів — провести один експеримент, оцінити набір перевірки та в результаті змінити модель, — тоді буде просочуватися все більший обсяг інформації про набір перевірки в модель.

Зрештою, якщо використовувати лише два набори, то модель буде штучно добре працювати на даних перевірки, оскільки саме для цього її оптимізували. Зазвичай ми дбаємо про продуктивність на абсолютно нових даних, а не на даних перевірки, тому вам потрібно використовувати зовсім інший, ніколи раніше не бачений набір даних для оцінки моделі: тестовий набір даних. Модель не повинна мати доступу до будь-якої інформації про тестовий набір, навіть опосередковано. Якщо щось у моделі було налаштовано на основі продуктивності тестового набору, то показник узагальнення буде помилковим.

2.11 Проблема XOR і її розв'язання з допомогою Python+Keras

Виняткова диз'юнкція або додавання за модулем два -- це логічна операція що набуває значення істина коли лише один із аргументів є істинним. Таблиця істинності наведена в таблиці 2.1.

Таблиця 2.1 — Таблиця істинності XOR функції

A	B	XOR(A, B)
0	0	0
0	1	1
1	0	1
1	1	0

Проблему буде розв’язано з допомогою бібліотеки Keras. Щоб почати роботу з нею треба встановити мову Python та бібліотеку Tensorflow. Так як Keras тепер є частиною Tensorflow, то буде достатньо встановити лише його.

Я використовую середовище Anaconda, тому для встановлення всього необхідного треба лише ввести

```
$ conda install tensorflow=2.9.1 python ipykernel
```

Після чого можна розпочинати писати код програми. Програма починається з імпортування бібліотек. Пакет keras є в просторі імен tensorflow, але він також доступний як окремий пакет. Імпортуємо keras та декілька важливий класів з keras.layers.

```
1 import keras
2 import tensorflow as tf
3 from keras.models import Sequential
4 from keras.layers import Dense, Dropout, Activation
```

Рисунок 2.9 — Секція з імпортами

Після чого треба задати дані для тренування. Зазвичай, роблять як мінімум два набори даних: для навчання і для валідації. В даному випадку усі можливі комбінації вхідних даних вже перераховано. Вхідні дані розбиті на колонки та рядки, як і в таблиці істинності. Так першим значенням з масиву features є варіант коли обидва вхідні значення є нулями. Йому відповідає перший елемент масиву labels, це значення функції від двох аргументів з першого елементу першого масива.

```
1 features = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
2 labels = np.array([0, 1, 1, 0])
```

Рисунок 2.10 — Вхідні данні

Найважливіша частина це визначення архітектури мережі. Дана модель використовує Sequential архітектуру з двома вхідними тензорами, трьома тензорами в скритому шарі та один вихідний тензор. Sequential або послідовна архітектура це де кожен шар має точно один вхідний тензор і один вихідний тензор. А тензором у Tensorflow прийнято називати один нейрон.

З коду видно що модель має три шари. Кожен шар має свою функцію активації: перший tanh; другий tanh; третій sigmoid. Їх графіки можна побачити на рисунках 2.4 та 2.5.

Після визначення архітектури треба задати параметри навчання, це робиться в методі compile. Він приймає багато різних параметрів, але для цієї моделі нам потрібно лише визначити функцію помилки та опціонально передати масив метрик що будуть виводитися при навчанні мережі. Так як модель має лише дві категорії, тобто навчається на данних в яких може бути лише 0 чи 1 на виході, в якості функції помилки використовується бінарна кросентропія (Binary Crossentropy). Для того щоб знати коли припинити навчання можна використати Binary Accuracy, це вид точності коли числа більше 0.5 вважаються одиницею, а числа менше 0.5 нулем.

Останнім рядком я виводжу дані про архітектуру моделі. Там можна побачити кількість параметрів для тренування.

```
model = Sequential(
    [
        Dense(2, activation="tanh", input_shape=(2,)),
        Dense(3, activation="tanh"),
        Dense(1, activation="sigmoid")
    ]
)
loss = keras.losses.BinaryCrossentropy()
metrics = [keras.metrics.BinaryAccuracy()]
model.compile(loss=loss, metrics=metrics)
model.summary()
```

Рисунок 2.11 — Секція задання архітектури моделі

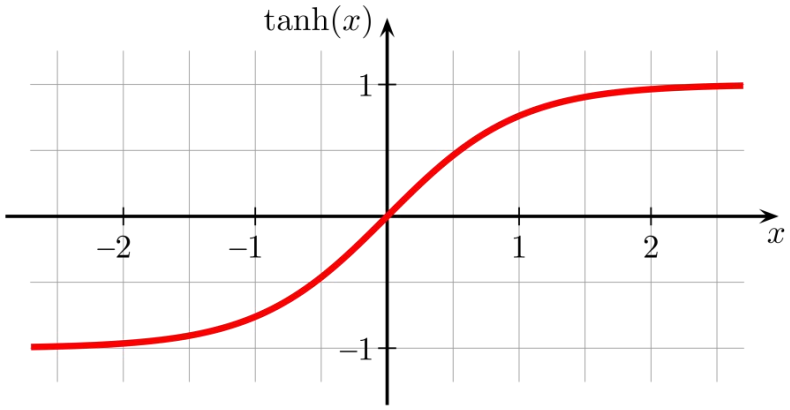

Рисунок 2.12 — Графік функції tanh

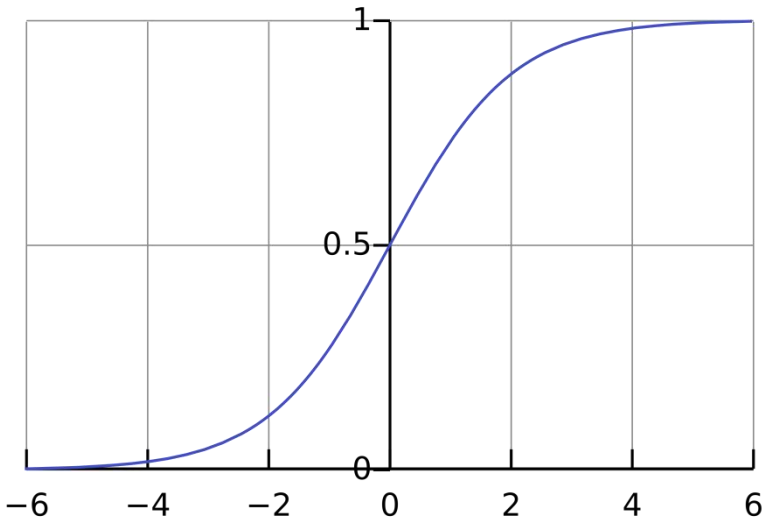

Рисунок 2.13 — Графік функції sigmoid

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 dense_3 (Dense)              (None, 2)                 6

 dense_4 (Dense)              (None, 3)                 9

 dense_5 (Dense)              (None, 1)                 4

=================================================================
Total params: 19
Trainable params: 19
Non-trainable params: 0
_________________________________________________________________
```

Рисунок 2.14 — Вивід підсумовуючої інформації після компіляції моделі

Для тренування цієї моделі треба викликати метод fit та передати дані для тренування. Опціонально вказавши кількість епох, розмір партії та інші параметри.

```
model.fit(features, labels, epochs=1000, batch_size=10)
model.summary()
```

Рисунок 2.15 — Параметри тренування моделі

Під час тренування в термінал буде виводитися інформація за кожну епоху. Можна побачити що помилка (loss) зменшується з кожною ітерацією. На рисунку видно що точність моделі вже досягла ста відсотків, тому не має сенсу зменшувати помилку до нуля.

```
Epoch 1/1500
1/1 [==============================] - 0s 7ms/step - loss: 0.3092 - binary_accuracy: 1.0000
Epoch 2/1500
1/1 [==============================] - 0s 11ms/step - loss: 0.3088 - binary_accuracy: 1.0000
Epoch 3/1500
1/1 [==============================] - 0s 51ms/step - loss: 0.3084 - binary_accuracy: 1.0000
Epoch 4/1500
1/1 [==============================] - 0s 30ms/step - loss: 0.3081 - binary_accuracy: 1.0000
Epoch 5/1500
1/1 [==============================] - 0s 25ms/step - loss: 0.3077 - binary_accuracy: 1.0000
Epoch 6/1500
1/1 [==============================] - 0s 12ms/step - loss: 0.3074 - binary_accuracy: 1.0000
Epoch 7/1500
1/1 [==============================] - 0s 24ms/step - loss: 0.3070 - binary_accuracy: 1.0000
Epoch 8/1500
1/1 [==============================] - 0s 16ms/step - loss: 0.3066 - binary_accuracy: 1.0000
Epoch 9/1500
1/1 [==============================] - 0s 29ms/step - loss: 0.3063 - binary_accuracy: 1.0000
```

Рисунок 2.16 — Вивід під час тренування моделі

І це все що треба для створення моделі з допомогою бібліотеки Keras. Так як даних для валідації немає, то не треба перевіряти модель на них. Це повністю готова модель.

2.12 Класифікація зображень

Розпізнавання зображень або ком’ютерне бачення стало одним із рушіїв популяризації та подальшого розвитку глибокого навчання, тому що глибоке навчання дуже добре підходить для вирішення цієї задачі. Перед ГН використовували неглибокі моделі, що вимагало ручної людської роботи по виборці ознак, а це дуже складна робота що займала велику частку часу при розв’язані проблем.

В даному розділі буде розв’язано задачу класифікації зображень домашніх тварин на дві категорії: собак і котів. Набір даних для навчання

буде взято з сайту Kaggle https://www.kaggle.com/competitions/dogs-vs-cats/data.

Для вирішення таких типів задач треба використовувати згорткові нейронні мережі. Як вони працюють було описано у першому розділі.

Перше що відрізняється це набір даних, який тепер має вигляд картинок. Так як такі набори можуть займати багато гігабайтів то їх неможливо завантажити в пам'ять одночасно, тому треба використовувати спеціальний клас Dataset. На рисунку 2.9 показано набір даних що буде використано для тренування розрізнення між котами та собаками. Всього лише тисяча картинок для кожного класу. Це дуже маленький набір даних для задачі розпізнавання, але використовуючи аугментацію даних, можна дуже сильно сповільнити перенавчання на такому наборі.

```
├── test
│   ├── cat  [1000 entries]
│   └── dog  [1000 entries]
├── train
│   ├── cat  [1000 entries]
│   └── dog  [1000 entries]
└── validation
    ├── cat  [500 entries]
    └── dog  [500 entries]
```

Рисунок 2.17 — Структура папок набору даних

Перед тим як тренувати модель, треба обробити вхідні дані. Так наприклад, модель не може опрацьовувати зображення будь якого розміру, тому як крок попередньої обробки, треба змінити розмір усіх зображень. Це був би не дуже легкий процес, але на щастя в бібліотеці Keras передбачили таке використання і бібліотека має функцію image_dataset_from_directory, що автоматично попередньо обробить файли, див рисунок 2.10. А ще, ця функція працює по розумному, і не завантажує усі картинки одночасно у пам'ять комп'ютера, тому вона може обробляти набори даних з мільойнів картинок.

```python
from tensorflow.keras.utils import image_dataset_from_directory

def make_dataset(name):
    return image_dataset_from_directory(
        new_base_dir / name,
        image_size=(180, 180),
        batch_size=32,
    )

train_dataset = make_dataset("train")
validation_dataset = make_dataset("validation")
test_dataset = make_dataset("test")
```

Рисунок 2.18 — Створення змінних класу Dataset

Далі треба створити модель, тобто визначити її архітектуру. Модель буде приймати на вхід зображення розміром 180 на 180 пікселей з трьома каналами кольорів, тобто RGB кодування. Із-за використання маленького набору даних, треба активно боротися з перенавчанням, а для цього використаю аугментацію даних та Dropout шар.

```
from tensorflow import keras
from tensorflow.keras import layers

data_augmentation = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.2),
    ]
)

inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)

model.summary()
```

Рисунок 2.19 — Архітектура моделі у функціональному стилі

Останній рядок, model.summary() виводить на екран кількість параметрів для навчання, див рисунок 2.12. Як бачимо, кожні декілька шарів мережі розмір вхідних даних зменшується по висоті і ширині вдвічі, а розмір фільтрів, тобто останнього числа, збільшується. Це поширена архітектура згорткових нейронних мереж. Також це не маленька модель, майже мільйон параметрів для тренування, можна зробити деякі оптимізації щоб зменшити кількість параметрів, але це не завжди пришвидшить навчання.

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 180, 180, 3)]     0

 rescaling_1 (Rescaling)     (None, 180, 180, 3)       0

 conv2d_5 (Conv2D)           (None, 178, 178, 32)      896

 max_pooling2d_4 (MaxPooling  (None, 89, 89, 32)       0
 2D)

 conv2d_6 (Conv2D)           (None, 87, 87, 64)        18496

 max_pooling2d_5 (MaxPooling  (None, 43, 43, 64)       0
 2D)

 conv2d_7 (Conv2D)           (None, 41, 41, 128)       73856

 max_pooling2d_6 (MaxPooling  (None, 20, 20, 128)      0
 2D)

 conv2d_8 (Conv2D)           (None, 18, 18, 256)       295168

 max_pooling2d_7 (MaxPooling  (None, 9, 9, 256)        0
 2D)

 conv2d_9 (Conv2D)           (None, 7, 7, 256)         590080

 flatten_1 (Flatten)         (None, 12544)             0

 dropout_1 (Dropout)         (None, 12544)             0

 dense_1 (Dense)             (None, 1)                 12545

=================================================================
Total params: 991,041
Trainable params: 991,041
Non-trainable params: 0
_________________________________________________________________
```

Рисунок 2.20 — Опис архітектурии мережі і параметрів до тренування

Keras має спеціальну функцію для відображення графу архітектури, але у даному випадку він не дуже цікавий бо архітектура повноз'єднана, тобто кожен шар під'єднаний до наступного.

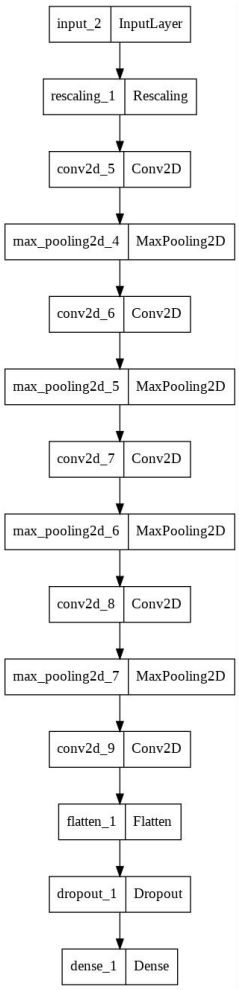

Рисунок 2.21 — Граф архітектури моделі

В якості оптимізатора обрано rmsprop. Він добре підходить під багато задач. Функція помилки буде binary_crossentropy, тому що набір даних має лише дві категорії, тобто бінарний.

```
model.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])
```

Рисунок 2.22 — Вибір функції помилки і оптимізатора

Після тренування моделі на 30 епохах було побудовано графік зміни помилки та точності від епохи, див рисунок 2.15. З графіків видно що десь після одинадцятої епохи, помилка на даних валідації почала зростати. А пікова точність на даних валідації була 75% приблизно на 25 епосі. Хоча і було застосовано два методи проти перенавчання, але все одно це не змінює того факту що кількість даних для навчання замала.

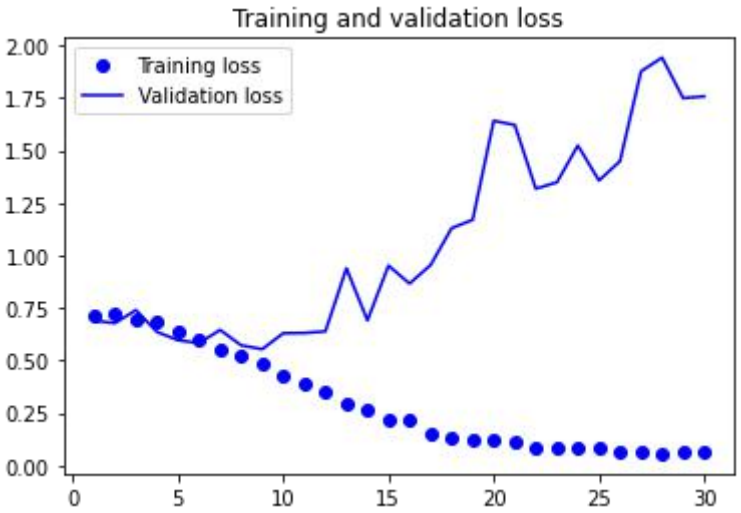

Рисунок 2.23 — Графік помилки від епохи

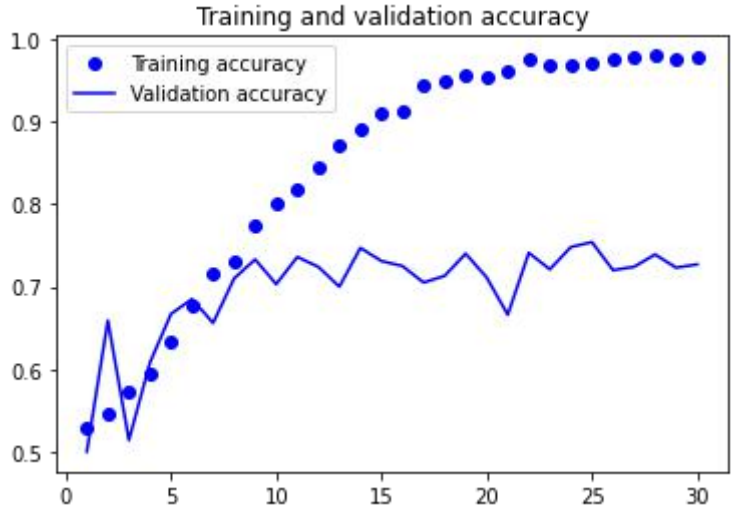

Рисунок 2.24 — Графік точності від епохи

2.13 Висновки за розділом

Машинне навчання отримало значну популярність за мунулі десятиріччя. З такою популярністю почали робити більше досліджень; більше програмних інструментів; навчальних матеріалів та іншого. Зараз машиним навчанням можуть займатися не тільки науковці з багажем теоретичних знань, а й звичайні програмісти.

Виявляється що машинне навчання займає ключову роль у всії великих програмних системах. Будь то Google, Meta, Apple, усі вони використовують ті або інші варіанти машинного навчання для задоволення потреб їх користувачів, для оптимізації пошукових запитів, для систем рекомендацій контента.

У цьому розділі було створенно найпростішу модель на базі бібліотеки Keras. Написання моделі на цій бібліотеці займає лише декілька

стрічок коду. Перша це архітектура моделі, тобто зв’язність тензорів, кількість вхідних та вихідних тензорів, кількість скритих шарів та функції активації для кожного шару. Після чого модель готова тренуватися за наданими даними

3 Бінарна класифікація спаму

3.1 Спам

Спам – це будь-яке небажане повідомлення, надіслане масово. Спам, який зазвичай надсилається електронною поштою, також поширюється через текстові повідомлення (SMS), соціальні мережі або телефонні дзвінки. Спам-повідомлення часто надходять у формі нешкідливих (хоча і дратівливих) рекламних електронних листів. Але іноді спам є шахрайським або зловмисним шахрайством.

Походження терміну «спам» для масових повідомлень відноситься до сценки Монті Пайтона. У ньому група відвідувачів (одягнених у костюми вікінгів) голосно та неодноразово проголошує, що всі мають їсти спам (spiced ham або шинка зі спеціями) — хочуть вони цього чи ні. Це схоже на те, як електронний спамер заповнює папку "Вхідні" небажаними повідомленнями.

Спам може варіюватися від дратівливих електронних листів до різних типів інтернет-спаму, як-от коментарі в соціальних мережах, повні надмірних посилань, або навіть сенсаційні заголовки в ЗМІ та на інших веб-сайтах, які не можливо не побачити [10].

3.1.1 SEO спам

SEO-спам означає маніпулювання методами пошукової оптимізації (SEO) для покращення рейтингу веб-сайту спамера в пошукових системах. Ця категорія ділиться на дві великі категорії.

— Спам вмістом

Деякі спамери наповнюють свої сторінки популярними ключовими словами, щоб спробувати підвищити рейтинг сторінок свого веб-сайту, коли люди здійснюють пошук за цими ключовими словами. Інші використовуватимуть наявний вміст без дозволу, щоб зробити свої власні сторінки більш суттєвими та унікальними.

— Спам посиланнями

Зазвичай це коментар в блозі чи на форумі, наповнений нерелевантними посиланнями. Спамер намагається використати механізм SEO, відому як «зворотні посилання», щоб залучити трафік на свою сторінку.

3.1.2 Спам у соціальних мережах

З розвитком соціальних медіа спамери швидко користуються перевагами цих платформ, поширюючи свій спам через ботів та інші невідомі облікові записи. Більшість спаму в соціальних мережах містить посилання на комерційні сторінки, метою яких є збільшення трафіку або прибутку веб-сайту спамера.

3.1.3 Спам SMS повідомлення та спам-дзвінки

Деякі спамери надсилають текстові повідомлення (SMS), push-повідомлення або навіть дзвонять на мобільний телефон. Спам також може приймати форму миттєвих повідомлень через такі популярні програми обміну повідомленнями, як Telegram, Skype і Instagram. Найкраще блокувати спам-повідомлення та дзвінки від підозрюваних спамерів, не відповідати на дивні повідомлення та ніколи не натискати посилання на спам-повідомлення.

3.1.4 Шахрайство з поточними подіями

Потік сенсаційних новин, що публікується щодня, дає спамерам можливість використовувати заголовки, щоб заробити на трагедіях чи політичних подіях. Наприклад, спамери можуть відправити спам із проханням внести свій внесок у кампанію збору коштів, яка не є законною.

3.2 Email спам

Спам в електронній пошті це те коли з’явилося слово спам. Напевно це й досі найпоширеніша форма відправки небажанних повідомлень. Хоча з розвитком штучного інтелекту та глибокого навчання, сервіси протидії спаму збільшували свою силу та точність розпізнавання.

І справді, методи машинного навчання підходять дуже добре під цей тип задач. Більшість повідомлень будуть ліквідовані мережею, а якщо якісь і пройшли фільтр то користувачі email клієнта позначать це

повідомлення як спам, тобто дадуть нову інформацію нейронній мережі, на якій вона потім буде навчатися.

3.3 Статистика Email повідомлень

За даними опитування Radicati[23] у 2023 році кожен день буде відправлятися 350 мільярдів повідомлень, а кількість користувачів поштових сервісів сягне 4.3 мільярди. Якщо припустити що в середньому одне повідомлення містить 100 байтів, хоча насправді це значення набагато більше, то це дає число 32 тера байти даних в день. Тому швидка та ефективна обробка спам повідомлень має велике значення.

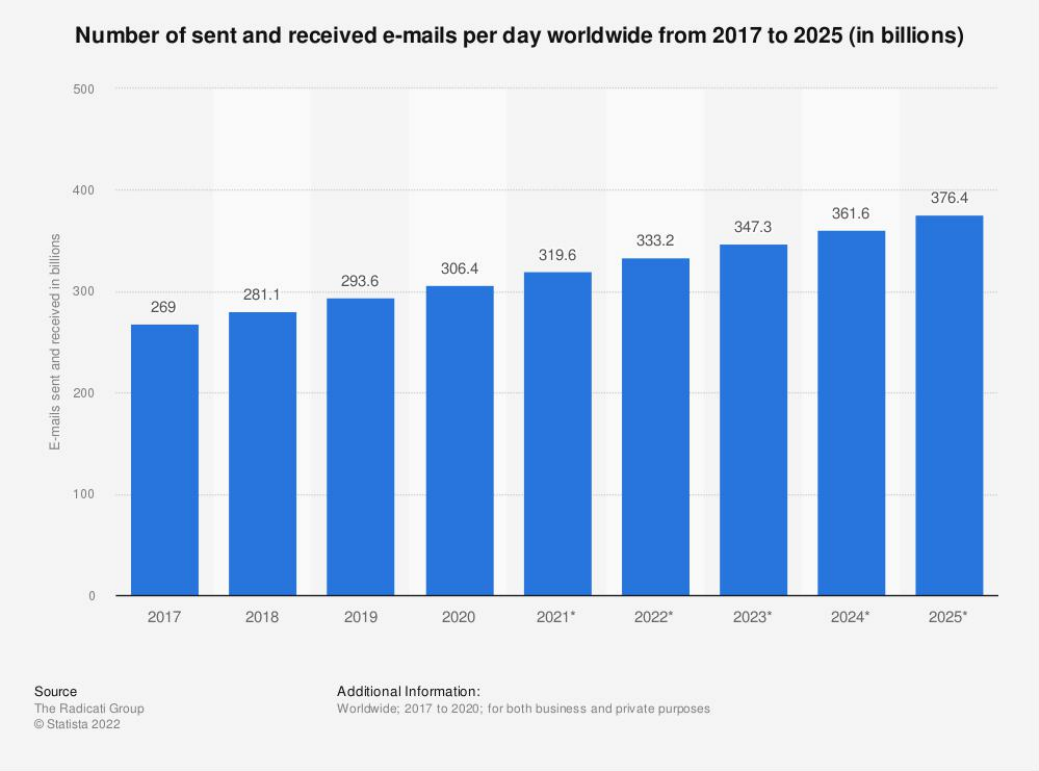

Рисунок 3.1 — Графік відправлених і отриманих email повідомлень

3.6 Набір даних

Apache SpamAssassin — це платформа захисту від спаму із відкритим сирцевим кодом, яка надає системним адміністраторам фільтр для класифікації електронної пошти та блокування спаму (небажаної масової електронної пошти).

Він використовує надійну систему підрахунку балів і плагіни для інтеграції широкого спектру розширених евристичних і статистичних тестів аналізу заголовків і основного тексту електронних листів,

включаючи аналіз тексту, байєсівську фільтрацію, списки блокування DNS і бази даних спільної фільтрації.

Рисунок 3.2 — Логотип програми

Apache зробили доступними набір даних зі спам та не спам повідомленнями. Частина цього набору даних буде використано для тренування моделі глибокого навчання. Повний набір даних доступний для всіх бажаючих за даним посиланням https://spamassassin.apache.org/old/publiccorpus/.

3.7 Формат набору даних та попередня обробка

Лист електронної пошти може складатися по різному. Це може бути:

- текстове повідомлення;
- вкладені файли;
- HTML;
- комбінації тексту, HTML, картинок та вкладених файлів.

З цього випливає що можна по різному розібрати електронний лист на його складники. Для тренування моделі я обрав метод аналізу тексту листа. Так як спам розрахований на те щоб його прочитали або клікнули, то переважну більшість часу він має якийсь текст для людини, щоб змусити її клікнути. Тому очевидним способом виявлення спаму є аналіз

тільки текстової частини повідомлення, тобто видалення усілякого форматування такого як HTML та усіх файлів.

Набір даних поставляється як файли у вихідному форматі email. Тому для початку буде використано бібліотеку email яка поставляється разом з мовою програмування Python.

```
From 12a1mailbot1@web.de  Thu Aug 22 13:17:22 2002
Return-Path: <12a1mailbot1@web.de>
Delivered-To: zzzz@localhost.example.com
Received: from localhost (localhost [127.0.0.1])
    by phobos.labs.example.com (Postfix) with ESMTP id 136B943C32
    for <zzzz@localhost>; Thu, 22 Aug 2002 08:17:21 -0400 (EDT)
Received: from mail.webnote.net [193.120.211.219]
    by localhost with POP3 (fetchmail-5.9.0)
    for zzzz@localhost (single-drop); Thu, 22 Aug 2002 13:17:21 +0100 (IST)
Received: from dd_it7 ([210.97.77.167])
    by webnote.net (8.9.3/8.9.3) with ESMTP id NAA04623
    for <zzzz@example.com>; Thu, 22 Aug 2002 13:09:41 +0100
From: 12a1mailbot1@web.de
Received: from r-smtp.korea.com - 203.122.2.197 by dd_it7  with Microsoft
SMTPSVC(5.5.1775.675.6);
     Sat, 24 Aug 2002 09:42:10 +0900
To: <dcek1a1@netsgo.com>
Subject: Life Insurance - Why Pay More?
Date: Wed, 21 Aug 2002 20:31:57 -1600
MIME-Version: 1.0
Message-ID: <0103c1042001882DD_IT7@dd_it7>
Content-Type: text/html; charset="iso-8859-1"
Content-Transfer-Encoding: quoted-printable
```

Рисунок 3.3 — Вигляд заголовка email з категорії спам

На рисунку 3.4 показано парсинг вихідного формату email в клас Message, який далі буде оброблено іншими функціями.

```
import pathlib
import os
import email
from email import policy

dataset_path = pathlib.Path("datasets/email-spam/")


def parse_email(path, is_spam):
    subfolder = "ham"
    if is_spam:
        subfolder = "spam"

    with open(os.path.join(dataset_path/subfolder, path), 'rb') as f:
        return email.parser.BytesParser(policy=policy.default).parse(f)


ham_emails = [parse_email(path, False)
              for path in sorted(os.listdir(dataset_path/"ham"))]
spam_emails = [parse_email(path, True)
               for path in sorted(os.listdir(dataset_path/"spam"))]
```

Рисунок 3.4 — Код завантаження електронних листів як email.Message клас

Після парсингу електронних листів у класи, треба звірити що усі потрібні файли були зчитані. На рисунку 3.5 виводиться базова інформація про кількість листів та співвідношення спам листів до не спам. Це не збалансований набір даних, тому це треба враховувати при навчанні і зробити відповідні рішення на гіперпараметри моделі.

```
print('Amount of ham files:', len(ham_emails))
# Amount of ham files: 2551
print('Amount of spam files:', len(spam_emails))
# Amount of spam files: 501
print('Spam to Ham Ratio:', len(spam_emails)/len(ham_emails))
# Spam to Ham Ratio: 0.1963
```

Рисунок 3.5 — Інформація про набір даних

Після звірення, що всі дані успішно завантажені треба витягнути текст з електроних листів. Електронна пошта може мати багато форматів, один з них це Multipart, тобто лист містить декілька інших різних форматів всередені. Тому треба передбачи усі можливі формати.

```
from bs4 import BeautifulSoup


def html_to_plain(email):
    try:
        soup = BeautifulSoup(email.get_content(), 'html.parser')
        return soup.text.replace('\n\n', '')
    except:
        return "empty"


def email_to_plain(email):
    for part in email.walk():
        partContentType = part.get_content_type()
        if partContentType not in ['text/plain', 'text/html']:
            continue
        try:
            partContent = part.get_content()
        except:  # in case of encoding issues
            partContent = str(part.get_payload())
        if partContentType == 'text/plain':
            return partContent
        else:
            return html_to_plain(part)


email_to_plain(ham_emails[0])
```

Рисунок 3.6 — Витягування текстових даних з електроних листів

Після кожного кроку обробки даних, треба перевіряти чи результат задовільняє очікування. На рисунку 3.7 показаний один з листів після витягування лише текстових даних з нього.

```
Output exceeds the size limit. Open the full output data in a text editor
    Date:        Wed, 21 Aug 2002 10:54:46 -0500
    From:        Chris Garrigues <cwg-dated-1030377287.06fa6d@DeepEddy.Com>
    Message-ID:  <1029945287.4797.TMDA@deepeddy.vircio.com>


  | I can't reproduce this error.

For me it is very repeatable... (like every time, without fail).

This is the debug log of the pick happening ...

18:19:03 Pick_It {exec pick +inbox -list -lbrace -lbrace -subject ftp -rbrace -rbrace}
  {4852-4852 -sequence mercury}
18:19:03 exec pick +inbox -list -lbrace -lbrace -subject ftp -rbrace -rbrace 4852-4852
  -sequence mercury
18:19:04 Ftoc_PickMsgs {{1 hit}}
18:19:04 Marking 1 hits
18:19:04 tkerror: syntax error in expression "int ...

Note, if I run the pick command by hand ...

delta$ pick +inbox -list -lbrace -lbrace -subject ftp -rbrace -rbrace  4852-4852
  -sequence mercury
1 hit

That's where the "1 hit" comes from (obviously).  The version of nmh I'm
using is ...
...
Exmh-workers@redhat.com
https://listman.redhat.com/mailman/listinfo/exmh-workers
```

Рисунок 3.7 — Приклад одного з оброблених листів

Нейронним мережам не обов’язково показувати правильні слова, насправді модель можна навчати на повністю випадкових даних. Одна з оптимізацій текстових наборів даних полягає в прибранні афіксів слів, залишаючи тільки основу слова. Для цього є бібліотека з назвою nltk (natural language tool kit). На рисунку 3.8 і 3.9 показано мінімальний приклад програми та її вивід. Цей шматок коду виконує функцію stem, що залишає тільки основу слова і виводить результат. На даний час nltk не підтримує українську мову, але це не завадить бо використовується набор даних на англійській мові.

```python
import nltk

stemmer = nltk.PorterStemmer()
for word in ("Working", "Work", "Works", "Worked"):
    print(word, "=>", stemmer.stem(word))
```

Рисунок 3.8 — Демонстрація роботи бібліотеки nltk

```
Working => work
Work => work
Works => work
Worked => work
```

Рисунок 3.9 — Вивід основи слова

На рисунку 3.10 показано наступний крок в попередній обробці набору даних. Це витягування тексту, а якщо тексту в листі не було заміна його на слово empty. Після чого відкидання афіксів кожного слова в тексті.

```python
stemmer = nltk.PorterStemmer()


def process_email(email, stemming=True):
    text = email_to_plain(email)
    if text is None:
        text = 'empty'

    if stemming:
        text = ' '.join(map(stemmer.stem, text.split(' ')))

    return text
```

Рисунок 3.10 — Наступник крок обробки, видалення афіксів слів

Так як набір даних відносно малий, тобто вміщяється в оперативну пам’ять комп’ютера, то зберігатися оброблений набір даних буде в оперативній пам’яті. Але все одно, обробити три тисячі листів не швидка операція, тому треба використати декілька ядер процесора для обробки. Для цього використано клас multiprocessing.Pool, що автоматично розпаралелює map операцію на багато потоків і синхронізує їх.

```
import multiprocessing
pool = multiprocessing.Pool()

text_ham_emails = pool.map(process_email, ham_emails)
text_spam_emails = pool.map(process_email, spam_emails)
pool.close()
```

Рисунок 3.11 — Пералельна обробка елементів набору даних

На рисунку 3.12 показано як виглядає текст листа після фінального кроку обробки, тобто такий текст буде приймати модель (векторизація буде першим шаром моделі).

```
martin a posted:
tasso papadopoulos, the greek sculptor behind the plan, judg that the
 limeston of mount kerdylio, 70 mile east of salonika and not far from the
 mount atho monast community, wa ideal for the patriot sculpture.

 as well as alexander' granit features, 240 ft high and 170 ft wide, a
 museum, a restor amphitheatr and car park for admir crowd are
planned
---------------------
so is thi mountain limeston or granite?
if it' limestone, it'll weather pretti fast.

------------------------ yahoo! group sponsor ----------------------~-->
4 dvd free +s&p join now
http://us.click.yahoo.com/pt6ybb/nxieaa/mg3haa/7gsolb/tm
---------------------------------------------------------------------~->

to unsubscrib from thi group, send an email to:
forteana-unsubscribe@egroups.com



your use of yahoo! group is subject to http://docs.yahoo.com/info/terms/
```

Рисунок 3.12 — Текст листа після фінальної обробки

Keras приймає два масиви, з X та Y. Тобто першим масивом повинні бути листи, а другим спам чи ні. Так як спам і не спам листи в окремих масивах, треба їх об’єднати. Див рисунок 3.13, так як ми знаємо порядок об’єднання можна одразу згенерувати масив з нулей та одиниць. Нуль буде означати не спамовий лист, а одиниця спам. Так як спочатку в масиві йдуть усі не спам листи, то масиви треба перемішати.

```
features = np.array(text_ham_emails + text_spam_emails)
labels = np.array([0 for _ in range(len(text_ham_emails))] +
                  [1 for _ in range(len(text_spam_emails))])

indices = np.arange(features.shape[0])
np.random.shuffle(indices)

features = features[indices]
labels = labels[indices]
```

Рисунок 3.13 — Створення масивів для навчання

Майбутня модель буде обробляти стрічки символім (тип str), а для цього треба якимось чином перетворити символи або слова на цифри, бо модель може розуміти і навчатися тільки на цифрах.

Keras має клас шару TextVectorization який приймає текст та виводить дані у різних форматах в залежності від конфігурації. В даному випадку я використовую output_mode=`int` щоб кожне слово було цілочисленого типу, наприклад 1, 2, 40, 1337. Але просто вказати тип виводу не достатньо, треба обмежити максимальний розмір словника та розмір вихідного вектора. Ці два параметри визначені на перших двух стрічках. Розмір словника обмежено в десять тисяч слів, а максимальна довжина вихідного вектора тисяча. Від цих параметрів залежить кількість параметрів до тренування, тобто розмір мережі. На десятій стрічці виконується адаптація шару векторизації, це необхідний крок, без нього на стадії навчання буде помилка.

```
VOCAB_SIZE = 10_000
SEQUENCE_LENGTH = 1000

text_vectorization = keras.layers.TextVectorization(
    max_tokens=VOCAB_SIZE,
    output_mode='int',
    output_sequence_length=SEQUENCE_LENGTH,
)

text_vectorization.adapt(features)
```

Рисунок 3.14 — Шар векторизації тексту

3.8 Архітектура моделі

Після підготовки першого шару векторизації тексту, треба визначити архітектуру моделі, див рисунок 3.15. З допомогою класу layers.Input вказуємо розмір тензора та тип даних, післлля чого передаємо щойно створений шар у шар векторизації тексту. Такий спосіб завдання архітектури називається функціональним API в keras. Далі слідує шар

вставки який шукає зв’язки між словами в тексті, зазвичай це дуже сильно покращує роботу моделі, бо слова можуть мати різний контекст, але цей шар дуже великий, і його складніше тренувати. Майже увесь розмір моделі виходить з цього шару. Після чого можна використати різні підходи, зазвичай використовують рекуренті нейронні мережі або трансформер архітектуру, але у них є свої мінуси в тому що вони ресурсноємкі, і через це дуже довго навчаються. Альтернативою є згортковий шар у однопросторовому режимі (1D). Так як рекурентний шар збільшує розмірність даних, то треба якимось чином зменшити її. Це робить layers.GlobalMaxPool1D шар, на рисунку 3.16 можна побачити як після цього шару розмірність даних падає на 499. Завершальним шаром є один повнозв’язний тензор з функцією активації сігмоіда.

Компіляція моделі вимагає передати функцію помилки та опціонально метрики які будуть відображатися на екрані. У даному випадку треба використати бінарну точність, влучність та повноту.

```
def create_model():
    inputs = keras.layers.Input(shape=(1,), dtype="string")
    x = text_vectorization(inputs)
    x = keras.layers.Embedding(
        input_dim=VOCAB_SIZE, output_dim=256, mask_zero=True)(x)
    x = keras.layers.Conv1D(16, 4, activation="relu", strides=2)(x)
    x = keras.layers.GlobalMaxPool1D()(x)
    outputs = keras.layers.Dense(1, activation="sigmoid")(x)

    model = keras.models.Model(inputs=inputs, outputs=outputs)
    model.compile(
        optimizer=keras.optimizers.RMSprop(),
        loss=keras.losses.BinaryCrossentropy(),
        metrics=[
            keras.metrics.BinaryAccuracy(threshold=0.9),
            keras.metrics.Recall(name="recall"),
            keras.metrics.Precision(name="precision"),
        ]
    )
    return model


model = create_model()
model.summary()
```

Рисунок 3.15 — Архітектура моделі

З рисунку 3.16 видно що модель має два мільйони і п’ятьсот тисяч параметрів до тренування, це досить багато, адже чим більше параметрів тим більше часу треба на навчання. Але ще, чим більше параметрів тим гірше перенавчання.

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 1)]               0

 text_vectorization_1 (TextV  (None, 1000)             0
 ectorization)

 embedding (Embedding)       (None, 1000, 256)         2560000

 conv1d (Conv1D)             (None, 499, 16)           16400

 global_max_pooling1d (Globa  (None, 16)               0
 lMaxPooling1D)

 dense (Dense)               (None, 1)                 17

=================================================================
Total params: 2,576,417
Trainable params: 2,576,417
Non-trainable params: 0
_________________________________________________________________
```

Рисунок 3.16 — Вивід опису архітектури моделі

3.9 Валідація

Keras має зручну можливість виводити метрики під час тренування, а ще якщо передати дані для валідації, то метрики будуть обраховуватися і на них. Найпростіший спосіб валідації це розділення набору даних на дві частини, з деякою пропорційністю, наприклад 8 до 2. З наступного рисунку видно що 20% відсотків набору даних буде використано як дані для валідації, в такий спосіб можна порівняти метрики для даних яка бачила модель і метрики для даних яких модель ніколи не бачила.

```
split = 0.2
split_at = int(len(features) * 0.2)

train_features, train_labels = features[split_at:], labels[split_at:]
val_data = (features[:split_at], labels[:split_at])

model.fit(features, labels, validation_data=val_data, epochs=100)
```

Рисунок 3.17 — Код тренування

На рисунку 3.18 показано графік значень функції помилки від епохи навчання. Як можна побачити це значення дуже швидко сягає майже нульового значення, десь за сім епох.

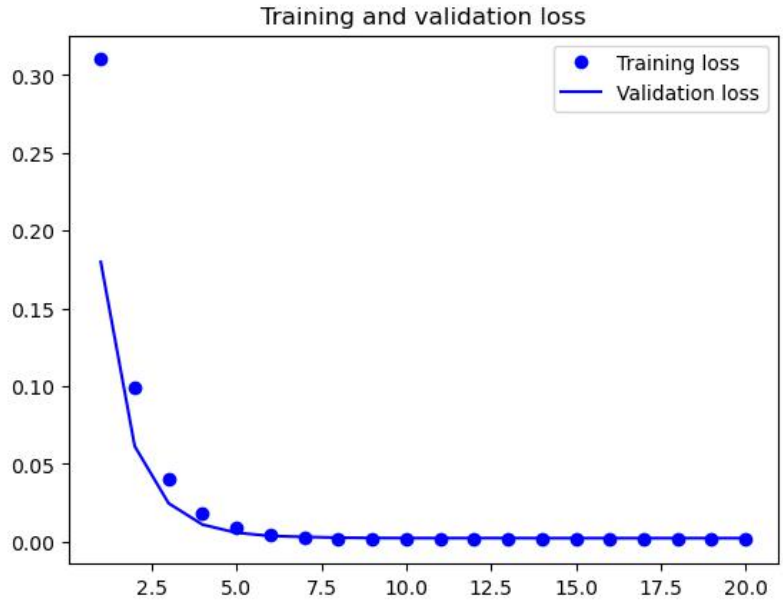

Рисунок 3.18 — Графік функції помилки

На рисунку 3.19 показано графік значень бінарної точності від епохи. Насправді метрика точності не дуже інформативна, якщо в бінарному наборі даних із тисячі елементів десять однієї категорії, а всі інші іншої то модель просто може передбачувати одну категорію увесь час, не вчась як узагальнювати дані. Модель вчиться патернам із набору даних, а головна задача підібрати правильну архітектуру та гіперпараметри.

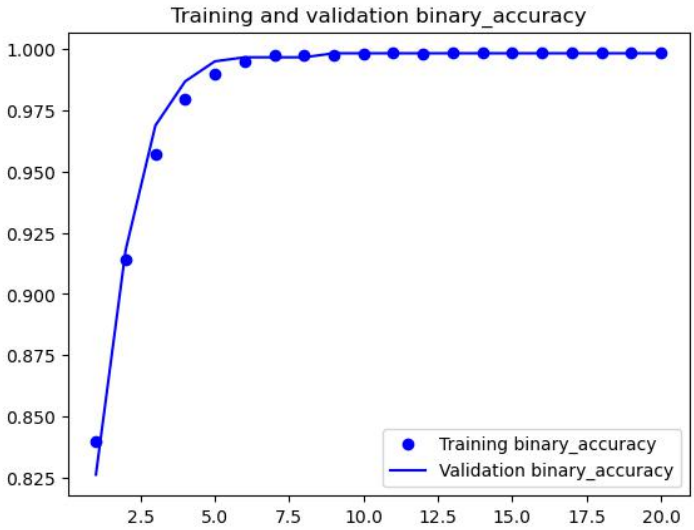

Рисунок 3.19 — Графік точності

На рисунку 3.20 показано графік повноти. Він показує співвідношення знайдених спам листів до усіх спам листів. З графіку видно що модель знаходить усі спам повідомлення, співвідношення дорівнює одиниці.

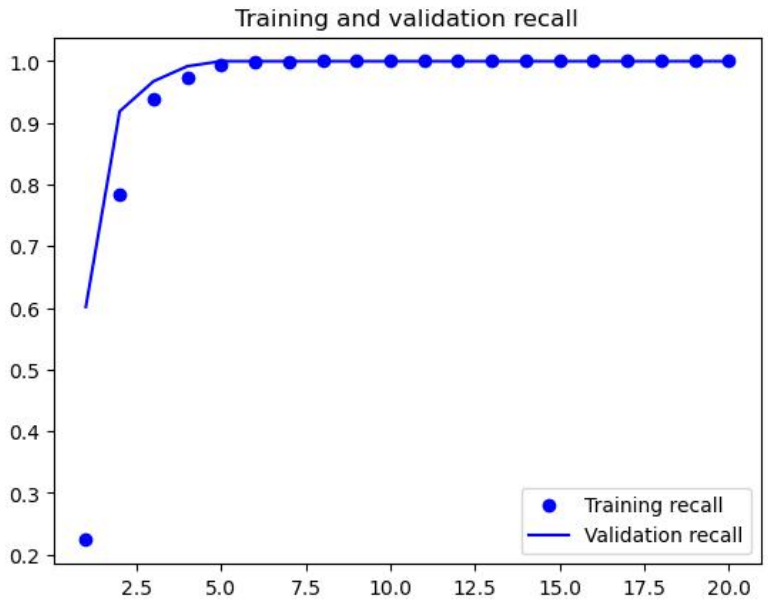

Рисунок 3.20 — Графік повноти

На рисунку 3.X показано графік влучності. Якщо влучність дорівнює одиниці це значить що модель не допускає помилок. В даному випадку влучність майже одиниця.

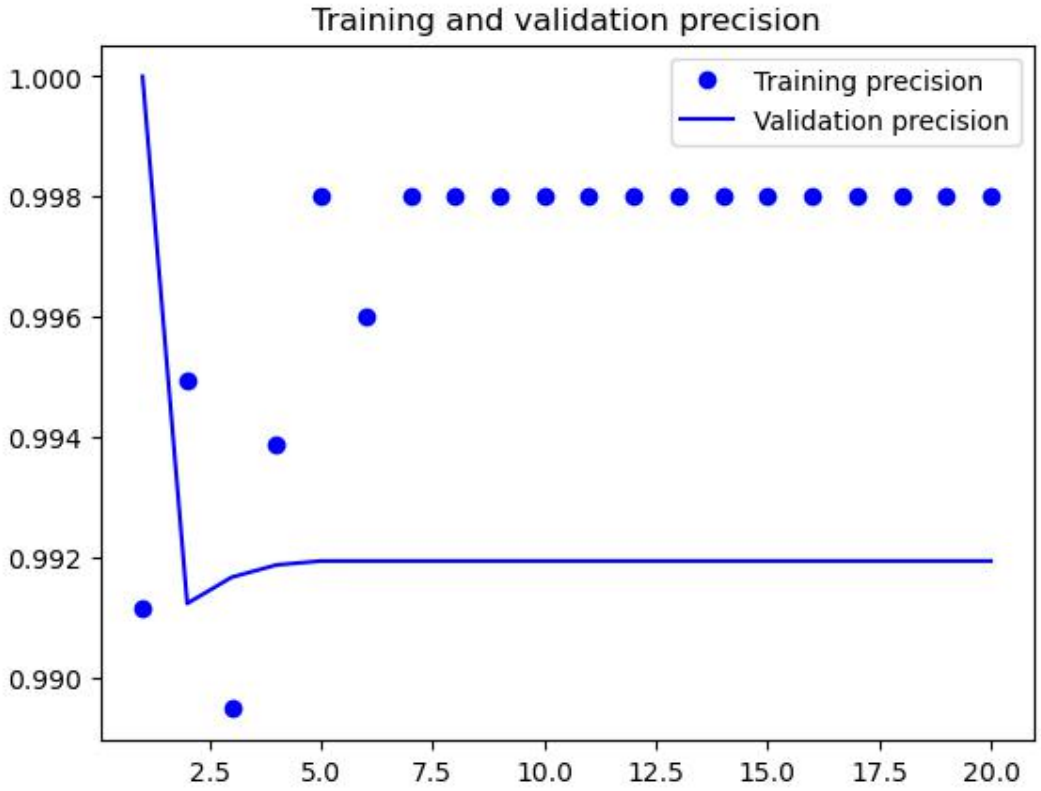

Рисунок 3.21 — Графік влучності

3.10 K-fold cross validation

При маленькому наборі даних розділення набору даних на частини може погіршити навчання. Тому в таким випадках використовують K-fold крос валідацію.

K-Fold — це техніка перевірки, за якої дані розбиваються на k-підмножини, а навчання повторюється k-разів, де кожна з k-підмножин використовується як тестовий набір, а інші k-1 підмножини використовуються з метою навчання. Потім обчислюється середня помилка з усіх цих k проб, що є більш надійним порівняно зі стандартним методом навчання.

Тож із цим підходом не потрібно піклуватися про те, як розподіляються дані.

На рисунку 3.22 показано код імплементації крос валідації. Перемішування та розділення набору даних проходить з допомогою бібліотеки sklearn.

```
from sklearn.model_selection import KFold

acc_per_fold = []
loss_per_fold = []

kfold = KFold(n_splits=5, shuffle=True)
fold_no = 1

for train, test in kfold.split(features, labels):
    model = create_model()

    print('------------------------------------------------------------------------')
    print(f'Training for fold {fold_no} ...')

    history = model.fit(
        features[train], labels[train],
        batch_size=32,
        epochs=10,
        verbose=0,
    )

    scores = model.evaluate(features[test], labels[test], verbose=0)
    print(f'Score for fold {fold_no}:')
    for i, metric in enumerate(model.metrics_names):
        print(f"\t{metric:20} of {scores[i]:.3f}")
    acc_per_fold.append(scores[1] * 100)
    loss_per_fold.append(scores[0])

    fold_no = fold_no + 1
```

Рисунок 3.22 — K-fold валідація з допомогою бібліотеки sklearn

З вихідних даних кросвалідації видно як змінюються метрики в залежності від фолду. По показникам влучності та повноти кращим є перший фолд.

```
------------------------------------
Training for fold 1 ...
Score for fold 1:
    loss                 of 0.031
    binary_accuracy      of 0.977
    recall               of 0.990
    precision            of 0.962
------------------------------------
Training for fold 2 ...
Score for fold 2:
    loss                 of 0.022
    binary_accuracy      of 0.987
    recall               of 0.947
    precision            of 1.000
------------------------------------
Training for fold 3 ...
Score for fold 3:
    loss                 of 0.039
    binary_accuracy      of 0.975
    recall               of 0.943
    precision            of 0.980
------------------------------------
Training for fold 4 ...
Score for fold 4:
    loss                 of 0.039
    binary_accuracy      of 0.979
    recall               of 0.948
    precision            of 0.968
```

Рисунок 3.23 — Статистика кожного фолду кросвалідації

3.11 Альтернативні архітектури моделі

3.11.1 Повнозв’язні шари

Моделі без прихованих шарів не дуже розповсюджені, бо вони не мають великої здатності сприйняття глибоких кореляцій в даних.

```python
def create_model():
    inputs = keras.layers.Input(shape=(1,), dtype="string")
    x = text_vectorization(inputs)
    outputs = keras.layers.Dense(1, activation="sigmoid")(x)

    model = keras.models.Model(inputs=inputs, outputs=outputs)
    model.compile(
        optimizer=keras.optimizers.RMSprop(),
        loss=keras.losses.BinaryCrossentropy(),
        metrics=[
            keras.metrics.BinaryAccuracy(),
            keras.metrics.Recall(name="recall"),
            keras.metrics.Precision(name="precision"),
        ]
    )
    return model


model = create_model()
model.summary()
```

Рисунок 3.24 — Створення моделі без прихованих шарів

Як видно з рисунку 3.25, модель має всього тисячу параметрів для навчання, але як буде видно, для цього набору даних такої кількості вистачає щоб досягнути неочікуваних результатів.

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input (InputLayer)          [(None, 1)]               0

 text_vectorization (TextVec  (None, 1000)             0
 torization)

 output (Dense)              (None, 1)                 1001

=================================================================
Total params: 1,001
Trainable params: 1,001
Non-trainable params: 0
_________________________________________________________________
```

Рисунок 3.25 — Вивід інформації про архітектуру без прихованих шарів

На рисунку 3.26 показано графік точності цієї моделі. Неочікувано але цей набір даних має такі властивості що визначення спамового листа дуже легке. Це пов’язано з тим що спам листи з набору даних мають унікальні слова, які в звичайних листах з’являються дуже рідко.

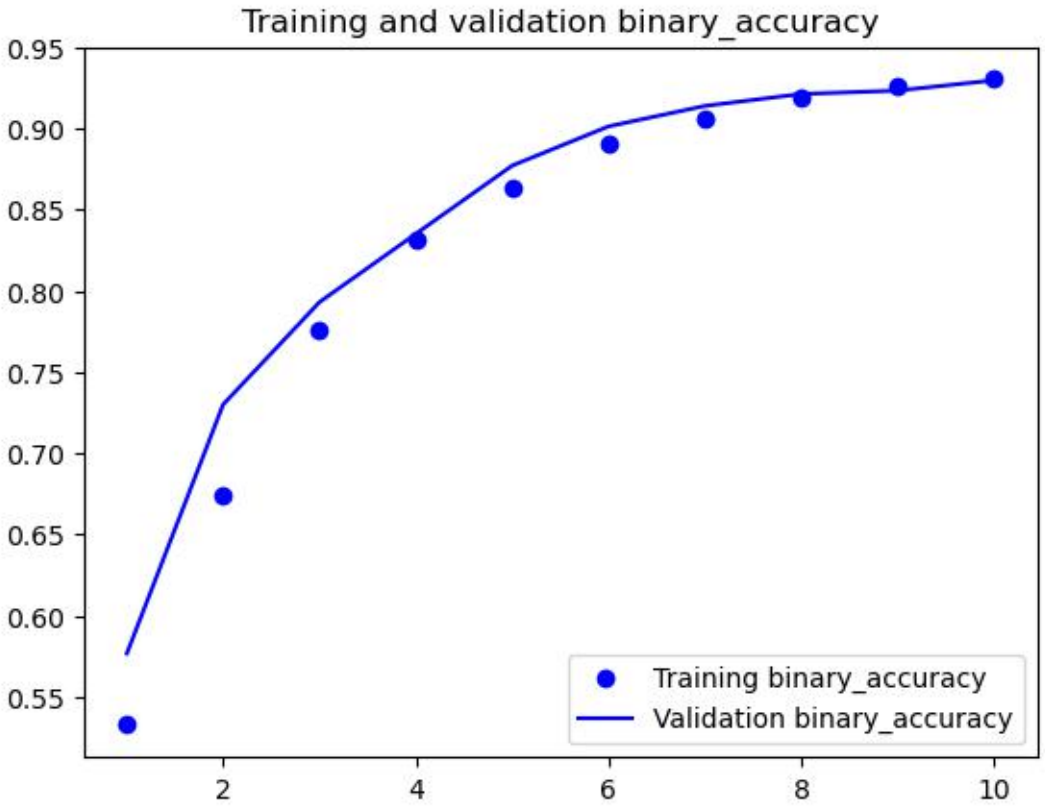

Рисунок 3.26 — Графік точності моделі

3.11.2 TF-IDF

TF-IDF (частота слова - обернена частота документа) — це статистичний показник, який оцінює релевантність слова документу в колекції документів.

Це робиться шляхом множення двох показників: скільки разів слово з'являється в документі та зворотної частоти цього слова в наборі документів.

$$TF = \frac{n_i}{\sum_k n_k} \tag{3.1}$$

$$IDF = \log \frac{|D|}{|(d_i \supset t_i)|} \tag{3.2}$$

$$TF\ IDF = TF \cdot IDF \tag{3.3}$$

Він має багато застосувань, особливо в автоматизованому аналізі тексту, і дуже корисний для підрахунку слів у алгоритмах машинного навчання для обробки природної мови.

Щоб використати TF-IDF треба змінити output_mode параметр шару векторизації. Архітектура моделі залишається як в минулому прикладі, без прихованих шарів.

```
VOCAB_SIZE = 1000

text_vectorization = keras.layers.TextVectorization(
    max_tokens=VOCAB_SIZE,
    output_mode='tf_idf',
    name="text_vectorization",
)

text_vectorization.adapt(features)
```

Рисунок 3.27 — Налаштування шару векторизації тексту

Якщо в минулому прикладі точність наближалася до 93% то з TF-IDF точність сягає 97% та ту саму кількість епох навчання. Це говорить про те що в різних категоріях є сильна кореляція слів. Якщо прибрати розповсюджені слова з обох категорій то спам листи будуть мати унікальні слова за якими легше визначити чи спам цей лист.

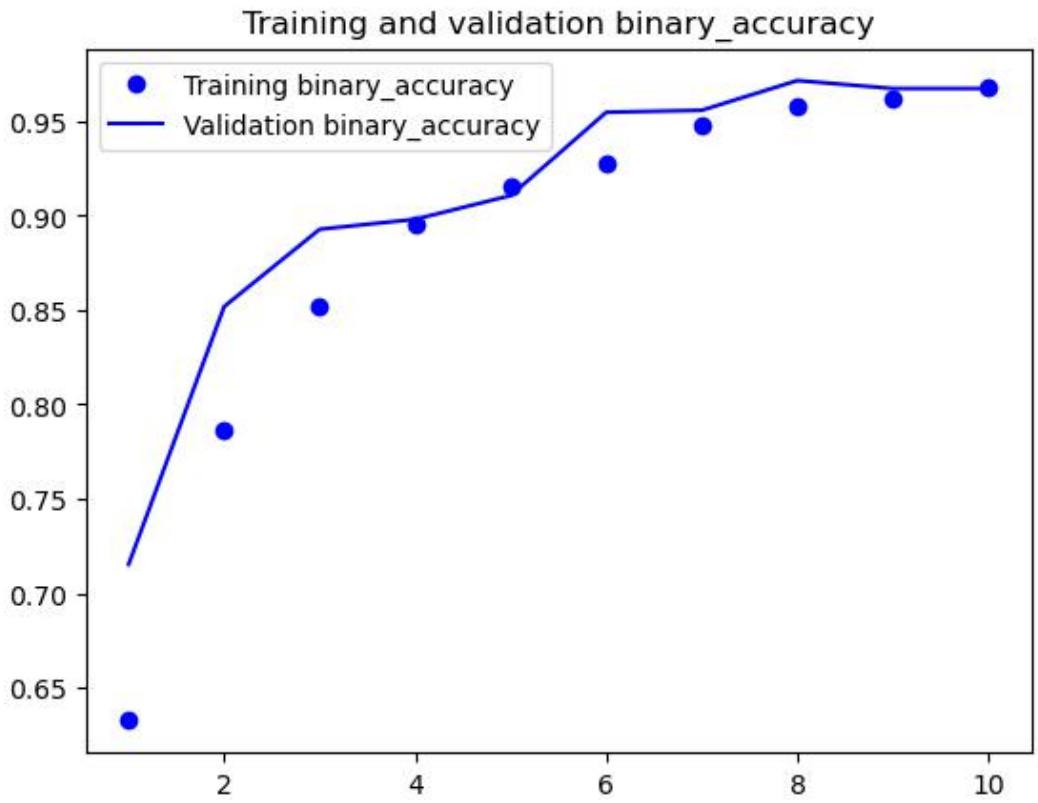

Рисунок 3.28 — Графік точності

Висновки

У першому розділі було розглянуто теоретичні аспекти машинного навчання та його підкатегорію глибоке навчання. Машинне навчання — це галузь дослідження, присвячена розумінню та розробці методів, які «навчаються», тобто методів, які використовують дані для підвищення ефективності певного набору завдань. Його розглядають як частину штучного інтелекту. Алгоритми машинного навчання створюють модель на основі вибіркових даних, відомих як навчальні дані, щоб робити прогнози чи приймати рішення без явного програмування. Слово глибина у глибокому навчанні відноситься до використання кількох рівнів у мережі. Сучасне глибоке навчання пов’язане з необмеженою кількістю шарів обмеженого розміру, що дозволяє практичне застосування та оптимізоване впровадження, зберігаючи теоретичну універсальність.

У другому розділі оглянуто сучасні інструменти розробки глибокого навчання. На даний момент найпопулярнішим способом є написання коду на мові програмування Python у середовищі Jupyter Notebooks. Це дозволяє мати швидкі ітерації адже це середовище дуже схоже на REPL цикл. За роки Python акомулював багато бібліотек для машинного навчання, одна з найпопулярніших це Keras. Keras базується на Tensorflow, і за замовчуванням є високорівневим, хоча при бажанні він надає доступ до низькорівневих компонентів. Для менеджменту бібліотек Python існує програма Anaconda, яка постачає пакети в легкій для установці формі. Також було розглянуто декілька прикладів написання коду для вирішення проблеми XOR і класифікації зображень.

Третій розділ присвячено вирішенню проблеми бінарної класифікації електроних спам листів. Розглянуто типи спам листів і набір даних для навчання.

Важливість підготовки та обробки набора даних не треба недооцінювати, зазвичай створення набору даних та його правильна обробка це найскладніша частина будь якого проєкту машинного навчання. Далі було проведено валідацію отриманих результатів двома методами: методом затримки даних та K-fold крос валідації. Останнім було розглянуто альтернативні архітектури моделі, серед яких: архітектура без прихованих шарів та архітектура на базі TF-IDF обробки набору даних.

Список використаних джерел

1. Lee, J., Bahri, Y., Novak, R., Schoenholz, S., Pennington, J., & Sohl-Dickstein, J.. (2017). Deep Neural Networks as Gaussian Processes.

2. What is gradient descent? IBM. (2020, October 27). Retrieved November 20, 2022, from https://www.ibm.com/cloud/learn/gradient-descent

3. Al-Masri, A. (2019, January 30). How does back-propagation in Artificial Neural Networks Work? Retrieved November 20, 2022, from https://towardsdatascience.com/how-does-back-propagation-in-artificial-neural-networks-work-c7cad873ea7

4. ICDAR 2011 : 12th International Conference on Document Analysis and Recognition. ICDAR 2011 : 12th International Conference on Document Analysis And recognition. (2011, September 18). Retrieved November 20, 2022, from http://www.wikicfp.com/cfp/servlet/event.showcfp?eventid=6641©ownerid=2

5. IJCNN 2011 : International Joint Conference on Neural Networks. (2011, July 31). Retrieved November 20, 2022, from http://www.wikicfp.com/cfp/servlet/event.showcfp?eventid=14568

6. Using Deep Learning Models / Convolutional Neural Networks. Using deep learning models / convolutional Neural Networks. (n.d.). Retrieved November 20, 2022, from https://docs.ecognition.com/eCognition_documentation/User%20Guide%20Developer/8%20Classification%20-%20Deep%20Learning.htm

7. Stross-Radschinski, A. C. (2019, January 30). The python brochure vol.1. Get the Python Brochure Vol.1 as download! -. Retrieved November 20, 2022, from https://brochure.getpython.info/

8. Tensorflow Core: Machine Learning for beginners and experts. TensorFlow. (n.d.). Retrieved November 20, 2022, from https://www.tensorflow.org/overview

9. Marcondes, D., Simonis, A., & Barrera, J. (2022). The role of prior information and computational power in Machine Learning. arXiv preprint arXiv:2211.01972.

10. Shahariar, G. M., Biswas, S., Omar, F., Shah, F. M., & Hassan, S. B. (2019, October). Spam review detection using deep learning. In 2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON) (pp. 0027-0033). IEEE.

11. Kostas, K., Kostas, R. Y., Zampella, F., & Alsehly, F. (2022, September). WiFi Based Distance Estimation Using Supervised Machine Learning. In 2022 IEEE 12th International Conference on Indoor Positioning and Indoor Navigation (IPIN) (pp. 1-8). IEEE.

12. Lee, J. H., Haghighatshoar, S., & Karbasi, A. (2022). Exact Gradient Computation for Spiking Neural Networks Through Forward Propagation. arXiv preprint arXiv:2210.15415.

13. Wang, Z., & Sirignano, J. (2022). A Forward Propagation Algorithm for Online Optimization of Nonlinear Stochastic Differential Equations. arXiv preprint arXiv:2207.04496.

14. A guide to deep learning layers. ADG Efficiency. (2020, November 16). Retrieved November 20, 2022, from https://adgefficiency.com/guide-deep-learning/

15. Urwin, M. (2022, October 5). Gradient descent in Machine Learning: A basic introduction. Built In. Retrieved November 20, 2022, from https://builtin.com/data-science/gradient-descent

16. Voss, C., Cammarata, N., Goh, G., Petrov, M., Schubert, L., Egan, B., Lim, S. K., & Olah, C. (2021, April 8). Visualizing weights. Distill.

Retrieved November 20, 2022, from https://distill.pub/2020/circuits/visualizing-weights/

17. Millidge, B., Salvatori, T., Song, Y., Bogacz, R., & Lukasiewicz, T. (2022). Predictive Coding: Towards a Future of Deep Learning beyond Backpropagation?. arXiv preprint arXiv:2202.09467.

18. Baldi, P., Sadowski, P., & Lu, Z. (2018). Learning in the machine: Random backpropagation and the deep learning channel. Artificial intelligence, 260, 1-35.

19. Innamorati, C., Ritschel, T., Weyrich, T., & Mitra, N. J. (2019, October 8). Learning on the edge: Investigating boundary filters in CNNS - International Journal of Computer Vision. SpringerLink. Retrieved November 20, 2022, from https://link.springer.com/article/10.1007/s11263-019-01223-y

20. Alto, V. (2020, July 21). Border effects and transition steps in Convolutional Neural Networks. Medium. Retrieved November 20, 2022, from https://medium.com/dataseries/border-effects-and-transition-steps-in-convolutional-neural-networks-75bc6a44fd43

21. Hu, B., Sun, Y., & Qin, A. K. (2022). A General Multiple Data Augmentation Based Framework for Training Deep Neural Networks. arXiv preprint arXiv:2205.14606.

22. Metrics to use to evaluate deep learning object detectors. KDnuggets. (n.d.). Retrieved November 20, 2022, from https://www.kdnuggets.com/2020/08/metrics-evaluate-deep-learning-object-detectors.html

23. https://www.radicati.com/wp/wp-content/uploads/2020/12/Email-Statistics-Report-2021-2025-Executive-Summary.pdf

Додатки

Додаток А — Повний код програми

```
import numpy as np
import multiprocessing
import nltk
from tensorflow import keras
import tensorflow as tf
import pathlib
import os
import email
from bs4 import BeautifulSoup
from email import policy

dataset_path = pathlib.Path("datasets/email-spam/")


def parse_email(path, is_spam):
    subfolder = "ham"
    if is_spam:
        subfolder = "spam"

    with open(os.path.join(dataset_path/subfolder,
path), 'rb') as f:
```

```
        return
email.parser.BytesParser(policy=policy.default).parse
(f)


ham_emails = [parse_email(path, False)
              for path in
sorted(os.listdir(dataset_path/"ham"))]
spam_emails = [parse_email(path, True)
               for path in
sorted(os.listdir(dataset_path/"spam"))]

print('Amount of ham files:', len(ham_emails))
print('Amount of spam files:', len(spam_emails))
print('Spam to Ham Ratio:',
len(spam_emails)/len(ham_emails))

testEmail = ham_emails[0]
print('Header Field Names:', testEmail.keys())
print('\n\n')
print('Message Field Values:', testEmail.values())
print('\n\n')
print('Message Content:', testEmail.get_content())

def html_to_plain(email):
    try:
```

```
        soup = BeautifulSoup(email.get_content(),
'html.parser')
        return soup.text.replace('\n\n', '')
    except:
        return "empty"

def email_to_plain(email):
    for part in email.walk():
        partContentType = part.get_content_type()
        if partContentType not in ['text/plain',
'text/html']:
            continue
        try:
            partContent = part.get_content()
        except:  # in case of encoding issues
            partContent = str(part.get_payload())
        if partContentType == 'text/plain':
            return partContent
        else:
            return html_to_plain(part)


print(email_to_plain(ham_emails[0]))

stemmer = nltk.PorterStemmer()
for word in ("Working", "Work", "Works", "Worked"):
```

```
    print(word, "=>", stemmer.stem(word))

stemmer = nltk.PorterStemmer()
stemming = False

def process_email(email):
    text = email_to_plain(email)
    if text is None:
        text = 'empty'

    if stemming:
        # nltk.word_tokenize(text))
        text = ' '.join(map(stemmer.stem,
text.split(' ')))

    return text

pool = multiprocessing.Pool()

text_ham_emails = pool.map(process_email, ham_emails)
text_spam_emails = pool.map(process_email,
spam_emails)
pool.close()
print(text_ham_emails[1])
```

```
features = np.array(text_ham_emails +
text_spam_emails)
labels = np.array([0 for _ in
range(len(text_ham_emails))] +
                  [1 for _ in
range(len(text_spam_emails))])

indices = np.arange(features.shape[0])
np.random.shuffle(indices)

features = features[indices]
labels = labels[indices]

VOCAB_SIZE = 1000
SEQUENCE_LENGTH = 500

text_vectorization = keras.layers.TextVectorization(
    max_tokens=VOCAB_SIZE,
    output_mode='int',
    output_sequence_length=SEQUENCE_LENGTH,
    name="text_vectorization",
)

text_vectorization.adapt(features)

def create_model():
```

```
    inputs = keras.layers.Input(shape=(1,),
dtype="string", name="input")
    x = text_vectorization(inputs)
    x = keras.layers.Embedding(
        input_dim=VOCAB_SIZE, output_dim=256,
mask_zero=True)(x)
    x = keras.layers.Conv1D(16, 4, activation="relu",
strides=2)(x)
    x = keras.layers.GlobalMaxPool1D()(x)
    outputs = keras.layers.Dense(1,
activation="sigmoid", name="output")(x)

    model = keras.models.Model(inputs=inputs,
outputs=outputs, name="model")
    model.compile(
        optimizer=keras.optimizers.RMSprop(),
        loss=keras.losses.BinaryCrossentropy(),
        metrics=[

keras.metrics.BinaryAccuracy(threshold=0.7),
            keras.metrics.Recall(name="recall"),
            keras.metrics.Precision(name="precision"),
        ]
    )
    return model
```

```
model = create_model()
model.summary()

split = 0.2
split_at = int(len(features) * 0.2)

train_features, train_labels = features[split_at:],
labels[split_at:]
val_data = (features[:split_at], labels[:split_at])

history = model.fit(features, labels,
validation_data=val_data, epochs=10)



import utils
for metric in ["binary_accuracy", "loss", "recall",
"precision"]:
    utils.plot_history(history, metric_name=metric)
utils.show_plot()



from sklearn.model_selection import KFold

acc_per_fold = []
loss_per_fold = []
```

```
kfold = KFold(n_splits=5, shuffle=True)
fold_no = 1

for train, test in kfold.split(features, labels):
    model = create_model()

    # Generate a print
    print('------------------------------------------
------------------------------')
    print(f'Training for fold {fold_no} ...')

    # Fit data to model
    history = model.fit(
        features[train], labels[train],
        batch_size=32,
        epochs=10,
        verbose=0,
    )

    # Generate generalization metrics
    scores = model.evaluate(features[test],
labels[test], verbose=0)
    print(f'Score for fold {fold_no}:')
    for i, metric in enumerate(model.metrics_names):
        print(f"\t{metric:20} of {scores[i]:.3f}")
```

```
    acc_per_fold.append(scores[1] * 100)
    loss_per_fold.append(scores[0])

    # Increase fold number
    fold_no = fold_no + 1



print('------------------------------------------------
--------------------------')
print('Score per fold')
for i in range(0, len(acc_per_fold)):
    print('------------------------------------------
------------------------------')
    print(
        f'> Fold {i+1} - Loss:
{loss_per_fold[i]:2.03f} - Accuracy:
{acc_per_fold[i]:2.02f}%')
print('------------------------------------------------
--------------------------')
print('Average scores for all folds:')
print(f'> Accuracy: {np.mean(acc_per_fold)} (+-
{np.std(acc_per_fold)})')
print(f'> Loss: {np.mean(loss_per_fold)}')
print('------------------------------------------------
--------------------------')
```