Державний вищий навчальний заклад
«Прикарпатський національний університет імені Василя Стефаника»
Фізико-технічний факультет
Кафедра комп'ютерної інженерії та електроніки

Соломовський Роман Володимирович
Solomovskyi Roman

УДК 004:032.26

Спеціальність 123 «комп'ютерна інженерія»

Кваліфікаційна робота
на здобуття освітнього ступеня бакалавр

Аналіз і обробка текстових даних за допомогою нейронних мереж
Analyzing and processing text data using neural networks

Науковий керівник:
Проф. Запухляк Р.І.
Рецензент:
Доц. Соломко А.В.

Івано-Франківськ
2020

Формат	Поз.	Позначення	Найменування	К-ть	Прим.
А4			Пояснювальна записка	59	
А4			Додаток А	4	

Змн.	Арк.	№ докум.	Підпис	Дата	123.КІ-41.26		
Розробив		Соломовський Р.В.			Специфікація	Літ.	Арк. 2 / Аркушів 1
Перевірив		Запухляк Р.І					
Н. Контр.							
Затвердив							

АНОТАЦІЯ

У дипломній роботі розглянуто способи проектування та використання штучних нейронних мереж для аналізу та обробки текстових даних.

В результаті роботи було розроблено штучну нейронну мережу для тональної класифікації текстових даних на основі декількох типів нейронних мереж. Розробка здійснювалась мовою програмування Python з використанням бібліотек для обробки природних мов та бібліотек для проектування і навчання нейронних мереж. Було протестовано та показано переваги та недоліки тієї чи іншої мережі у даній проблематиці.

Ключові слова: штучні нейронні мережі, обробка природної мови, машинне навчання.

Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*			
Розробив		Соломовський Р.В.			Анотація	*Літ.*	*Арк.* 3 / *Аркушів* 1
Перевірив		Запухляк Р.І.					
Н. Контр.							
Затвердив							

ABSTRACT

The bachelor's work deals with the ways of designing and using artificial neural networks for analysis and processing of textual data.

As a result, an artificial neural network was developed to tone textual classification based on several types of neural networks. The development was done in Python programming language using natural language processing libraries and neural network design and training libraries. The advantages and disadvantages of a particular network have been tested and demonstrated.

Keywords: artificial neural networks, natural language processing, machine learning.

Міністерство освіти і науки України
Державний вищий навчальний заклад
«Прикарпатський національний університет імені Василя Стефаника»
Фізико-технічний факультет
Кафедра «Комп’ютерної інженерії та електроніки»

Пояснювальна записка
до кваліфікаційної роботи на тему:

Аналіз і обробка текстових даних за допомогою нейронних мереж

					123.КІ-41.26		
Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*			
Розробив		Соломовський Р.В.			Пояснювальна записка	*Літ.*	*Арк.* 5 / *Аркушів* 59
Перевірив		Запухляк Р.І.					
Н. Контр.							
Затвердив							

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ШНМ – Штучні Нейронні Мережі;

ШІ – Штучний Інтелект;

API – Application Programming Interface

ANN – Artificial Neural Network;

CNN – Convolutional neural network;

CV – Computer Vision;

DL – Deep Learning;

DNN – Deep Neural Network;

RNN – Recurrent Neural Network;

NN – Neural Network;

NLP – Natural Language Processing;

ML – Machine Learning;

MLP – Multilayer Perceptron;

ВСТУП

Кількість даних зростає щоденно, більшість із них не структуровані, а значну частину займає інформація виражена природною мовою (природна мова являє собою сукупність певних звуків та символів загальноприйнятих у певному суспільстві, за допомогою яких люди виражають свої думки). Станом на 2020 рік людством створено 50.5 екзабайт даних (1 екзабайт = 2^{70} байт). Аналітики прогнозують, що до 2025 року загальна кількість даних зросте до 175 екзабайт. Постійне зростання кількості та складності даних спонукає розробляти інструменти та проводити дослідження у цій галузі, адже потенційно можна тримати значну кількість корисної інформації аналізуючи такі дані. Однак через високу складність аналізу та структурування таких даних їх зазвичай ніяк і ніде не використовують.

Актуальність теми на даний момент, як ніколи велика, і з плином часу тільки ростиме. Ще з перших версій електронно-обчислювальних машин, програмісти намагались навчити комп'ютер розуміти природну мову. Причина досить зрозуміла – за тисячі років люди згенерували таку велику кількість інформації такого типу, що не звертати увагу просто не можливо. На жаль комп'ютери не можуть в певній мірі розуміти природну мову так, як це роблять люди, але їх можливості значно розширились за останні два десятки років, з розвитком обробки природної мови (Natural Language Processing, NLP). NLP – це одна із галузей штучного інтелекту, яка займається аналізом та синтезом природної мови. Останні розробки у сфері NLP доступні через відкриті бібліотеки spaCy та NLTK, на багатьох мовах програмування, зокрема Python.

В даній роботі розглянуто: способи проектування та використання нейронних мереж, методи опрацювання природної мови.

Мета даної бакалаврської роботи - розробка нейронної мережі для аналізу текстових даних, з використанням бібліотек проектування та навчання нейронних мереж: TensorFlow та Keras та бібліотек NLP: SpaCy, Text Blob.

РОЗДІЛ 1. ЗАГАЛЬНА ІНФОРМАЦІЯ ТА АНАЛІЗ ПІДХОДІВ ДО ПРОЕКТУВАННЯ І НАВЧАННЯ ШТУЧНИХ НЕЙРОННИХ МЕРЕЖ

Нейронні мережі - це комп'ютерні програми, що імітують функції мозку. Вони інтерпретують вхідні дані через своєрідне машинне сприйняття, маркування або групування даних. Шаблони, які вони розпізнають, є чисельними та містяться у векторах, у які повинні бути переведені всі вхідні дані, а саме зображення, звук, текст, щоб мережа могла їх обробити.

Штучна нейронна мережа складається з штучних нейронів, які активуються певною функцією активації та організована в три взаємопов'язані шари: вхідний, прихований, який може включати більше одного шару, і вихідний.

Особливість нейронних мереж полягає в тому, що знання про її домен поширюються по всій мережі, а не чітко записуються в програму. Ці знання моделюються як зв'язки між елементами обробки (штучними нейронами) та адаптивними вагами кожного з цих з'єднань. Потім мережа вчиться через вплив різних ситуацій. Нейронні мережі здатні досягти цього шляхом регулювання ваги зв’язків між сполученими нейронами, згрупованими в шари.

Вхідний шар штучних нейронів отримує інформацію з навколишнього середовища, а вихідний шар передає відповідь; між цими шарами може бути один або кілька прихованих шарів (де немає прямого контакту з навколишнім середовищем), де відбувається обробка більшості інформації.

Вихід нейронної мережі залежить від ваги зв’язків між нейронами в різних шарах. Кожна вага вказує на відносну важливість конкретного з'єднання. Якщо загальна сума всіх зважених входів, отриманих певним нейроном, перевищує певне порогове значення, нейрон посилатиме сигнал кожному нейрону, до якого він підключений у наступному шарі.

1.1 Виникнення штучних нейронних мереж

Ідея про те, як нейрони (рис. 1.1) мозку можуть працювати була опублікована у 1943 році нейрофізіологом Уореном Мак-Каллохом та математиком Уолтером Пітсом. Вченим вдалося змоделювати просту модель штучної нейронної мережі (ШНМ) за допомогою електричних кіл. Вони використовували комбінацію алгоритмів та математики, яку вони називали “пороговою логікою”, щоб імітувати процес мислення [1]. Робота призвела до поділу досліджень нейронних мереж на два підходи: дослідження біологічних процесів мозку та застосування нейронних мереж у галузі штучного інтелекту.

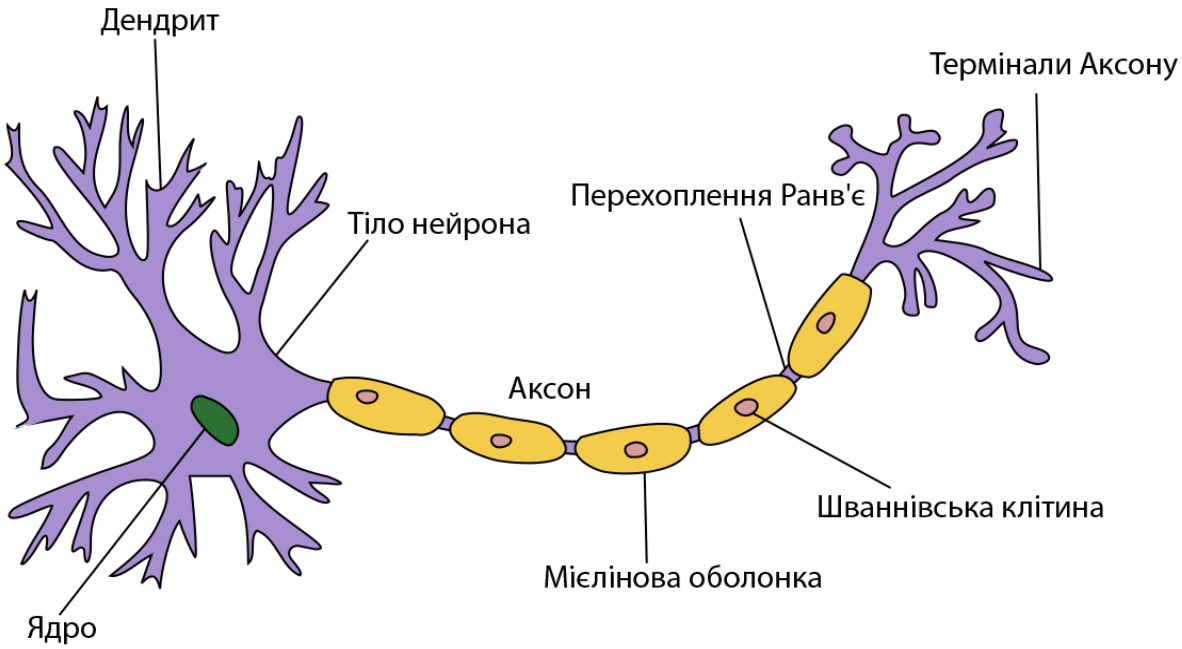

Рис. 1.1. Структура нейрона.

Наступним важливим кроком у 1949 році стала праця Дональда Гебба “Організація поведінки”, у якій вдалося показати, що нервові шляхи зміцнюються кожен раз, коли вони використовуються. Відкриття вказує на фундаментальні властивості, які важливі, як і для навчання людей, так і способів навчання штучних нейронних мереж [2]. Гебб першим запропонував працюючий алгоритм для навчання штучних нейронних мереж.

У 1957 році Джон фон Нейман запропонував імітувати прості функції нейронів за допомогою телеграфних реле або вакуумних трубок.

В зв'язку з швидким розвитком комп'ютерів та розвитком штучних нейронів (рис. 1.2) у 1959 році вченим зі Стенфорда вдалося розробити дві моделі, що називались “ADALINE” та “MADALINE”. Модель “ADALINE” була спроектована для обробки та розпізнавання бінарних шаблонів потокових бітів телефонної лінії з метою передбачення наступного біту інформації. У свою чергу “MADALINE” застосовували для усування відлуння в телефонних лініях, що стало першим застосуванням штучної нейронної мережі яка б допомагала вирішити реальні проблеми світу.

Того ж року Френк Розенблат на основі перцептрона (рис. 1.2) створив один із перших у світі нейрокомп'ютерів “Марк-1”, що був здатен розпізнати декілька літер англійської абетки. Математична модель перцептрона запропонована ним у 1957 році по праву може вважатися однією із перших ШНМ.

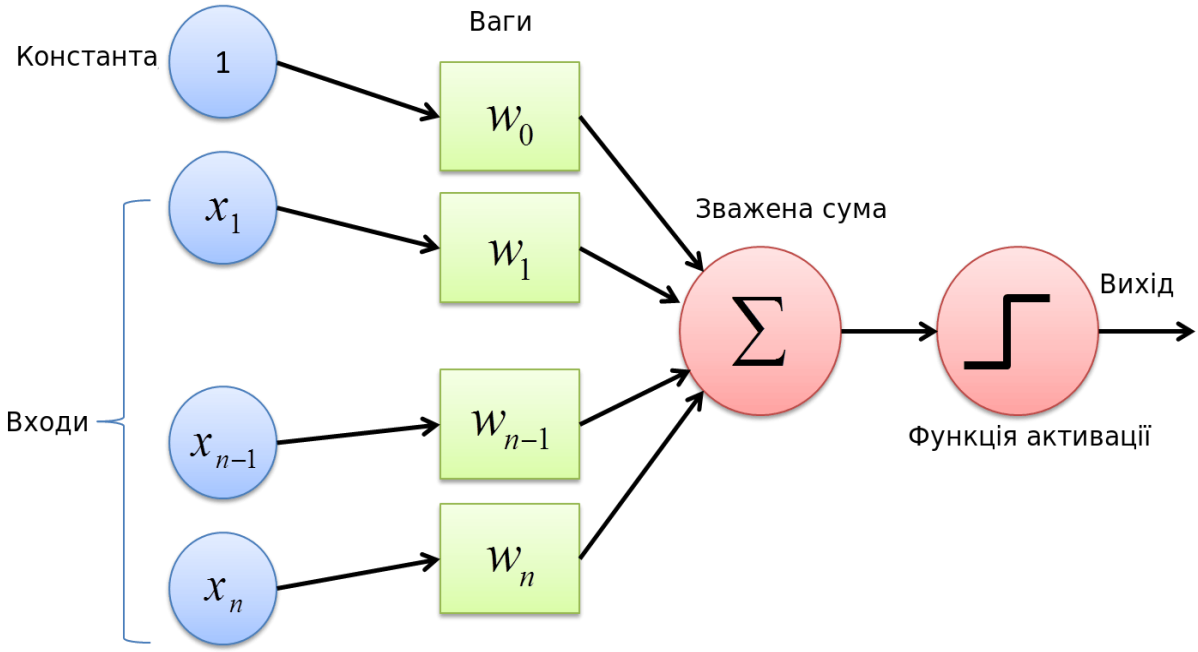

Рис. 1.2. Структура перцептрона.

Перші вдалі експерименти з перцептроном сприяли залученню до досліджень великої кількості науковців та надзвичайно оптимістичних прогнозів, щодо можливості створення інтелектуальних систем, що значно перевершують людський мозок.

У 1969 році виходить книга “Перцептрон” за авторством Марвіна Мінського та Сеймура Пейперта, робота присвячена аналізу проблем та обмеженості перцептрона. Автори у результаті роботи прийшли до висновку, що подальші дослідження у цьому напрямку безперспективні [3]. Висока складність проектування та використання, в сумі із малою ефективністю та критика багатьох подібних систем призвели до значного скорочення фінансування досліджень у даному напрямку. В результаті розвиток штучних нейронних мереж був заморожений більш ніж на 10 років.

У 1972 році Теуво Кохонен запропонував новий тип нейронних мереж, здатних функціонувати як пам'ять, також відомих, як самоорганізаційні Карти Кохонена (рисунок 1.3). Зокрема, Кохонен працював над розробкою фундаментальної теорії асоціативної пам'яті та запропонував спосіб навчання ШНМ з учителем для мереж векторного квантування [4].

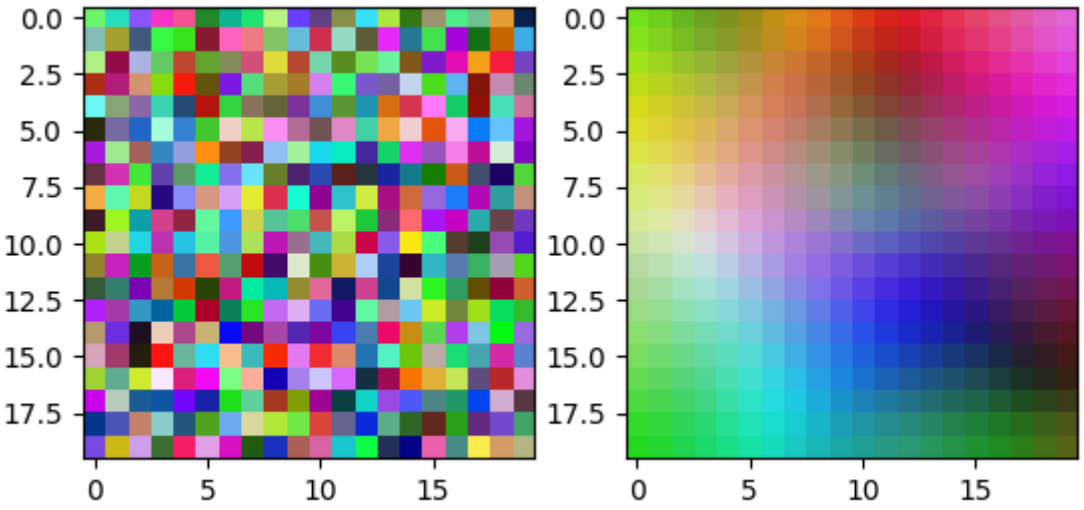

Рисунок 1.3. Самоорганізаційна Карта Кохонена.

Відновити цікавість до нейронних мереж вдалося Джону Хопфілду в 1982 році, який представив свою роботу, що стала відомою, як Мережа Хопфілда. Йому вдалося показати, як подолати проблеми перших поколінь ШНМ та оптимізувати їх роботу, що у свою чергу відновило дослідження та розробку нейрокомп'ютерів.

Алгоритм зворотного розповсюдження був запропонований у 1970-х роках, але повністю його потенціал було оцінено лише у 1986 році, завдяки Девіду Румельхарту, Джефрі Гінтону та Рональду Вільямсу. Вченим вдалось описати декілька нейронних мереж, де зворотне розповсюдження працює значно швидше, ніж попередні підходи до навчання, що дозволяє у певній мірі використовувати нейронні мережі для вирішення проблем, які раніше були б неможливими без цього підходу. На сьогоднішній день алгоритм зворотного розповсюдження є одним із найпопулярніших способів навчання нейронних мереж.

Незабаром Американський інститут фізики в 1985 році створив щорічну зустріч "Нейронні Мережі в Обчисленнях"("Neural Networks in Computing"), після якої відбулася перша у світі міжнародна конференція з нейронних мереж, організована Інститутом Інженерів з Електротехніки та Електроніки (IEEE) у 1987 році. На конференцію приїхали більш ніж 1800 спеціалістів.

У 1987 році США, Японія, та Європа почали масштабне фінансування досліджень у галузі ШНМ.

З 1989 - 1990 років дослідження в галузі штучних нейронних мереж значно популяризувались та почали проводитися більшістю великих електротехнічних компаній.

У 1992 - 1998 роках дослідження в області ШНМ активно розвиваються, кожного року проводяться форуми та конференції по всьому світу, кількість спеціалізованих наукових робіт у галузі ШНМ досягає декількох сотень, а кількість спеціалістів, що працюють у галузі зростає в рази.

Наступний важливий крок відбувся у 1999 році з появою графічних прискорювачів (GPU) та розвитком комп'ютерів, що дозволило суттєво пришвидшити обробку даних у тисячі разів. За відносно короткий проміжок часу

ШНМ почали конкурувати з опорно векторними машинами (SVM). Незважаючи на те, що нейронна мережа може бути повільною порівняно з векторною, ШНМ показують значно кращі результати з тими самими даними, оскільки їх перевага в тому, що вони постійно покращуються зі зростанням кількості даних.

В 2001 році у дослідницькому звіті Gather було описано можливості та потенційні проблеми зростання тривимірних даних, як наслідок збільшення кількості джерел та типів даних. Звіт був закликом підготовки до Big Data (Великі Дані).

Джефрі Гінтону в університеті Торонто у 2007 році вдалося створити алгоритми для глибокого навчання багатошарових нейронних мереж. Успіх полягає в тому, що Гінтон використовував обмежену машину Больцмана (RBM Boltzmann) для тренування нижніх шарів мережі [5].

Швидкість графічних процесорів значно зросла на протязі десяти років, що дозволило з 2011 року тренувати згорткові нейронні мережі (CNN) без попереднього навчання шар за шаром. Зі збільшенням швидкості обробки даних стало очевидно, що глибоке навчання значно ефективніше. Прикладом може бути мережа AlexNet, згорткова нейронна мережа, що виграла декілька міжнародних змагань на протязі 2011 – 2012 років.

В 2016 році групі розробників DeepMind компанії Google, вдалося за допомогою своєї штучної нейронної мережі AlphaGo перемогти світового чемпіона з Ґо (Китайська настільна стратегічна гра у яку грає два гравці. В плані стратегії вона значно важча за шахмати).

На сьогоднішній день штучні нейронні мережі активно розвиваються. Вчені зі всього світу працюють над новими алгоритмами навчання та новими архітектурами. На даний момент працюють сотні комерційних та не комерційних проектів, які вже давно довели свою ефективність.

1.2 Функції активації нейронів

Одна із найважливіших частин штучної нейронної мережі – функція активації. За допомогою цієї функції визначають вихід (результат роботи) моделі, точність цього виходу та ефективність з якою навчається модель.

Важливим параметром функції активації є те, що вона повинна бути ефективною, оскільки використовується сотнями, тисячами, а у деяких випадках мільйонами штучних нейронів. А з використанням алгоритму навчання зі зворотнім розповсюдженням помилки, на функцію покладається ще більше навантаження.

Популярною функцією активації є сигмоїдна функція (рис. 1.4). Діапазон її значень зазвичай знаходиться від -1 до 1, у деяких випадках використовують варіацію зі значеннями від 0 до 1.

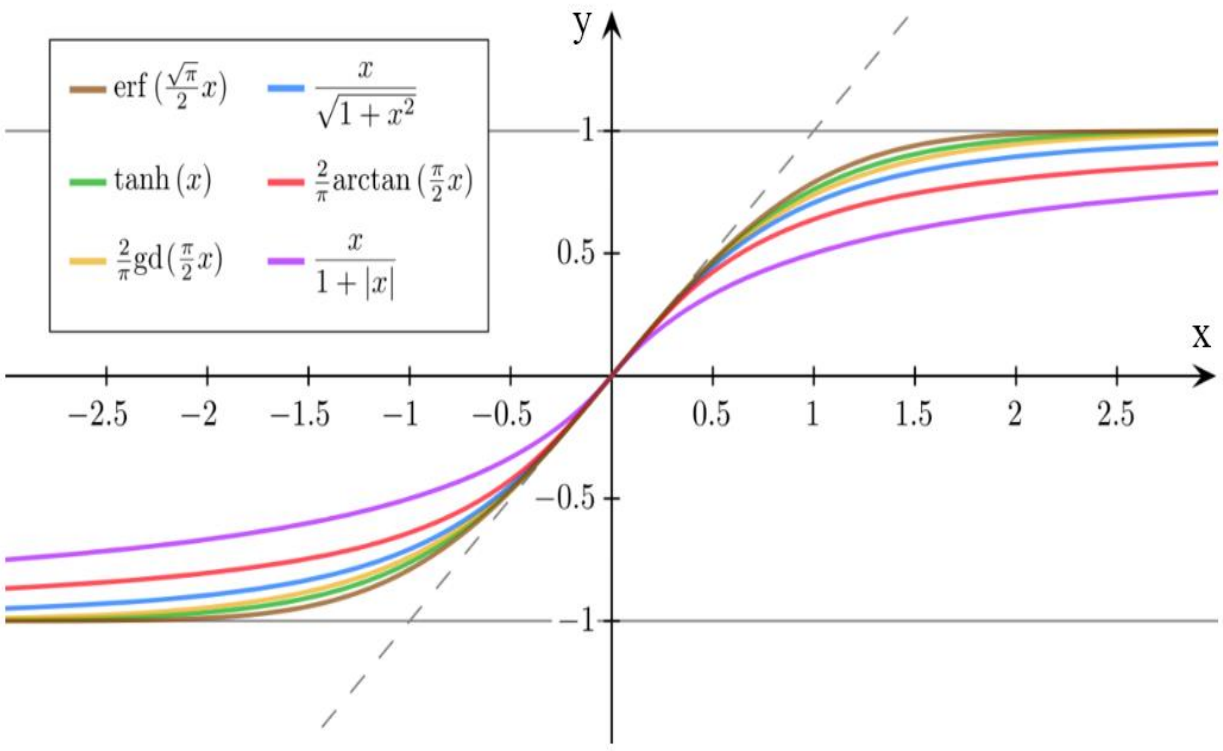

Рис. 1.4. Графік сигмоїдних функцій активації.

Ще одна нині популярна функція активації – ReLU (Випрямний лінійний блок). На даний момент є найбільш популярною функцією. Описується формулою:

$$f(x) = \max(0, x) \qquad (1.1)$$

Функція також має свої підвиди: ReLU Softplus та ReLU Rectifiler (рис. 1.5) та декілька інших. Хоча і існує велика кількість альтернатив для ReLU, ні одна функція не змогла повністю замінити її.

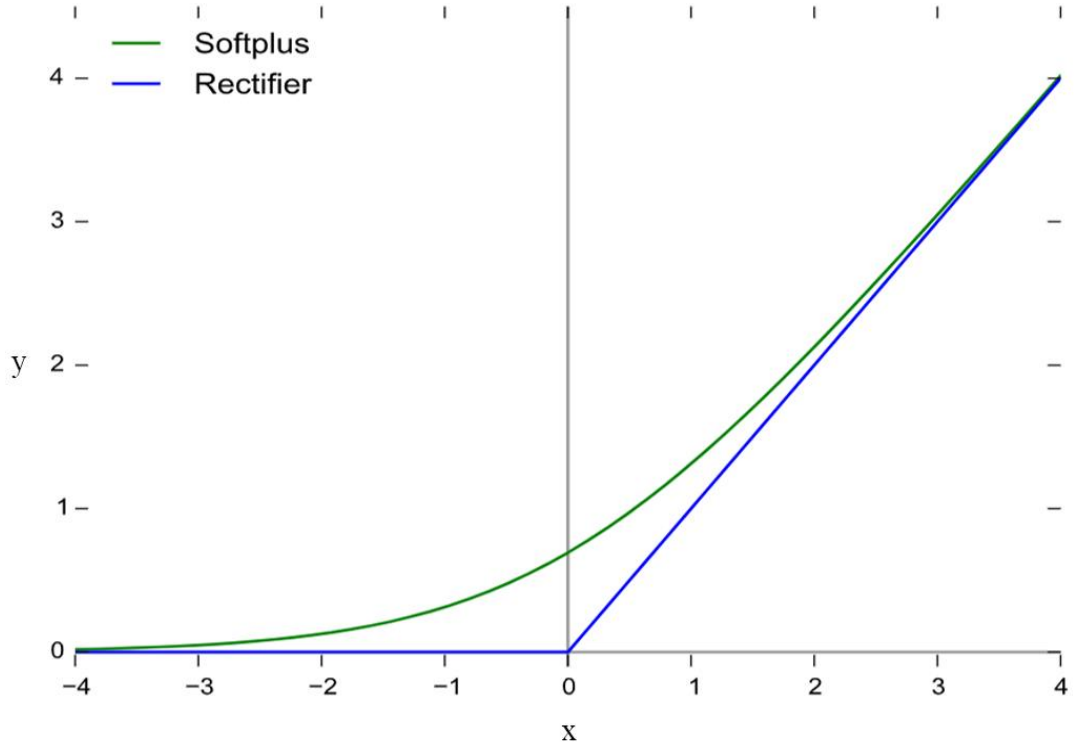

Рис. 1.5. Графік функцій активації ReLU: Softplus та Rectifiler.

Команда Google Brain Team у 2017 році запропонувала нову функцію активації - Swish (Рисунок 1.6.). Функція досить схожа на ReLU, але показує себе у деяких випадках значно краще. При тестуванні до 40 шарів штучної нейронної мережі з використанням ReLU та Swish, функції показують схожий результат, але при збільшенні шарів та більш складних даних Swish значно виходить в перед. Команді дослідників вдалося в деяких випадках збільшити точність на 0.9%. Формула, що описує дану функцію:

$$f(x) = x\ sigmoid(x), \tag{1.2}$$

де $sigmoid\ (x) = 1/(1 + exp(-x)$

В 2018 році дослідник Ерік Алкайде, запропонував дещо модифіковану версію функції Swish, він назвав її - E-swish (рис. 1.6.). Формула функції:

$$f(x) = \beta x\ sigmoid(x), \tag{1.3}$$

де $sigmoid\ (x) = 1/(1 + exp(-x)$

Властивості E-swish дуже схожі на Swish, а при $\beta = 1$ повністю повторює Swish. Однак в результаті тестування E-swish забезпечив зростання точності нейронної мережі на 0.35% відносно ReLU та на 0.6% відносно Swish.

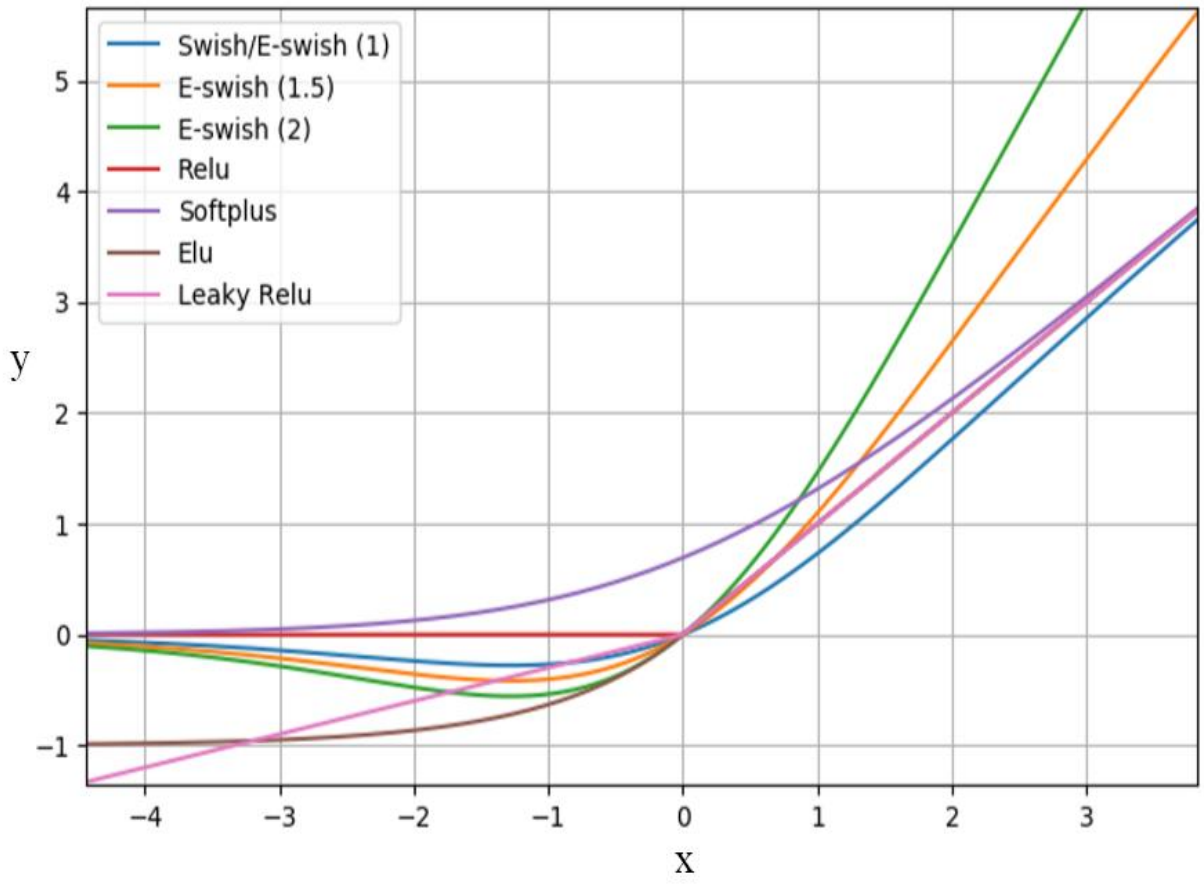

Рис. 1.6. Графіки Swish, E-Swish, та інших популярних функцій активації.

Наступна, популярна раніше функція, яка нині вже не так часто застосовується – Binary step (рис 1.7), яка може приймати тільки 2 значення: 0 та 1. Функція повертає результат 1(істина), за умови, що вхід проходить певний граничний поріг, або 0 (хиба), коли вхід не переходить граничний поріг. Binary step часто використовують для досліджень бінарної класифікації. Формула, що описує дану крокову функцію:

$$f(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases} \qquad (1.4)$$

Власне, майже всі традиційні логічні функції можуть бути реалізовані штучними нейронними мережами. Тому крокові функції у більшості випадків використовуються в примітивних нейронних мережах без прихованого шару або взагалі в одношарових мережах. Такий тип мережі може класифікувати лінійно відокремлювані проблеми, як наприклад: логічні AND та OR. Іншими словами, всі класи (0 і 1) можна розділити однією прямою лінією

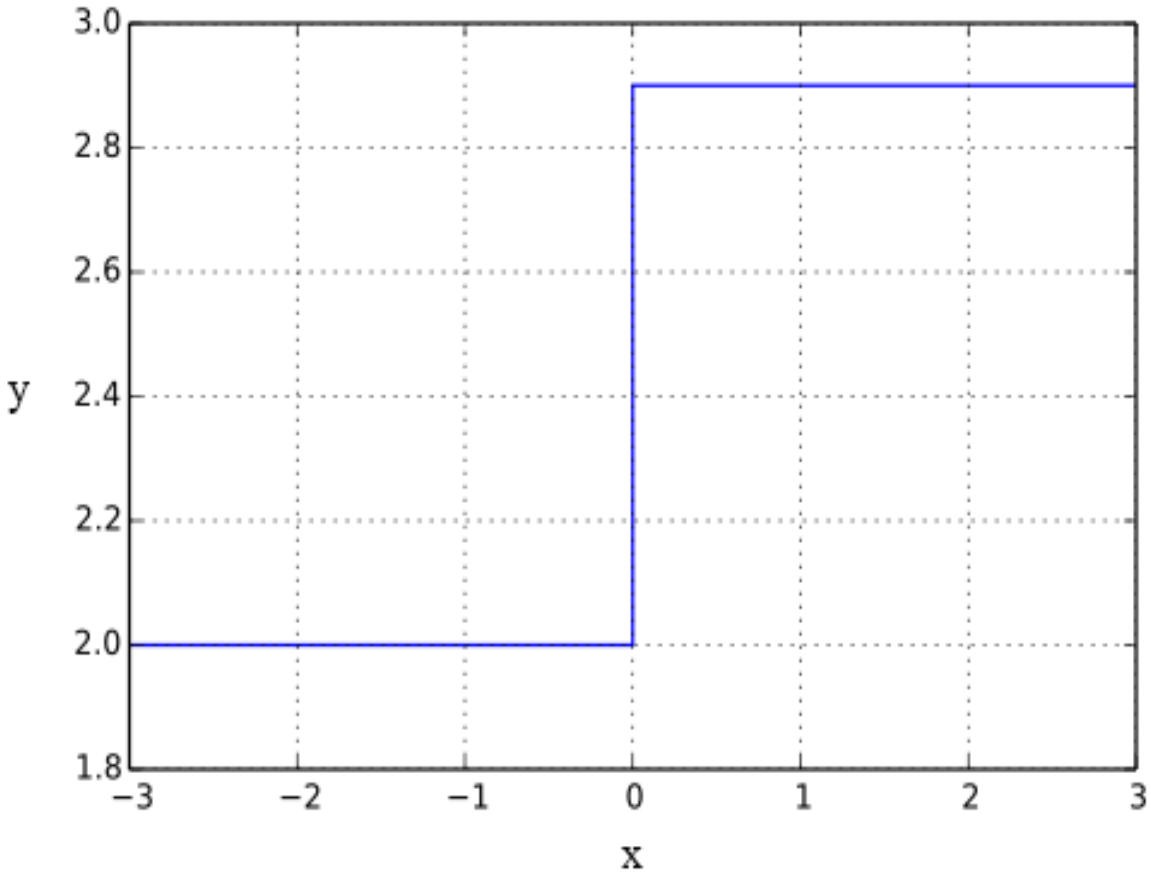

Рис. 1.7. Графік функції Binary step.

Існує ще велика кількість функцій активації штучних нейронів, мало того, їх кількість зростає щорічно. Варто зазначити, що у виборі функції активації з такого різноманіття є багато нюансів, починаючи від завдань штучної нейронної мережі, і закінчуючи її навчанням. На сьогоднішній день найбільш популярними і продуктивними можна назвати функції активації: ReLU, Swish.

Сигмоїдні функції та їх комбінації зазвичай краще показують себе у завданнях класифікації, однак в деяких випадках варто їх оминути, через проблему зникаючого градієнту.

ReLU широко використовується по всьому світу в різноманітних ШНМ. Є основним варіантом за замовчуванням, в більшості випадків показує найкращу продуктивність. Однак дану функцію варто використовувати тільки в прихованих шарах.

Альтернативою ReLU у деяких випадках виступає Swish та E-swish, що можуть показати значно кращу продуктивність за певних умов та в певних обставинах. У Swish та E-swish значно плавніший графік вихідного ландшафту (рис. 1.8) штучної нейронної мережі. Плавність ландшафту точно корелює з ландшафтом помилок, тому важливо, щоб графік був, якомога плавнішим, що значно полегшить знаходження та обрахунок помилки.

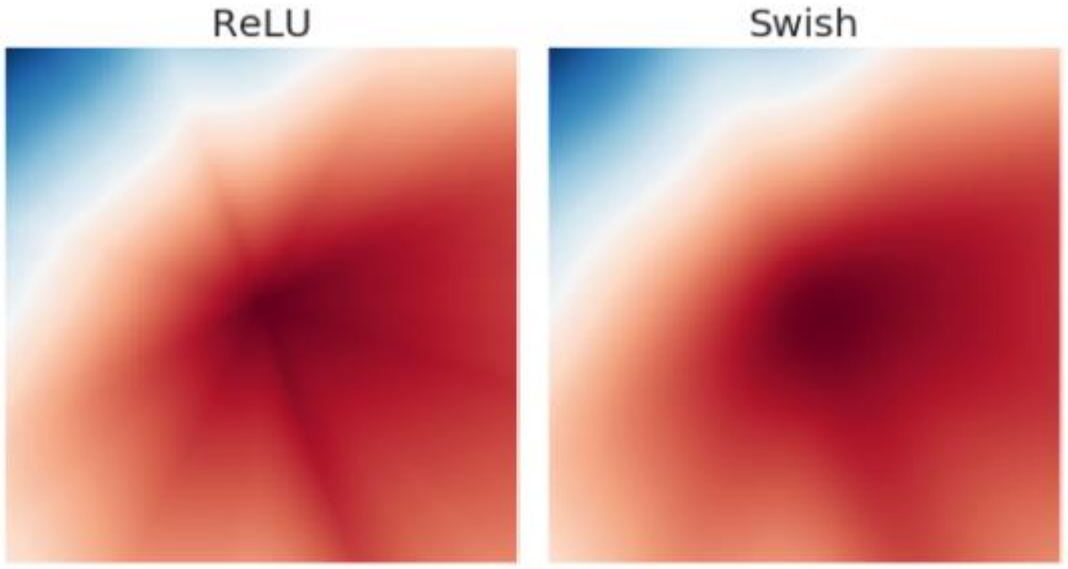

Рис. 1.8. Візуалізація вихідного ландшафту Swish та ReLU.

1.3. Архітектура нейронних мереж

1.3.1. Перцептрон

Першим поколінням ШНМ був перцептрон (рисунок 1.8) – це проста обчислювальна модель одного нейрона. Математична модель якого була запропонована Ф. Розенблатом на початку 1960-х. Стандартна архітектура перцептрона - мережа прямого поширення (feedforward neural network), тобто вхідні дані надходять в нейрон, обробляються і результат надходить до виходу.

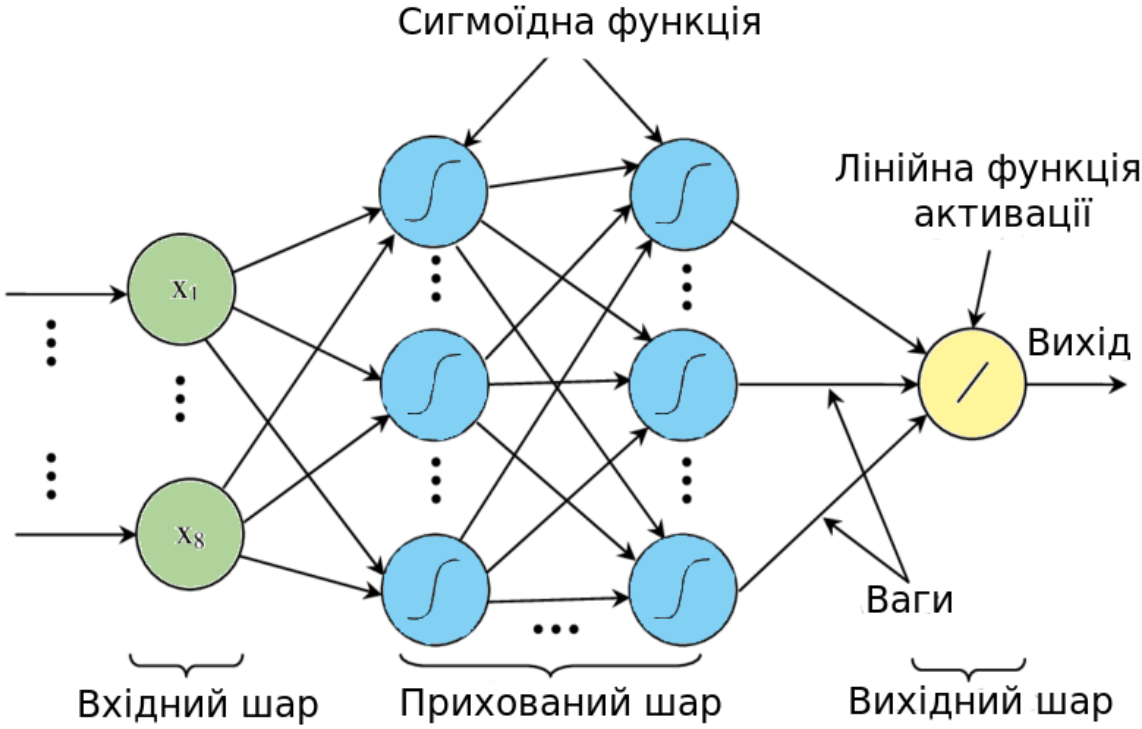

Рис. 1.8. Архітектура багатошарового перцептрона (MLP).

Така побудова архітектури означає, що мережа аналізує дані зліва направо. Ідея полягає в тому, що ми передаємо числові дані вперед по мережі, зробивши з ними багато операцій. Для отримання правильних операцій і щоб будь-який поданий вхід завжди давав бажаний результат, потрібне навчання перцептрона. Навчання по суті полягає в пошуку того, що дає найкращі результати, та застосуванні цих результатів до мережі. Варіанти використання перцептрона: комп'ютерне бачення (системи, які можуть розрізняти об'єкти на цифрових

зображеннях та відео), обробка природних мов та основа для інших нейронних мереж.

Однак у перцептрона є значні недоліки, оскільки він не може розпізнати шаблони після того як вони пройшли певні перетворення. Марвін Мінський та Сеймур Пейперт у своїй критиці перцептрона стверджують, що частина перцептрона, яка проходить навчання, не може навчитися, якщо перетворення утворюють групу. Тобто для боротьби з такими перетвореннями перцептрон повинен використовувати декілька функціональних одиниць для розпізнавання перетворень інформативних підшаблонів. Тому складну частину розпізнавання шаблонів, варто вирішувати за допомогою детекторів, що запрограмовані вручну, а не за допомогою процедури навчання [3].

1.3.2. Згорткові нейронні мережі

Група дослідників: Я. ЛеКун та його помічники на початку 2000-них розробили нейронну мережу (рисунок 1.9), для якісного розпізнавання написаних від руки цифр, яку вони назвали LeNet. Дослідники використовували алгоритм зворотнього розповсюдження у зворотній згортковій мережі зі значною кількістю прихованих шарів та безліччю карт повторюваних частин. Вони об'єднали виходи сусідніх копій в широку мережу, яка може впоратися з декількома символами одночасно, навіть якщо вони перетинаються [6]. Пізніше дана ШНМ формалізується під назвою згорткова нейронна мережа (CNN).

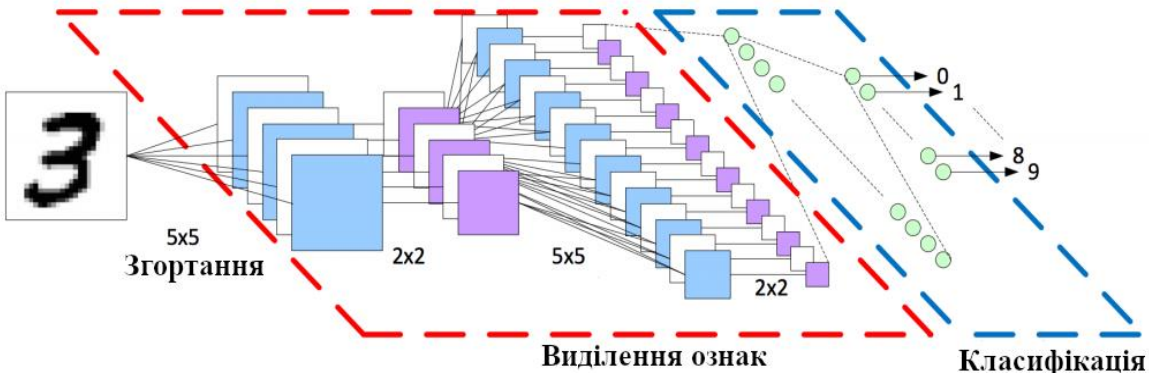

Рис. 1.9. Архітектура CNN для розпізнавання рукописних цифр.

CNN часто та вдало використовуються і у наші дні для вирішення завдань розпізнавання та класифікації. Завдання такого типу включають розпізнавання цифр, розпізнавання об'єктів та обробку природної мови. CNN приймають перекладену версію базової функції та об'єднують її для побудови трансляційних інваріантних функцій. Ділячись однією і тією ж базовою функцією в різних місцях, CNN має значно меншу кількість параметрів, які потрібно навчати, що дає можливість тренувати мережу з значно меншою кількістю прикладів, ніж якщо б абсолютно різні базові функції навчалися в різних місцях [7].

Згорткові мережі суттєво контрастують на фоні більшості інших мереж. Їх використання це зазвичай класифікація зображень, можливе і використання для обробки звукової інформації. Найпоширенішим випадком використання згорткової мережі є те, коли на вхід подається зображення з інтернету, а мережа має класифікувати його. CNN, у більшості випадків, приймають дані за допомогою вхідного сканера (рисунок 1.10), який не призначений для того, щоб одразу проаналізувати всі дані.

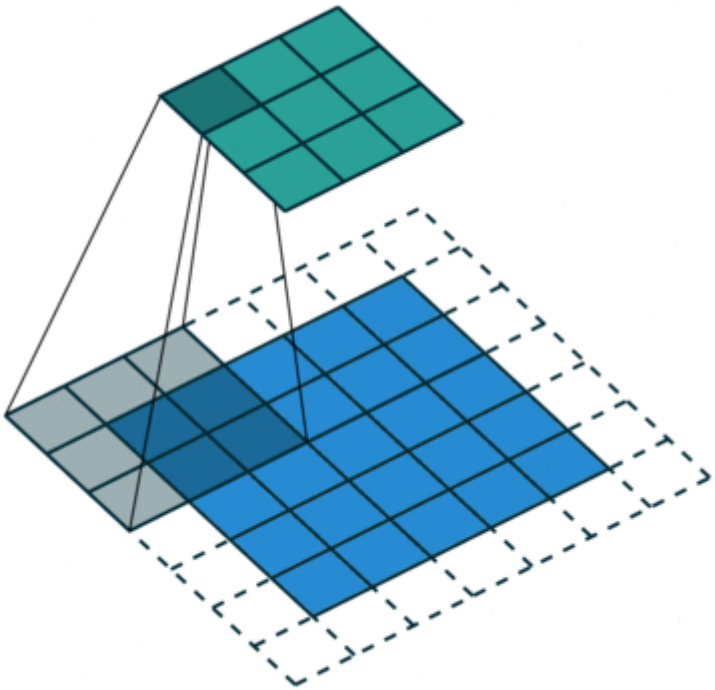

Рис. 1.10. Приклад роботи вхідного сканера CNN.

Наприклад, для введення зображення з параметрами 50 на 50 пікселів, щоб не робити вхідний шар з 2500 (загальна кількість пікселів даного зображення) вхідних вузлів, краще створити вхідний шар сканування, наприклад 5 на 5, на який подається перші 25 пікселів зображення. Після передачі цього вводу подати наступні 25 пікселів зображення, переміщуючи сканер на один піксель праворуч. Потім ці вхідні дані сканування передаються через згорткові шари замість простих шарів, у яких не всі вузли з'єднані з усіма між собою. Кожен вузол стосується лише тісних сусідніх комірок. Ці згорткові шари також мають тенденцію до скорочення, оскільки вони стають все глибшими, в основному за рахунок легко подільних факторів введення. Крім того, CNN користується поступальною інваріантністю, оскільки це жорстко запрограмовано в архітектурі мережі. Недоліком такого програмування є те, що архітектура охоплює лише поступальну інваріантність [8]. Наприклад, мережа не об'єднує між собою частини, які є тим самим об’єктом, але повернутим, або не фіксує складніші варіації, такі як повороти поза площиною.

1.3.3. Рекурентні нейронні мережі

Рекурентні нейронні мережі (RNN) на відміну від мереж прямого поширення, є варіантом рекурсивної мережі, де зв’язки між вузлами проходять певний цикл. Це означає, що результат роботи мережі залежить не тільки від теперішніх входів, але і від стану нейронів попереднього циклу. Ця пам'ять дає змогу вирішувати проблеми NLP, як-от підключення розпізнавання рукописного вводу або розпізнавання та обробка мови.

RNN запам'ятовує минуле, і на її рішення впливає те, що вона дізналася з попередніх результатів. Інші типи мереж також “запам'ятовують” речі, але вони пам'ятають тільки те, що вони дізналися під час навчання. Наприклад, мережа для класифікації цифр дізнається, як виглядає “5” під час навчання, а потім використовує ці знання для класифікації під час своєї роботи. У той час як RNN навчаються аналогічно під час навчання, крім того, вони запам'ятовують речі,

засвоєні з попередніми вхідними даними, створюючи вихідні результати. RNN можуть приймати один або декілька вхідних векторів і утворювати один або більше вихідних векторів, і на вихід (результат) впливають не тільки ваги, застосовані на входах, як і в звичайних нейронних мережах, але і прихований вектор стану, який являє собою контекст на основі попереднього входу / виходу [9]. Отже, один і той же вхід може давати різний вихід залежно від попередніх входів мережі.

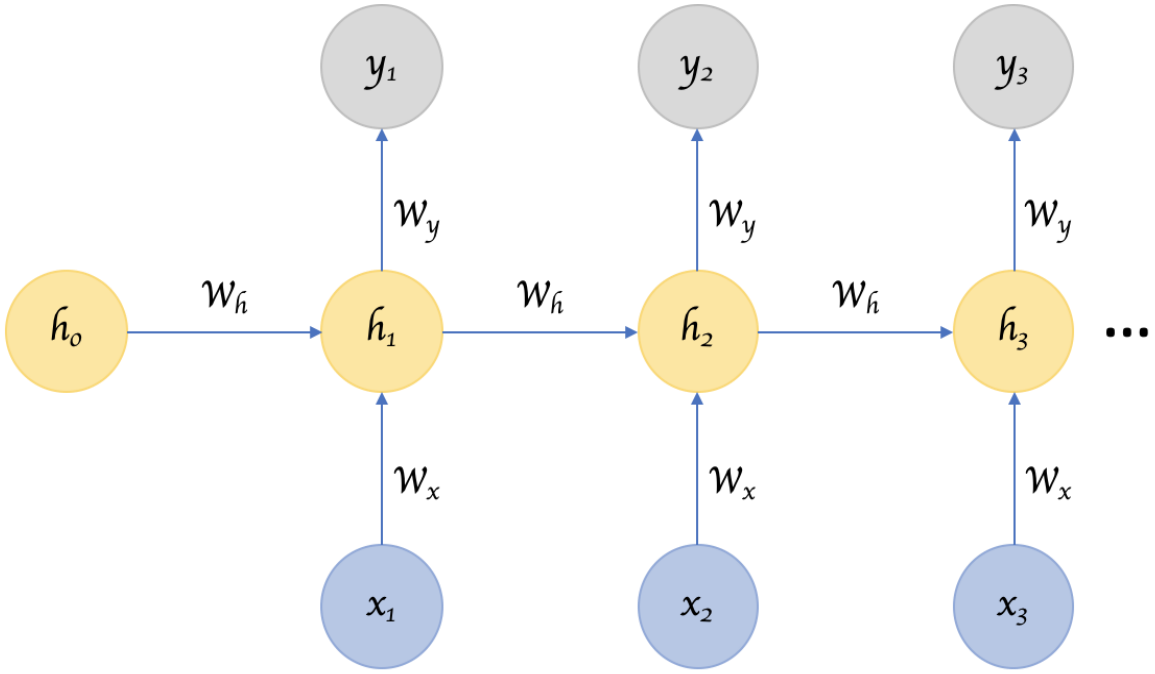

Рис. 1.11. Рекурентна нейронна мережа із прихованим шаром.

Де: x_n – вхідні вузли, h_n – приховані вузли,

y_n – вихідні вузли, w_n – ваги.

Група китайських даслідників: К. Лю, С. Лай та Д. Джао у статті "Генерування природних мов, перефразовування та узагальнення відгуків користувачів з рекурентними нейронними мережами" продемонстрували RNN мережу, для класифікації тексту без прописаних людиною ознак класифікації тесту. Їх мережу порівнювали з існуючими методами класифікації тексту, такими

як LR, SVM, LDA, RNN та CNN. Було показано, що дана модель значно краще справляється з завданнями такого типу, ніж традиційні методи для всіх використовуваних наборів даних. Підсумовуючи, у стандартних нейронних мереж вхідний вектор фіксованого розміру перетворюється на вихідний вектор фіксованого розміру. Таку ШНМ можна назвати рекурентною (рисунок 1.12.), коли вона неодноразово застосовує перетворення до серії заданих входів і виробляє з них серію вихідних векторів. Заздалегідь не встановлено обмеження на розмір вектора. І, крім генерування результату, який є функцією вхідного та прихованого стану, оновлюється і прихований стан на основі вхідної інформації, яка використовується при обробці наступного входу.

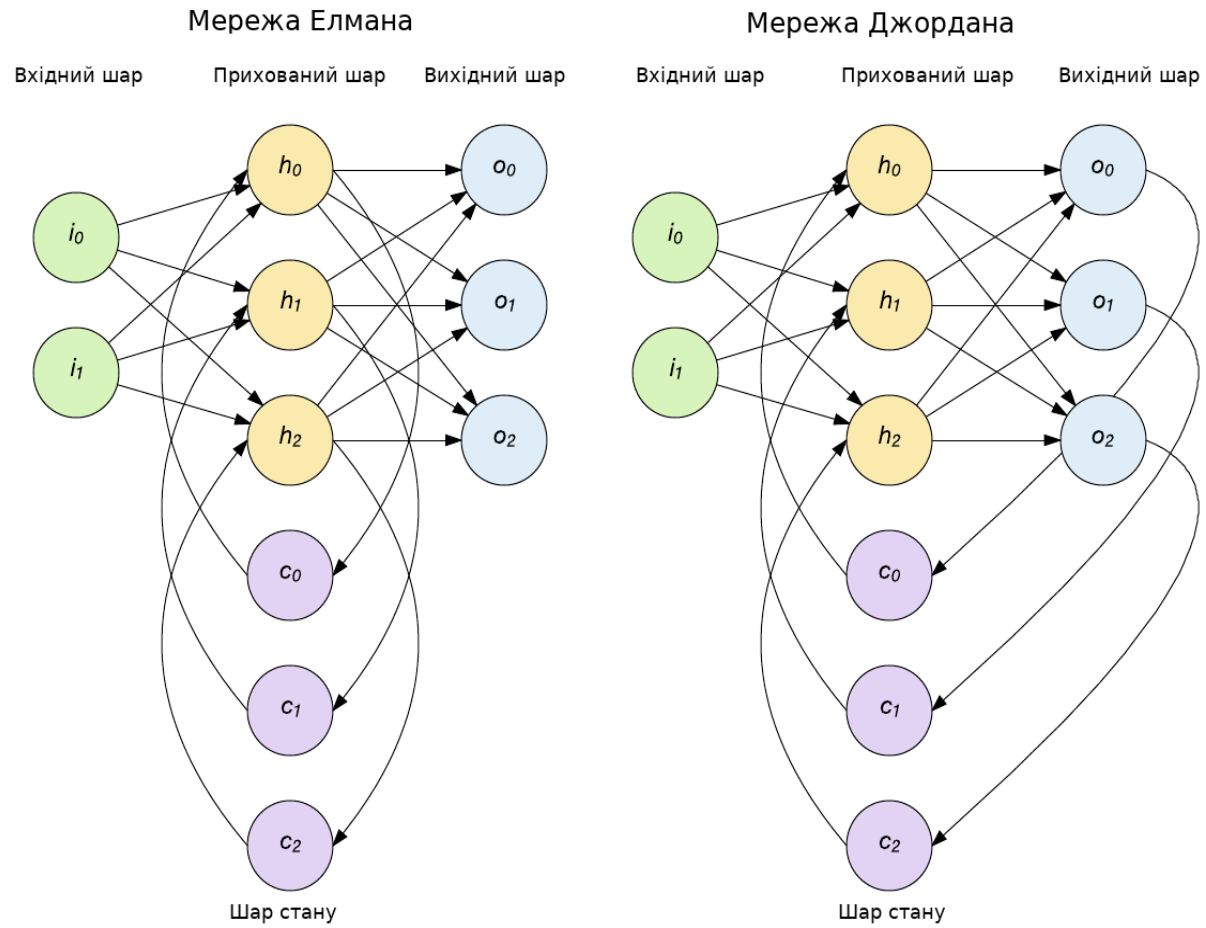

Рис. 1.12. Приклад рекурентних мереж Елмана та Джордана.

1.4. Навчання штучних нейронних мереж

Перший крок навчання – пряме поширення відбувається за допомогою функцій активації, коли мережа навчається на тренувальних даних, і вони перетинають всю нейронну мережу для їх прогнозування. Тобто передача вхідних даних через мережу таким чином, що всі нейрони застосовують свою трансформацію до інформації, яку вони отримують від нейронів попереднього шару, і передають її нейронам наступного шару. Коли дані пройдуть усі шари, і всі нейрони виконали свої обчислення, остаточний рівень буде досягнуто в результаті передбачення міток для цих вхідних прикладів [10] .

Наступний крок – використання функції втрат (помилки), щоб оцінити втрату (помилку) та порівняти наскільки хороший / поганий був прогноз мережі по відношенню до правильного результату. В ідеалі, потрібно, щоб похибка була нульовою, тобто без розбіжностей між отриманим та очікуваним результатом. Тому, коли модель навчається, ваги взаємозв'язків нейронів будуть поступово коригуватися, поки не будуть отримані хороші прогнози.

Після обчислення втрати ця інформація поширюється назад. Звідси і назва: зворотне розповсюдження помилки. Починаючи з вихідного шару, інформація про втрати поширюється на всі нейрони в прихованому шарі, які безпосередньо сприяють виводу. Однак нейрони прихованого шару отримують лише частину загального сигналу втрати, виходячи з відносного внеску, який кожен нейрон вніс у вихід. Цей процес повторюється, пошарово, до тих пір, поки всі нейрони мережі не отримають сигнал втрати, який описує їх відносний внесок у загальну помилку. Зворотне розповсюдження помилки можна розглядати, як метод зміни параметрів (ваги та зміщення) нейронної мережі в правильному напрямку. Метод починається спочатку з обчислення втрати, а потім параметри нейронної мережі коригуються у зворотному порядку за допомогою алгоритму оптимізації з урахуванням втрати.

Тепер, коли інформація поширилась назад, потрібно відрегулювати ваги

зв’язків між нейронами. Для цього використовується техніка, що називається градієнтним спуском. Ця методика міняє ваги невеликими кроками за допомогою обчислення похідної (градієнта) функції втрат, що дозволяє побачити, в якому напрямку потрібно “спуститись”, щоб дістатись до глобального мінімуму. Є одним із найпоширеніших алгоритмів оптимізації в машинному та глибокому навчанні. Градієнтний спуск використовує першу похідну (градієнт) функції втрат при оновленні параметрів. Процес полягає у прив'язці похідних втрат кожного прихованого шару до похідних втрат його верхнього шару та включенні його функції активації в обчислення (тому функції активації повинні обов’язково бути похідними).

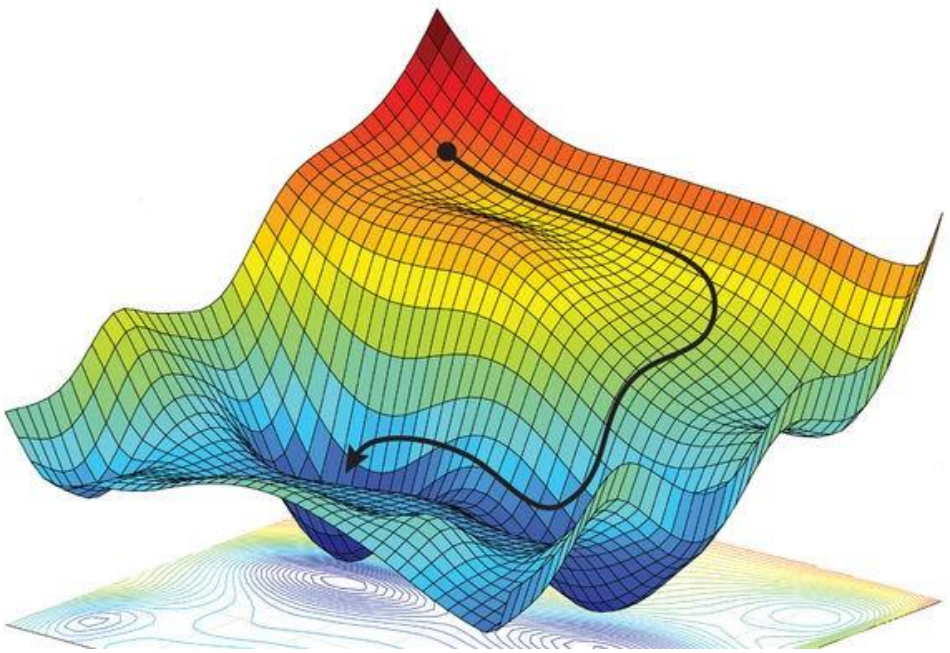

Рис. 1.13. Візуалізація градієнтного спуску.

У кожній із ітерацій, як тільки всі нейрони отримують значення градієнта функції втрати, що відповідає їм, значення параметрів оновлюються у зворотному напрямку, ніж зазначене градієнтом. Фактично, градієнт завжди вказує в тому напрямку, в якому збільшується значення функції втрат [10]. Тому, якщо використовується негативний градієнт, можна отримати напрямок, в якому функція втрат зменшується.

Отже, кроки, які потрібно виконати для навчання штучної нейронної мережі:

- Ініціалізація випадковими значеннями параметрів мережі (ваг та зміщення).
- Вибір набору даних та передавання їх через мережу, щоб отримати передбачення мережі.
- Порівняння отриманих прогнозів з очікуваними значеннями та обрахунок втрат (помилки).
- Виконати зворотне розповсюдження, щоб поширити цю втрату на кожен із параметрів, що складають модель нейронної мережі.
- Використовуючи отримані результати розповсюдити інформацію по мережі, для оновлення параметрів нейронної мережі зі спуском градієнта, таким чином, щоб зменшити загальні втрати та покращити модель.
- Продовжувати ітерацію, поки мережа не набере потрібний результат.

Висновок

В розділі розглянуто історію розвитку та створення штучних нейронних мереж. Також розглянуто різноманітні типи мереж, їх застосування, переваги та недоліки.

Для розробки програми обрано 3 мережі, що розглядались: мережа прямого поширення, згорткова мережа та рекурентна мережа. В якості функцій активації будуть використовуватись функції, що добре себе зарекомендували та показують непогану продуктивність, а саме: сигмоїдна функція та ReLU. Навчання буде здійснюватися за допомогою функції зворотнього розповсюдження помилки та градієнтного спуску.

РОЗДІЛ 2. ОГЛЯД ВИКОРИСТАНИХ ТЕХНОЛОГІЙ

2.1. Бібліотеки обробки природної мови

Обробка природної мови являє собою автоматичне опрацювання природної людської мови, такою як розмовна мова чи текст. Цінність цієї технології виходить із випадків використання. NLP може допомогти у виконанні безлічі завдань, а сфери застосування, здається, щодня збільшуються. Ось кілька прикладів:

- NLP дозволяє розпізнавати та прогнозувати захворювання на основі електронних медичних записів. Ці можливості досліджуються у сфері здоров'я, які переходять від серцево-судинних захворювань до депресії Наприклад, Amazon Comprehend Medical – це послуга, яка використовує NLP для знаходження захворювань, медикаментів та результатів лікування із записів пацієнтів, звітів про клінічні випробування та інших електронних медичних записів
- Організації можуть визначити, що говорять клієнти про послугу чи продукт, знайшовши та опрацювавши інформацію, в таких джерелах, як соціальні медіа. Цей аналіз настроїв може надати багато корисної інформації про вибір клієнтів та їх вподобання.
- Винахідник з IBM розробив помічник, який працює як персоналізована пошукова система, вивчаючи все про вас, а потім нагадує вам ім’я, пісню чи все, що ви не можете згадати в той момент, коли вам це потрібно.
- Такі компанії, як Yahoo та Google, фільтрують та класифікують електронні листи за допомогою NLP, аналізуючи текст в електронних листах, що проходять через їхні сервери та зупиняють спам, перш ніж він надійде у поштову скриньку.
- Alexa Siri від Amazon і Apple Siri – це приклади інтелектуальних голосових інтерфейсів, які використовують NLP для відповіді на голосові підказки і

роблять все, як знайти конкретний магазин, розповісти нам прогноз погоди, запропонувати найкращий маршрут.

- Розуміння того, що відбувається і про що люди говорять, може бути дуже цінним для фінансових трейдерів. NLP використовується для відстеження новин, звітів, коментарів про можливі злиття між компаніями, все це потім може бути включено в алгоритм торгівлі.

NLP за останні роки значно розвинувся (рис. 2.1), а особливо процвітає в галузі охорони здоров'я. Ця технологія покращує надання медичної допомоги, діагностування захворювань та зменшує витрати, в той час як організації охорони здоров'я переживають все більш широке прийняття електронних медичних записів. Те, що клінічну документацію можна вдосконалити, означає, що пацієнтів можна краще зрозуміти та збільшити ефективність лікування.

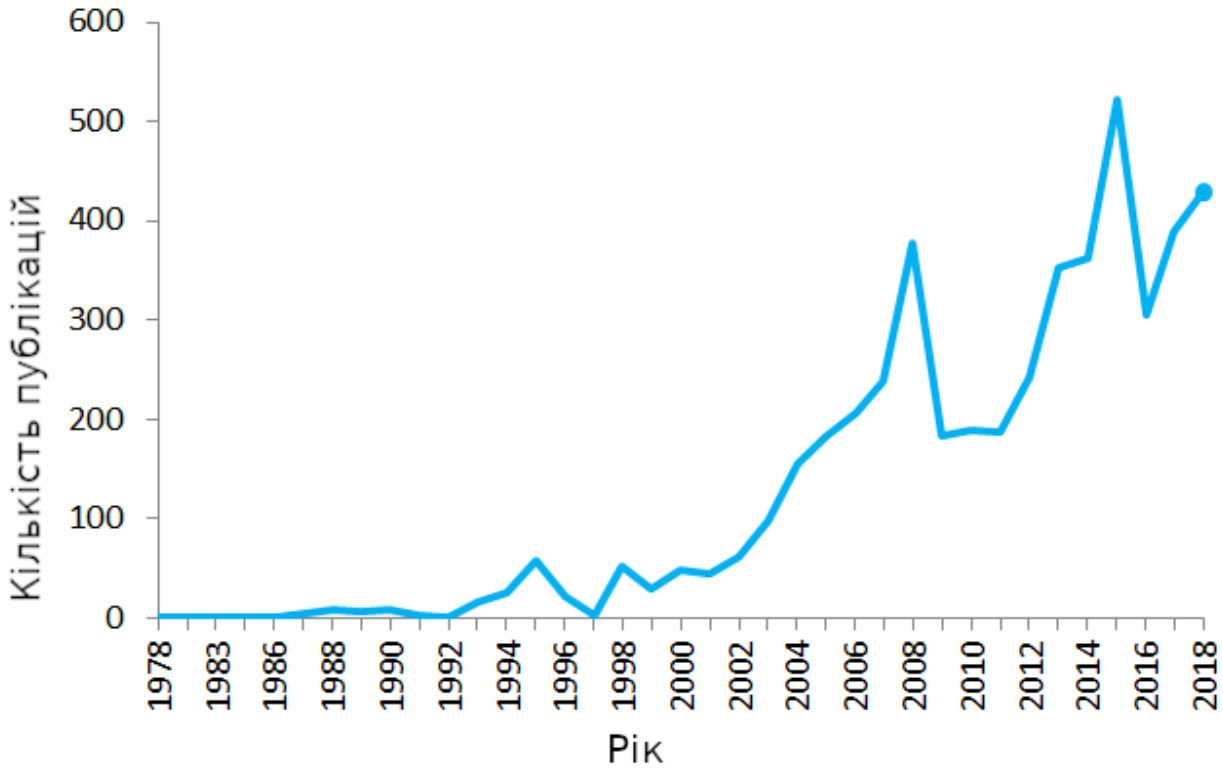

Рис. 2.1. Кількість публікацій в сфері NLP у період 1978 – 2018 років.

2.1.1. Бібліотека SpaCy

SpaCy (рис.2.2) – це бібліотека обробки природної мови, яка створена для роботи з популярною мовою програмування Python. Вона дозволяє робити глибокий пошук даних, який раніше був недосяжний для програмного забезпечення NLP [11].

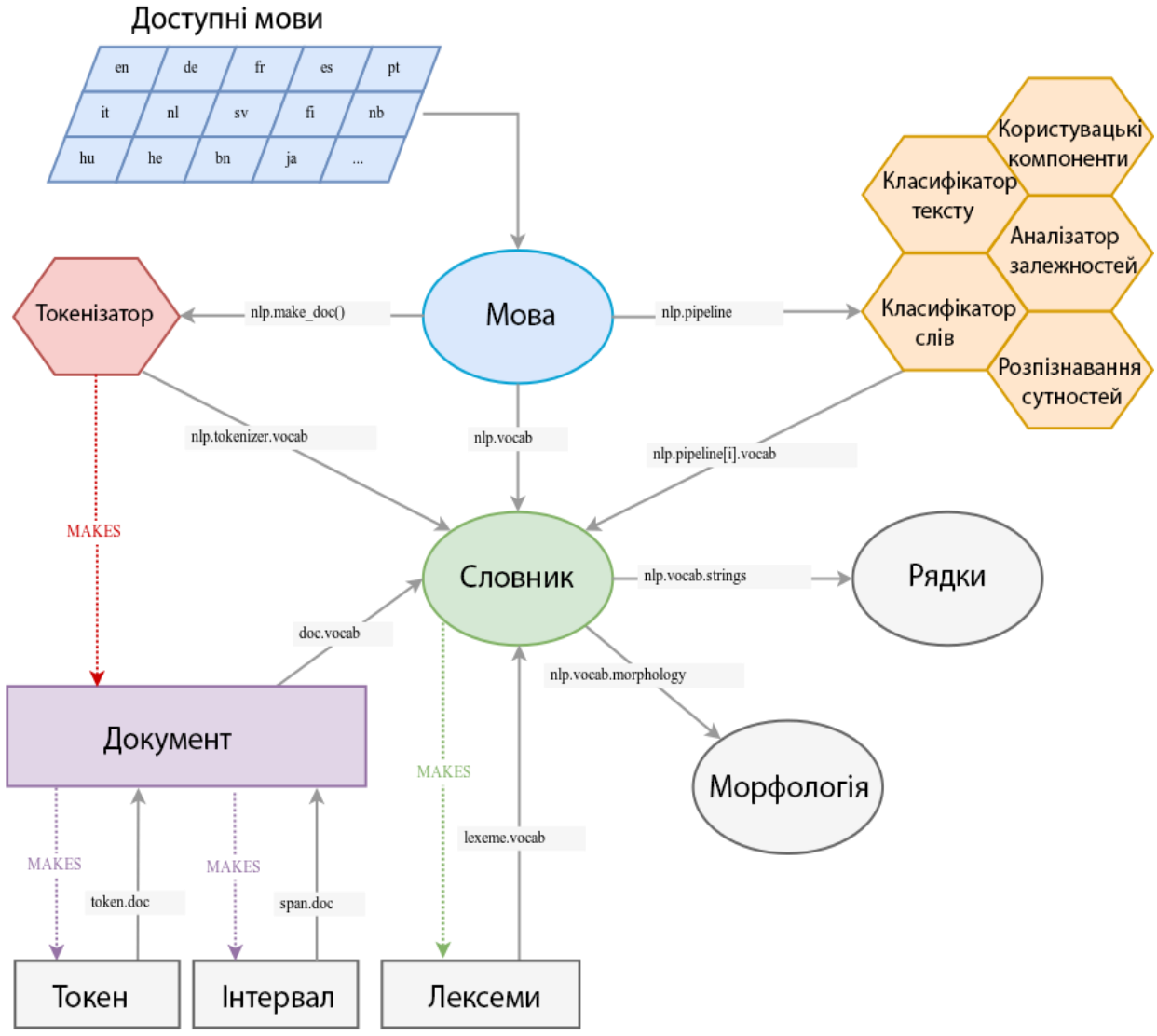

Рис.2.2. Архітектура бібліотеки SpaCy.

Дана бібліотека позначає більше складних частин мови і може знаходити та розпізнавати зразки з меншою кількістю дублікатів, завдяки можливостям та

навчанню своєї нейронної мережі (рис.2.3). Бібліотека SpaCy - один із найкращих варіантів для найважчих функцій обробки природних мов, що пропонує токенізацію (процес класифікації слів) та аналіз складних даних. SpaCy пропонує лематизацію (перетворення токенів до їх словникового вигляду) і є однією з небагатьох систем обробки природної мови високого рівня, які пропонують цю функціональність [11]. Частина мовного тегування (процес класифікації) проста і швидка, що робить її найшвидшим синтаксичним мовним аналізатором у світі. Код SpaCy є відкритим та загальнодоступним, він заснований на ліцензії MIT, та відразу готовий до застосування для процесів глибокого навчання та обробки природної мови. Перелік можливостей SpaCy:

- Токенізація.
- POS (Part of Speech Tagging) – визначення частин мови.
- Класифікація.
- Лематизація.
- Пошук залежностей.
- Представлення слів у вигляді векторів.

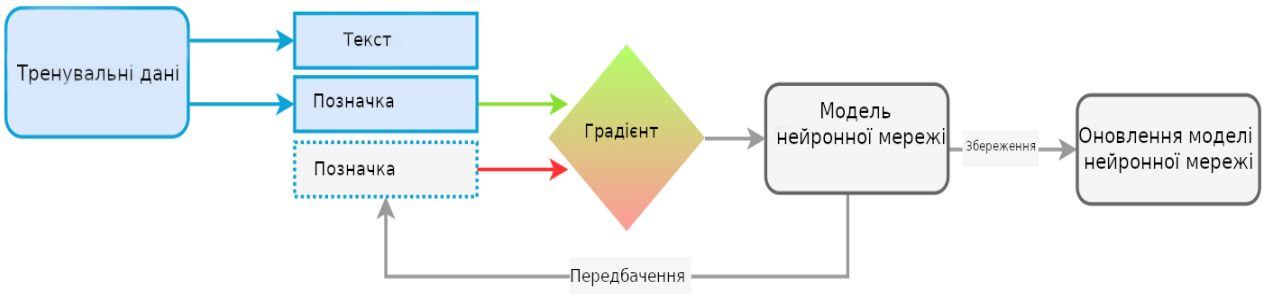

Рис. 2.3. Алгоритм навчання нейронної мережі SpaCy.

Користувачі SpaCy – люди, що займаються наукою даних. Вони сортують великі об’єми тексту, розуміють глобальні шаблони та створюють більш швидкі та більш оптимізовані програми з кращими можливостями прогнозування. Оскільки програмне забезпечення орієнтоване на виробництво, користувачі застосовують SpaCy для цілей, що виходять далеко за межі досліджень. SpaCy

створений для корпоративних рішень з великим навантаженням, аналізу текстових частин та пошуку будь-якого глибшого сенсу. Як і завжди з цими програмами, якщо ви не будуєте щось гідне для виробництва, це може бути занадто багато.

SpaCy має чудову документацію, яка допомагає застосовувати різноманітні алгоритми обробки тексту та даних. Однак, навіть якщо бібліотека не надала б документацію, вона створена з урахуванням нелінгвістичного мислення. Таким чином, SpaCy може будувати складні формули для визначення та тегування текстів, не вимагаючи лінгвістичного підґрунтя.

Підсумовуючи, SpaCy пропонує величезну екосистему інструментів обробки даних, побудованих на платформі з відкритим кодом, виготовленої для виробництва. Бібліотека дозволяє розробити, як і корпоративні рішення, так і унікальні рішення NLP для декількох мов. Це потужний інструмент, який підійде для будь-якого алгоритму глибокого навчання, який може тільки знадобитися. Навіть із найсучаснішими алгоритмами, бібліотека залишається відкритою та доступною. Сучасний метод розпізнавання суб'єктів працює якісно, а його тегування частин мови високоточне та швидке.

Плюси бібліотеки SpaCy:

- Призначена для використання у виробництві.
- Проста та легка у застосуванні.
- Найшвидший синтаксичний аналізатор.

Мінуси:

- Мала кількість підтримуваних мов.
- Недостатня гнучкість .
- Мала кількість методів обробки.

2.1.2. Бібліотека Text Blob

TextBlob – це бібліотека та API Python, яка пропонує вдосконалену, але водночас просту обробку природної мови. Вона повністю сумісна з реалізаціями Python 2, так і 3 версії. Обсяг бібліотеки досить широкий, що дозволяє підтримувати переважну більшість завдань на основі видобутку інформації. Це варіюється від вилучення предметника чи іменника до класифікації емоційної тональності тексту. TextBlob безкоштовна та ліцензована відповідно до GPL. Офіційної платної структури підтримки немає, але навколо програми можна знайти досить активну спільноту розробників [13]. Це означає, що безкоштовні оновлення та виправлення помилок у базових бібліотеках відбуваються регулярно.

TextBlob розроблена для та навколо Python, що дозволяє використовувати дану бібліотеку для управління даними NLP в рядку Python, звідки вона функціонує так, як можна було б очікувати від будь-якої стандартної функції. TextBlob також піддається швидкому прототипуванню та впровадженню. Бібліотека легко то швидко запускається за допомогою стандартного інтерпретатора Python, який може запропонувати простий спосіб отримати основні функції TextBlob.

Варто зазначити, що TextBlob призначена не лише для новачків. Інструмент пропонує широкий спектр функціонування на базі NLP, включаючи деякі цікаві доповнення. Наприклад, TextBlob не просто може ідентифікувати мову в певному рядку - а також пропонує функції легкого підключення до API Google Translate для майже миттєвого перекладу на інші мови.

TextBlob пропонує більшість стандартних функцій, присутніх в інших програмних пакетах NLP. Наприклад, бібліотека дозволяє легко налаштувати аналіз дерева тексту. Однак, TextBlob відрізняється рівнем абстракції, який використовується для цих базових функцій, що у свою чергу дозволяє налаштовувати складні процеси лише за допомогою декількох рядків коду на Python [12]. Наприклад, тональний аналіз настроїв може бути виконаний

приблизно з такою ж кількістю коду, що і початкова токенізація тексту. TextBlob пропонує ряд корисних функцій, що значно спрощують процес написання коду. До них належить згадана вище інтеграція з Google Translate, функція виправлення помилок, щоб виправити поширені орфографічні помилки. Більшість цих розширених функцій працюють просто та ефективно. Однак, як правило, легко отримати детальніші дані, якщо це потрібно. Наприклад, бібліотека дає можливість встановити за замовчуванням найбільш вірогідну пропозицію щодо написання орфографії. Можливості TextBlob:

- Токенізація.
- POS.
- Класифікація тексту.
- Пошук залежностей.
- Перевірка орфографії.

Підсумовуючи, TextBlob пропонує унікальне поєднання зрілої платформи NLP та швидкого прототипування. TextBlob – це один із найшвидших способів реалізації функціональності NLP. Вона зосереджена на наданні доступу до загальних операцій з обробки тексту через звичні інтерфейси. Об'єкти TextBlob можна розглядати як рядки Python, які навчаються з обробки природних мов. TextBlob пропонує API для виконання загальних завдань NLP, таких як визначення частин мови, аналіз настроїв, класифікація, переклад мови, перебір слів, та інтеграція WordNet. Стандартна система Python дозволяє легко тестувати функції в режимі реального часу. І дещо абстрагований характер як бібліотеки, так і самого Python зумовлює досить короткі цикли розробки продукту. Бібліотека TextBlob в цілому добре відшліфована і дотримується принципу єдиного дизайну

Переваги бібліотеки TextBlob:

- Проста у використанні.
- Забезпечує переклад та виявлення мови за допомогою Google. Translate.

- Присутня перевірка орфографії.
- Гнучка система.

Недоліки:

- Повільна.
- Потрібно використовувати в парі з іншою бібліотекою NLP.
- Відсутнє представлення слів у вигляді векторів.

2.2. Бібліотеки проектування та навчання нейронних мереж

За останні кілька років машинне та глибоке навчання (ML/Dl) дуже швидко зростає. Є численні бібліотеки, над якими працюють сотні розробників (рис. 2.5.), які використовуються для побудови моделей машинного та глибокого навчання, близько 50 з них досягли того рівня, щоб ними можна було б повноцінно користуватися.

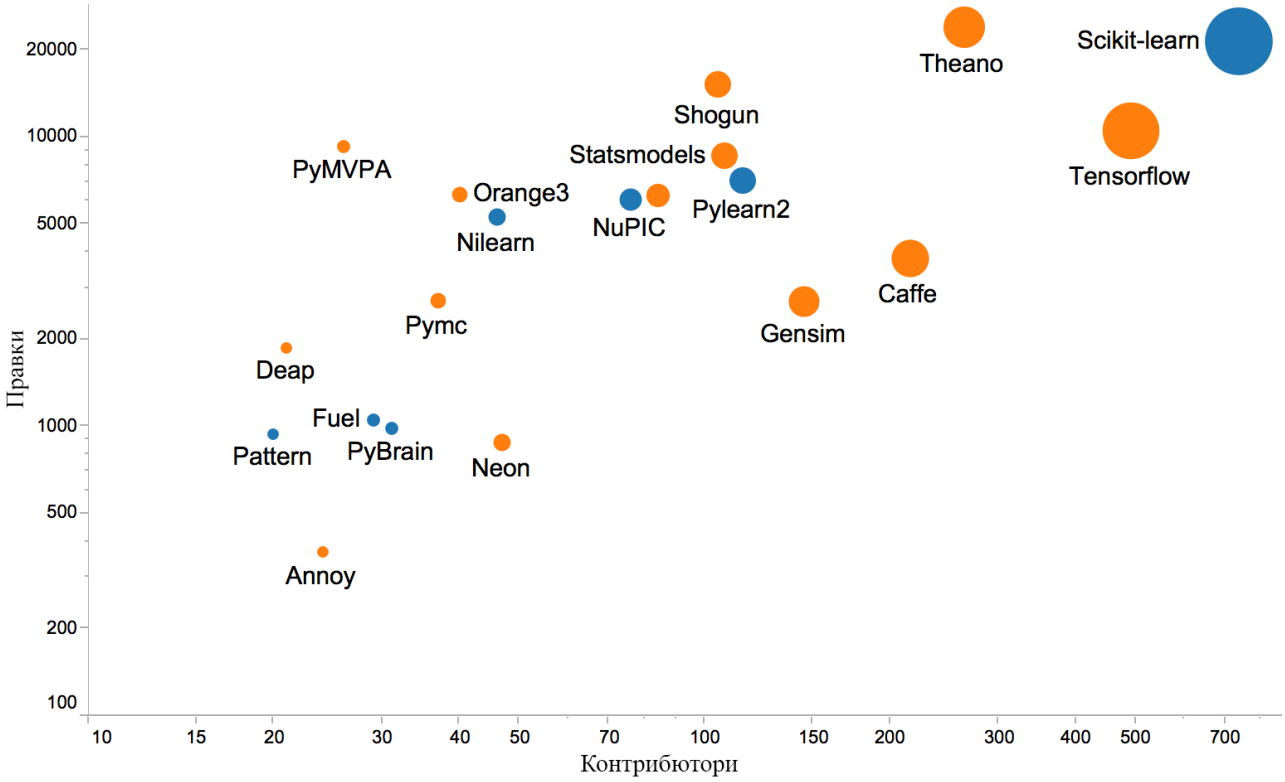

Рис. 2.5. Графік бібліотек ML/DL за кількістю контрибюторів та правок.

Машинне навчання, використовує математичні моделі загального призначення для відповіді на конкретні запитання за допомогою даних. ML застосовується для виявлення спаму в електронній пошті, створення інтелектуальних роботів, виявлення об'єктів за допомогою комп’ютерного бачення (CV), побудови розумних будинків, розпізнавання мови, побудови системи, здатної писати (романи, поезії тощо), рекомендувати продукти клієнтам та для прогнозування вартості товарів протягом багатьох років.

Раніше люди виконували завдання машинного навчання, вручну кодуючи всі алгоритми та математико-статистичну формулу. Це робило процес трудомістким, стомлюючим та неефективним. Але в сучасні дні це стає дуже просто та ефективно у порівнянні зі старими днями, користуючись різними бібліотеками та модулями Python. Сьогодні Python є однією із основних мов програмування, що використовуються для таких завдань, що дозволило замінити багато мов у галузі, одна з причин - велика колекція бібліотек. Деякі бібліотеки Python, що використовуються в машинному навчанні:

- TensorFlow.
- Keras.
- PyTorch.
- Scikit-learn.
- Numpy.
- Pandas.
- Theano.
- Keras.
- Caffe.
- Seaborn
- Matplotlib

2.2.1. Бібліотека TensorFlow

TensorFlow – це бібліотека з відкритим кодом для чисельних обчислень з використанням графів потоку даних. Спочатку вона була розроблена командою Google Brain в рамках дослідницької організації Google Machine Intelligence для машинного навчання та досліджень нейронних мереж, але ця система є загальною, щоб застосовувати її також у багатьох інших областях. У лютому 2017 року була випущена версія 1.0 та з того часу бібліотека продовжує бурхливий розвиток, на сьогоднішній день 21 000+ контрибуцій в проект [13].

Машинне навчання (рис. 2.6.) може швидко отримати складний характер, а моделі глибокого навчання можуть стати надто великими. Для багатьох моделей графів потрібно розподілити навчання, щоб мати можливість повторити його за незначний проміжок часу.

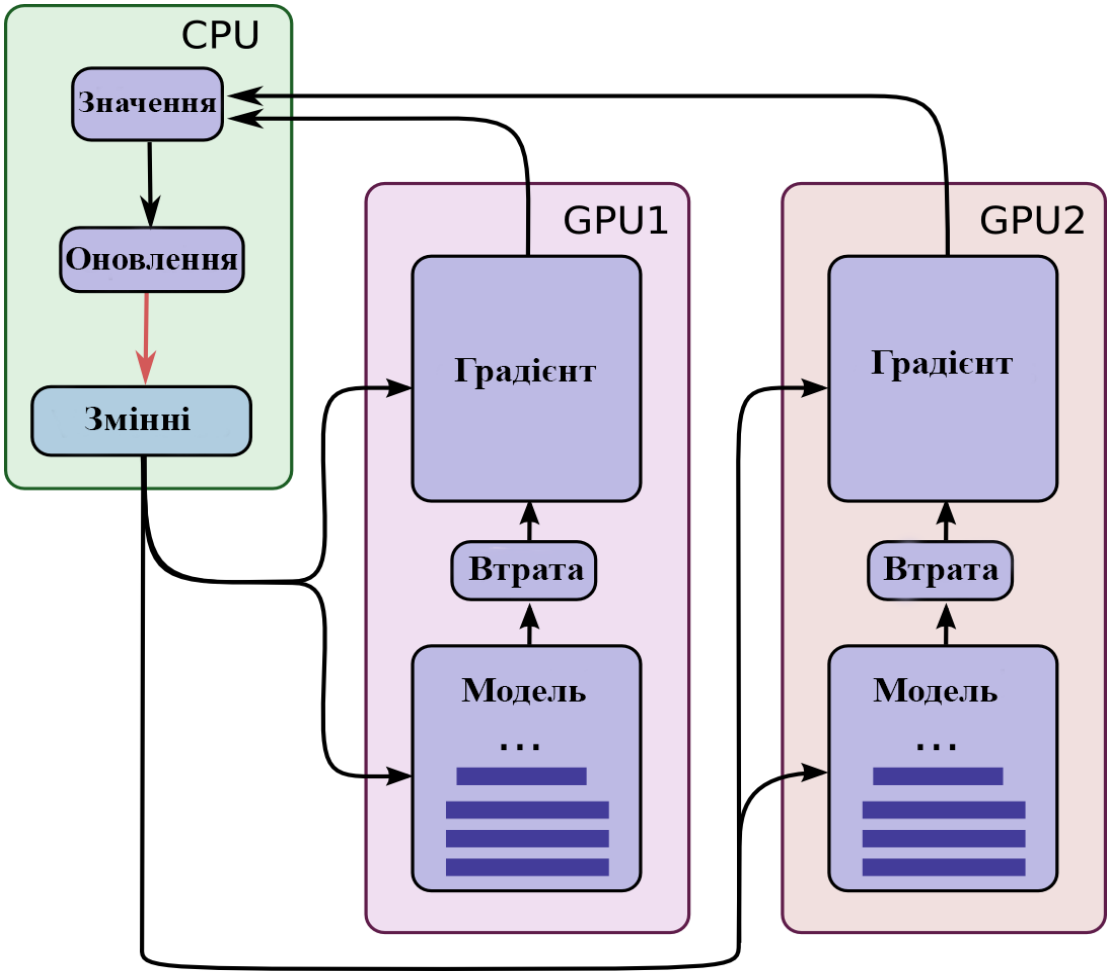

Рис. 2.6. Процес навчання ШНМ в TensorFlow.

З поточною версією TensorFlow потрібно написати код для побудови графа обчислень, а потім виконати його. Граф в TensorFlow – це структура даних, яка повністю описує обчислення, які потрібно виконати [13]. Така структура має декілька значних переваг, наприклад:

- Він портативний, оскільки граф можна виконати негайно або зберегти для використання пізніше, і він може працювати на декількох платформах: процесорах, графічних процесорах, TPU, мобільних платформах. Крім того, він може бути розгорнутий у виробництво без необхідності залежати від будь-якого коду, який побудував даний граф.
- Граф трансформується та оптимізується, оскільки він може бути перетворений для отримання більш оптимальної версії для даної платформи. Також можна здійснити оптимізацію пам'яті або обчислення та різноманітні компроміси між ними.
- Підтримка розподіленого виконання. API високого рівня TensorFlow у поєднанні з обчислювальними графами дозволяють забезпечити продуктивне та гнучке середовище розробки та потужні виробничі можливості в тих же рамках.
- Швидке виконання. Наступним доповненням до TensorFlow є швидке виконання, імперативний стиль написання коду TensorFlow. Увімкнувши швидке виконання, ядра TensorFlow будуть виконуватись негайно, а не будувати графи, які будуть виконані пізніше.

Такі властивості TensorFlow дозволяють легко перевірити і налагодити проміжні значення у графах, та використовувати потоки управління Python в API TensorFlow: цикли, умови, функції, тощо. А швидке виконання повинне зробити налагодження більш простим. Після того код TensorFlow запрацював, можна перетворити його в граф автоматично. Це полегшить збереження, розміщення та розповсюдження графів.

TensorFlow та спільнота програмного забезпечення з відкритим кодом: дана бібліотека була відкрита для загального доступу у значній частині, щоб спільнота

могла покращити її за допомогою власних внесків. Команда TensorFlow налаштувала процеси управління запитами, перегляд та подання проблем, а також відповіді на запитання від розробників по даній технології. TensorFlow має більше 76 000 зірок (Оцінені високо користувачами) на GitHub (рис.2.7.) і кількість інших репозиторіїв, які використовують TensorFlow , зростає з кожним місяцем, станом на сьогоднішній день їх кількість налічується понад 20 000. Багато з них – навчальні посібники, моделі, переклади та проекти, створені громадою. Вони можуть бути чудовим джерелом прикладів для новачків, які тільки починають працювати з машинним навчанням [13]. Команда TensorFlow активно слідкує за питаннями в Stack Overflow (система питань-відповідей для програмістів), і це хороший спосіб отримати відповіді на питання по даній технології (на теперішній час наявні більш ніж 8 000 відповідей).

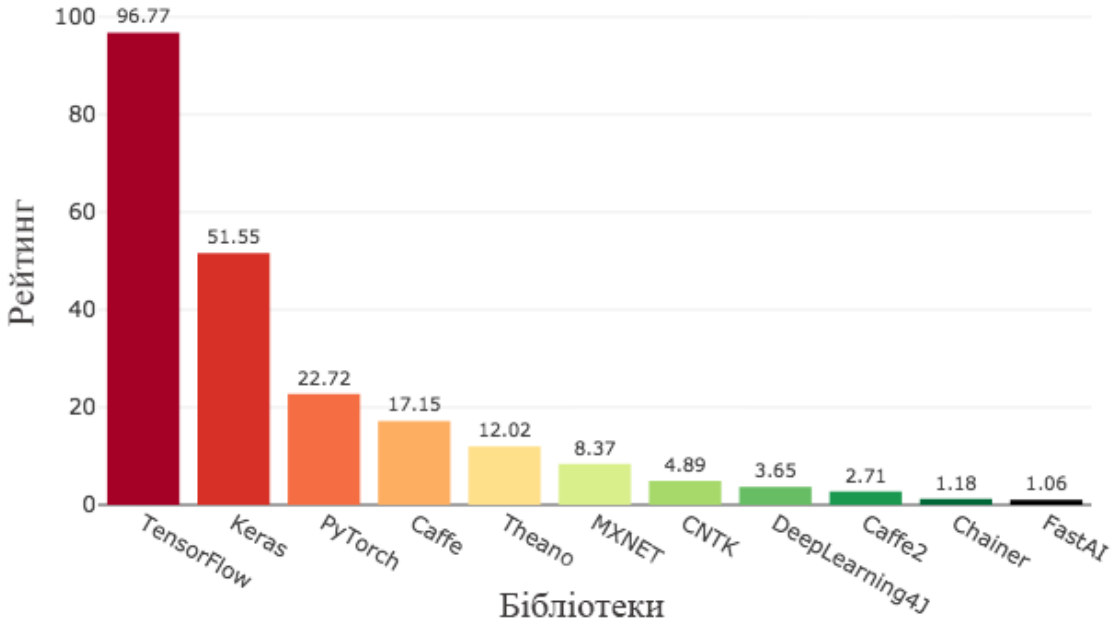

Рис. 2.7. Графік популярності бібліотек машинного навчання.

Отже, TensorFlow має високі стандарти щодо вимірювання ефективності та прозорості коду. Команда розробила набір детальних орієнтирів і дуже обережно включила їх в документацію. Розробники прислухаються та співпрацюють зі спільнотою. TensorFlow забезпечує чудову підтримку архітектури, що дозволяє

легко розгортати обчислення на широкому спектрі платформ, включаючи настільні комп'ютери, сервери та мобільні пристрої. Абстракція є головною перевагою TensorFlow щодо машинного навчання. Це дозволяє розробникам зосередитись на логіці програми замість того, щоб мати справу з подробицями впровадження алгоритмів.

2.2.2. Бібліотека Scikit-learn

Scikit-learn – це бібліотека машинного навчання загального призначення, написана на мові програмування Python, з широким спектром алгоритмів кластеризації, регресії та класифікації.. Вона забезпечує ефективну реалізацію найсучасніших алгоритмів, та багаторазового використання в наукових дисциплінах та сферах застосування [14]. Вона також використовує інтерактивність та модульність Python для забезпечення швидкого та простого прототипування. У Scikit-learn всі об'єкти та алгоритми приймають вхідні дані у вигляді двовимірних масивів. Об'єкти, які навчаються, поділяють між собою єдиний набір методів, який залежить від їх призначення: оцінки можуть вміщувати моделі з даних, прогнози можуть прогнозувати нові дані, а трансформатори перетворюють дані з одного представлення в інше [14].

Scikit-learn може взаємодіяти з науковими бібліотеками Python, такими як: NumPy та SciPy. Ця бібліотека підтримує як контрольовані, так і неконтрольовані ML. Ось перелік головних переваг Scikit-learn, що робить її потужною бібліотекою Python для машинного навчання:

- Алгоритми контрольованого та не контрольованого навчання.
- Регресія, включаючи лінійну та логістичну регресію.
- Класифікація, включаючи K-Найближчі сусіди.
- Кластеризація, включаючи K-засоби.
- Вибір моделі.
- Попередня обробка, включаючи нормалізацію Min-Max.

2.2.3. Бібліотека Keras

Keras (рис.2.8.) – провідна бібліотека Python з відкритим кодом, створена для побудови нейронних мереж та машинного навчання. Вона може працювати на Deeplearning4j, MXNet, Microsoft Cognitive Toolkit (CNTK), Theano або TensorFlow [15]. Keras пропонує майже всі самостійні модулі, включаючи оптимізатори, нейронні шари, функції активації, схеми ініціалізації, функції витрат та схеми регуляції [15]. Це дозволяє легко додавати нові модулі так само, як додавати нові функції та класи. Оскільки модель вже визначена в коді не потрібно мати окремі файли конфігурації моделі.

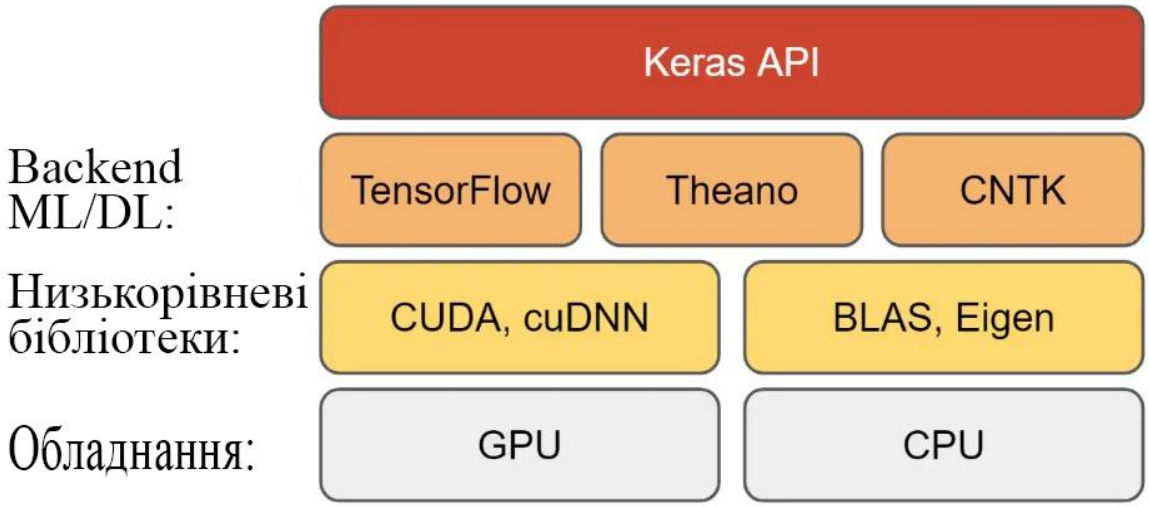

Рис. 2.8. Застосування бібліотеки Keras.

Keras спрощує навчання та проектування ШНМ та дозволяє легко розвивати нейронну мережу. Keras також вміє працювати з згортковими нейронними мережами. Бібліотека включає алгоритми для нормалізації, оптимізатора та шарів активації [15]. Замість того, щоб бути кінцевою бібліотекою машинного навчання Python, Keras функціонує як зручний, розширюваний інтерфейс, що підвищує модульність та гнучкість.

Висновки

У розділі розглянуто технології, що використовувались для розробки ШНМ для аналізу текстових даних. А саме: TensorFlow, Scikit-learn та Keras, що допоможуть у проектуванні для вирішення певної проблематики. Бібліотеки містять в собі всі необхідні функції та добре оптимізовані. Також бібліотеки обробки природної мови: SpaCy та TextBlob. За допомогою даних бібліотек буде проведене очищення вхідних даних та приведення їх до векторного виду, який зможуть зрозуміти нейронні мережі.

РОЗДІЛ 3. РОЗРОБКА ШТУЧНОЇ НЕЙРОННОЇ МЕРЕЖІ ДЛЯ АНАЛІЗУ ТЕКСТОВИХ ДАНИХ

Для розробки прикладної програми тональної класифікації текстових даних була вибрана мова програмування Python, операційна система Manjaro Linux 19.03. Бібліотеки Python, що використовувались:

- TensorFlow, для побудови та тренування нейронних мереж.
- Keras, як високорівневий API для TensorFlow.
- Scikit-learn, для класифікації та кластеризації.
- Бібліотеки NLP: SpaCy та Text Blob.
- Допоміжні бібліотеки: Pandas, Numpy.

Алгоритм розробки ШНМ можна поділити на 3 етапи:

1. Підготування даних для навчання та подальшого тестування. Надзвичайно важливо, щоб дані для навчання були підібрані правильно, та мали підходящий формат. За допомогою регулярних виразів та бібліотек NLP дані очистяться від непотрібних символів та будуть представлені у вигляді векторів.
2. Розробка нейронної мережі. У процесі розробки програми було використано 3 версії нейронних мереж: мережа прямого поширення, згорткова мережа, рекурентна мережа. Кожна мережа проектувалася за допомогою TensorFlow та Keras. У даних бібліотеках містяться всі необхідні функції оптимізації, втрат, та активації. А також всі необхідні методи для проектування та навчання нейронних мереж, за допомогою яких можна створити необхідну мережу. Код міститься в додатку до дипломної роботи.
3. Навчання та тестування нейронної мережі. Після того, як мережа спроектована, вона проходить навчання, в даному випадку 6 раз (6 епох), після чого вона проходить тестування на спеціально підготовлених даних, щоб можна було визначити точність з якою

вона їх обробляє. В результаті формуються графіки та мережа порівнюється і іншими типами ШНМ.

3.1. Вибір та підготовка набору даних

Запорукою машинного навчання є практика з найрізноманітніших проблем, від обробки зображень до розпізнавання та обробки природної мови. Кожна з цих проблем має свої унікальні підходи та нюанси, для вирішення яких потрібно мати високоякісні дані для навчання ШНМ. У Багатьох дослідницьких роботах використовуються власні набори даних, які зазвичай не публікуються у відкритому доступі для широкої громадськості.

Набори даних поділяються на три категорії: обробка зображень, обробка природних мов та обробка аудіо. Існує чимало способів використання цих наборів даних. Їх можна використовувати для застосування різних методів машинного навчання, щоб удосконалити свої навички, зрозуміти, як визначити та структурувати кожну проблему.

Для навчання штучних нейронних мереж у роботі буде використовуватись набір даних “IMDB Dataset” з сайту Kaggle (спільнота науковців даних та практиків машинного навчання). Набір зображений нижче:

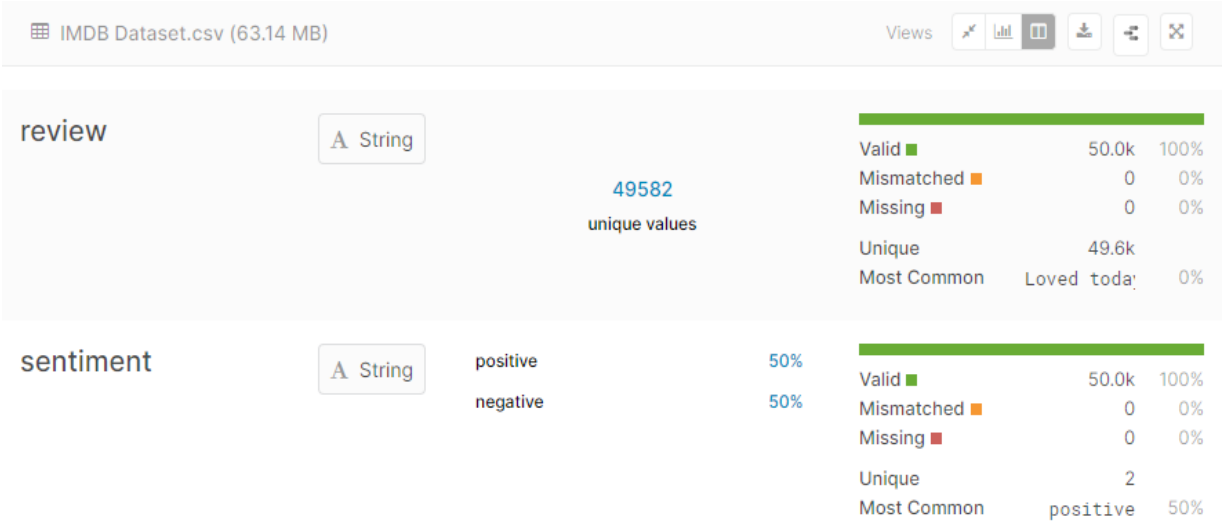

Рис. 3.1. Набір даних IMDB Dataset.

Набір має формат файлу CSV та містить 50 000 записів у дві колонки: огляд фільму та тональність. У колонці "Огляд" міститься текст рецензії користувача, а у колонці "тональність" знаходиться тип тональності огляду. Значення тональності може мати два значення, тобто "позитивне" та "негативне", що робить завдання бінарною проблемою класифікації. Перші 30 записів набору даних мають вигляд:

0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive
5	Probably my all-time favorite movie, a story o...	positive
6	I sure would like to see a resurrection of a u...	positive
7	This show was an amazing, fresh & innovative i...	negative
8	Encouraged by the positive comments about this...	negative
9	If you like original gut wrenching laughter yo...	positive
10	Phil the Alien is one of those quirky films wh...	negative
11	I saw this movie when I was about 12 when it c...	negative
12	So im not a big fan of Boll's work but then ag...	negative
13	The cast played Shakespeare. Shakes...	negative
14	This a fantastic movie of three prisoners who ...	positive
15	Kind of drawn in by the erotic scenes, only to...	negative
16	Some films just simply should not be remade. T...	positive
17	This movie made it into one of my top 10 most ...	negative
18	I remember this film,it was the first film i h...	positive
19	An awful film! It must have been up against so...	negative
20	After the success of Die Hard and it's sequels...	positive
21	I had the terrible misfortune of having to vie...	negative
22	What an absolutely stunning movie, if you have...	positive
23	First of all, let's get a few things straight ...	negative
24	This was the worst movie I saw at WorldFest an...	negative
25	The Karen Carpenter Story shows a little more ...	positive
26	"The Cell" is an exotic masterpiece, a dizzyin...	positive
27	This film tried to be too many things all at o...	negative
28	This movie was so frustrating. Everything seem...	negative
29	'War movie' is a Hollywood genre that has been...	positive

Рис. 3.2. Вигляд 30 перших рядків набору даних.

Зазвичай у даному наборі даних рецензія користувача до відповідного фільму не перевищує 150-200 слів. Повний відгук під індексом 2 та тональністю негативний, показаний на рис. 3.3. Як можна помітити в рецензії наявні

пунктуація, та різноманітні теги (

). Тому потрібно підготувати всі вхідні дані до обробки нейронною мережею. Оскільки розділові знаки, цифри та

HTML-теги не відіграють ніякого значення в тональній класифікації тексту тому їх потрібно очистити.

```
I thought this was a wonderful way to spend time on a too hot summer weekend, sitting
 in the air conditioned theater and watching a light-hearted comedy. The plot is simp
listic, but the dialogue is witty and the characters are likable (even the well bread
 suspected serial killer). While some may be disappointed when they realize this is n
ot Match Point 2: Risk Addiction, I thought it was proof that Woody Allen is still fu
lly in control of the style many of us have grown to love.<br /><br />This was the mo
st I'd laughed at one of Woody's comedies in years (dare I say a decade?). While I've
 never been impressed with Scarlet Johanson, in this she managed to tone down her "se
xy" image and jumped right into a average, but spirited young woman.<br /><br />This
may not be the crown jewel of his career, but it was wittier than "Devil Wears Prada"
 and more interesting than "Superman" a great comedy to go see with friends.
```

Рис. 3.3. Приклад негативної рецензії з індексом №2.

Один із найкращих способів видалити непотрібні символи - регулярні вирази (regex або їх ще називають - regular expression). Регулярні вирази – це послідовність символів, що співпадає з певним шуканим паттерном. Вони підтримуються великою кількістю текстових редакторів та багатьма мовами програмування, зокрема мовою Python. Отже, алгоритм дій для очищення рецензій за допомогою regex:

- Вилучити всі інші символи крім символів верхнього та нижнього регістру. Регулярний вираз, що використовувався : “ [^a-zA-Z] “.
- Наступний крок – видалення всіх одинарних символів, вони не мають важливої інформації, тому ними можна знехтувати. Вираз, що використовувався: “ \s+[a-zA-Z]\s+ “.
- Видалення декількох символів відступу підряд, за допомогою виразу: “ \s+ ”.
- Останній найбільш важливий крок – видалення HTML-тегів, яке виконувалося виразом: “ <[^>]+> “.

Після очищення даних, рецензія з індексом №2 буде мати вигляд:

```
I thought this was wonderful way to spend time on too hot summer weekend sitting in t
he air conditioned theater and watching light hearted comedy The plot is simplistic b
ut the dialogue is witty and the characters are likable even the well bread suspected
 serial killer While some may be disappointed when they realize this is not Match Poi
nt Risk Addiction thought it was proof that Woody Allen is still fully in control of
the style many of us have grown to love This was the most d laughed at one of Woody c
omedies in years dare say decade While ve never been impressed with Scarlet Johanson
in this she managed to tone down her sexy image and jumped right into average but spi
rited young woman This may not be the crown jewel of his career but it was wittier th
an Devil Wears Prada and more interesting than Superman great comedy to go see with f
riends
```

Рис. 3.4. Приклад очищеної рецензії №2.

Отже, після очищення вхідних даних, зайві символи були видалені у всіх рецензіях. Весь тест за допомого бібліотек NLP буде перетворено у векторний формат. І тепер на кінець потрібно перетворити стовпець “Тональність” з рядка у число, тобто, позитивна рецензія – 1, негативна – 0. Графік тональності відгуків до фільмів буде мати такий вигляд:

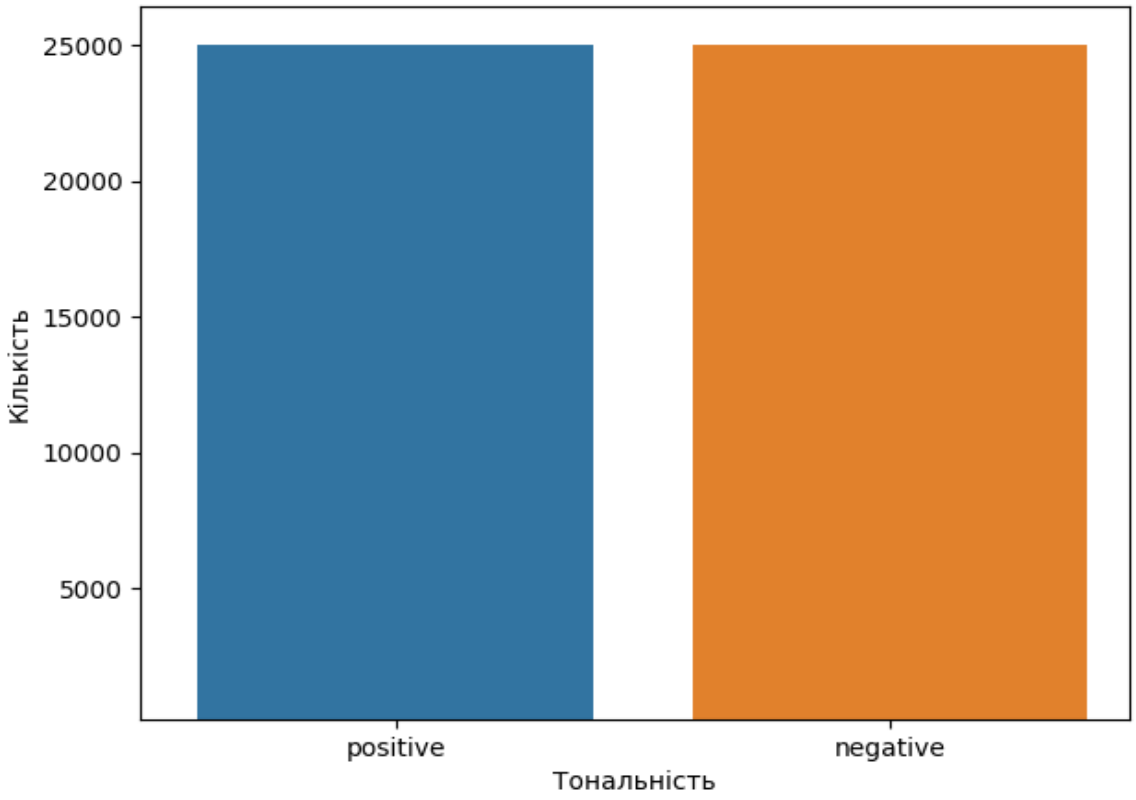

Рис. 3.5. Графік тональності відгуків набору даних IMDB Dataset.

3.2. Побудова програми на основі MLP

У даній версії програми використовується звичайна нейронна – багатошаровий перцептрон (рис. 3.6.). Для оптимізації застотовується оптимізатор adam та binary_crossentropy, як функція втрати, функція активації - сигмоїдна.

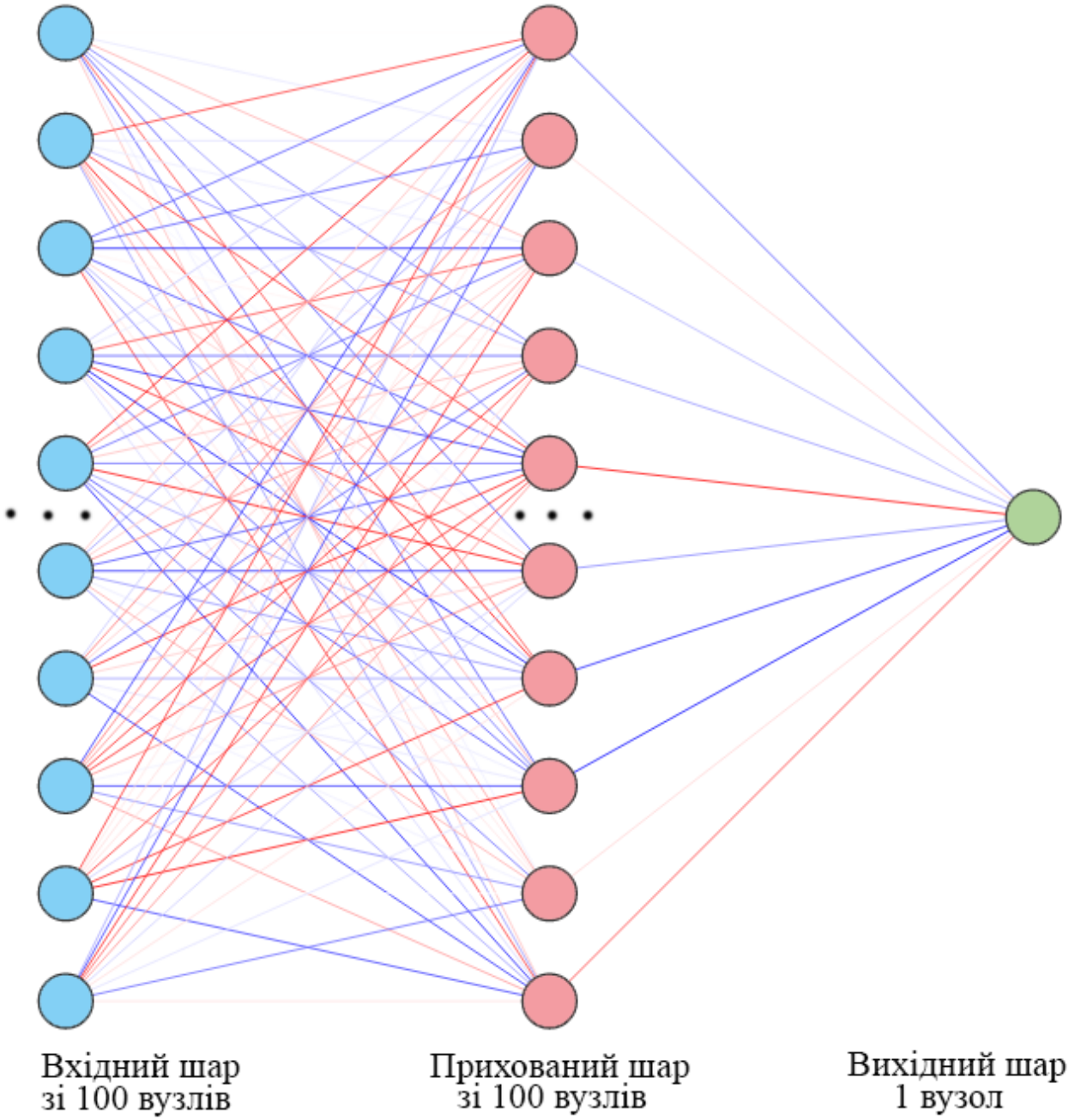

Рис. 3.6. Архітектура MLP.

Варто звернути увагу, що тренування проходить лише на тренувальному наборі даних, тобто 80% від початкового набору даних. Перевірка точності мережі відбувається на решті 20% даних, які не використовувались для тренування. Всі графіки формуються на основі результатів роботи бібліотеки TensorFlow. Після проходження навчання мережі та проходження тестування точності, мережа показала наступні результати:

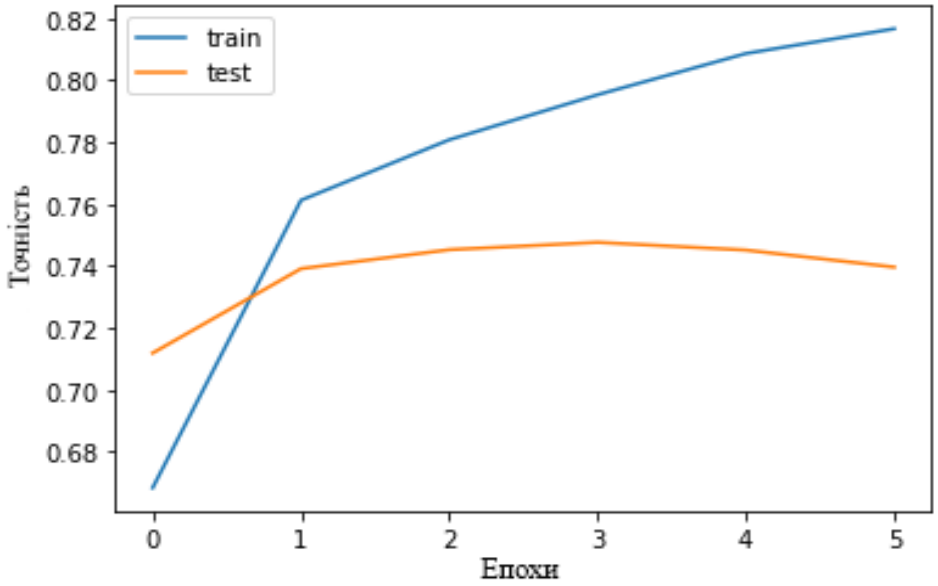

Рис. 3.7. Точність MLP.

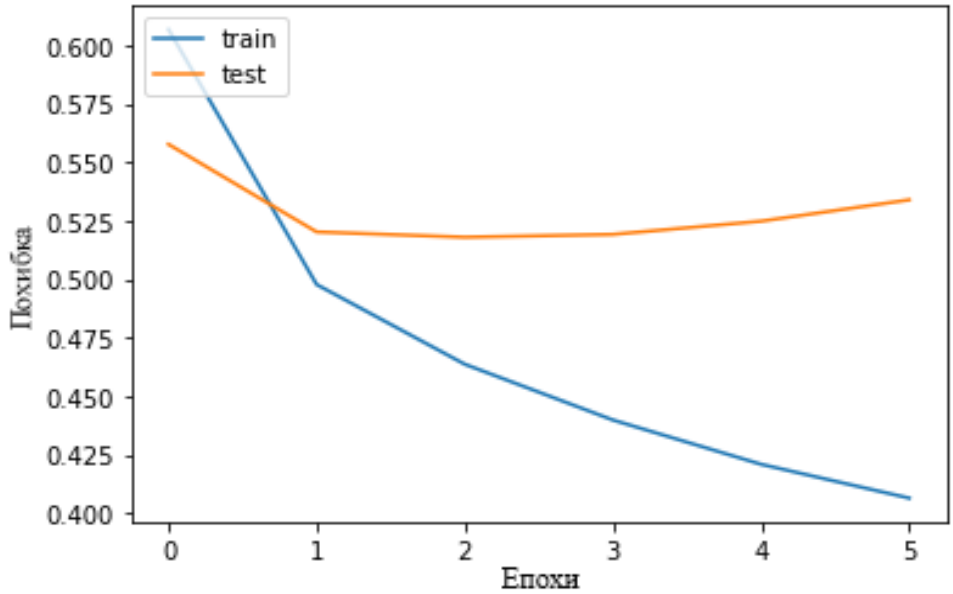

Рис. 3.8. Похибка MLP.

Отже під час навчання на тренувальних даних мережа показала результат в 82% точності, в свою чергу під час тестування на даних, які мережа ніколи не бачила, вона змогла набрати – 74% правильних відповідей.

3.2. Побудова програми на основі CNN

Згорткова нейронна мережа – це тип мережі, який в основному використовується для двовимірної класифікації даних, наприклад зображень. Згорткова мережа намагається знайти конкретні риси в зображенні на першому шарі. У наступних шарах спочатку виявлені ознаки об’єднуються разом, щоб утворити групи ознак. Таким чином обробляється все зображення. Було досліджено, що згорткові нейронні мережі добре працюють і з текстовими даними також.

Функція активації першого шару – ReLU, другого шару – сигмоїдна. Для оптимізації використовувався алгоритм adam, функція втрат - binary_crossentrop. Архітектура згорткової нейронної мережі зображена нижче:

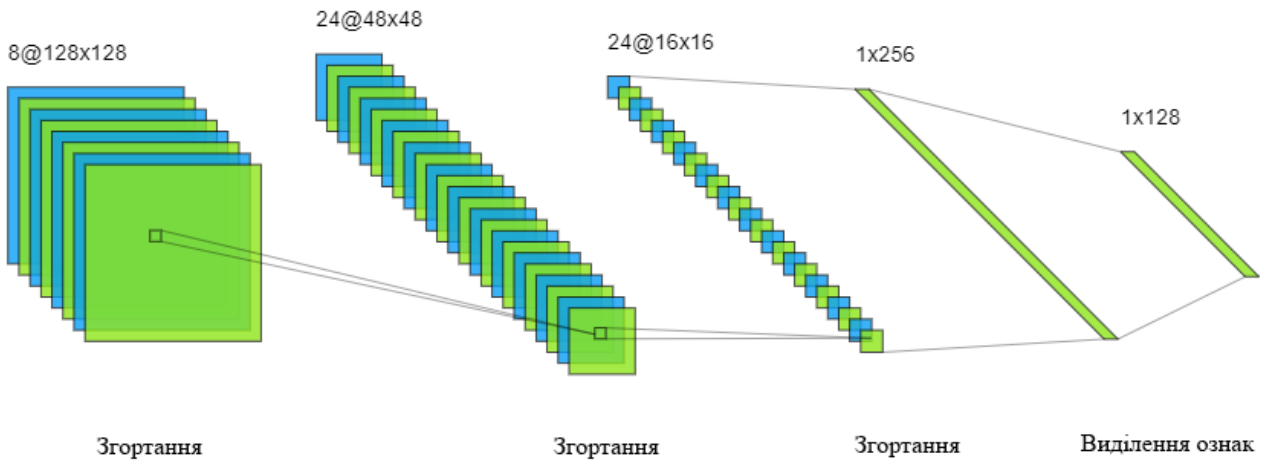

Рис. 3.9. Архітектура згорткової мережі.

Якщо порівнювати точність тренувань і тестів, можна побачити, що точність тренувань для згорткової нейронної мережі буде приблизно 92%, що

більше, ніж точність тренувань простої нейронної мережі. Точність тесту становить близько 84% для CNN, що також значно перевищує точність тесту для простої нейронної мережі, яка становила близько 74%. Однак модель CNN все ще є неточною, оскільки існує велика різниця між навчанням та результатом тесту. Графіки результату навчання та тестування:

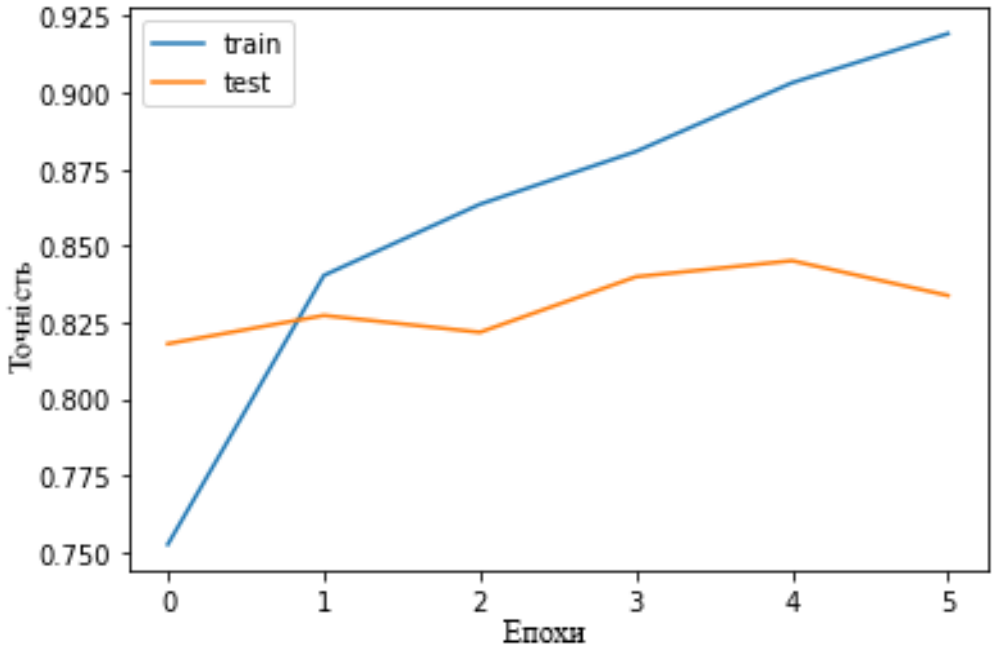

Рис. 3.10. Точність CNN.

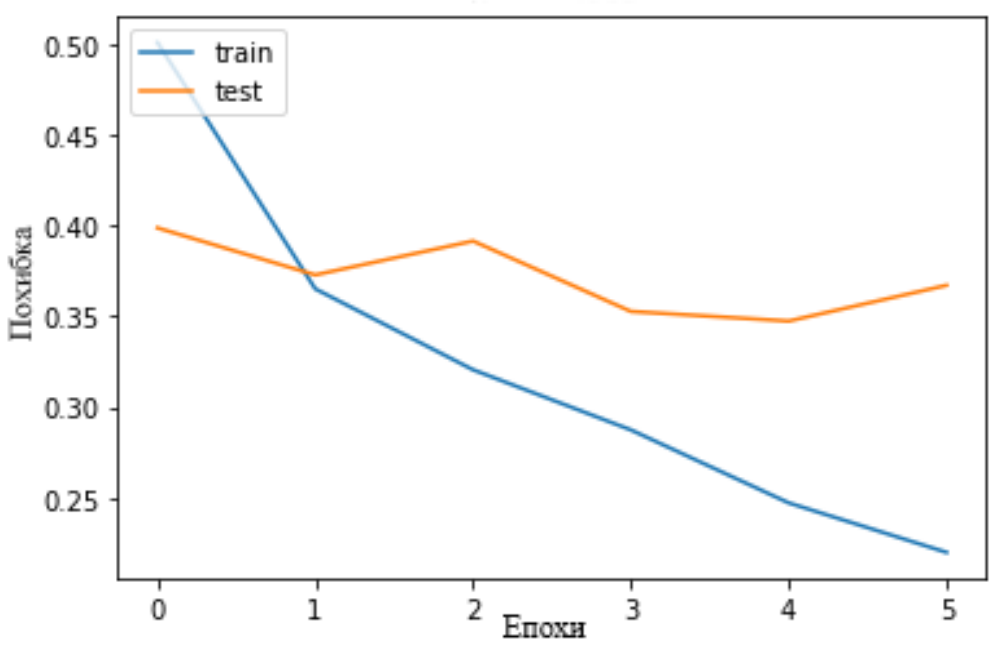

Рис. 3.11. Похибка CNN.

3.2. Побудова програми на основі RNN

Рекурентна нейронна мережа – це тип нейронних мереж, який, як відомо, добре працює з послідовними даними. Оскільки текст насправді є послідовністю слів, рекурентна нейронна мережа є непоганим вибором для вирішення проблем пов'язаних з текстом.

Дана мережа має 128 вхідних да 128 прихованих шарів. Для оптимізації використовувався алгоритм adam, функція втрат - binary_crossentrop, функція активації - сигмоїдна. Архітектура рекурентної мережі зображена нижче:

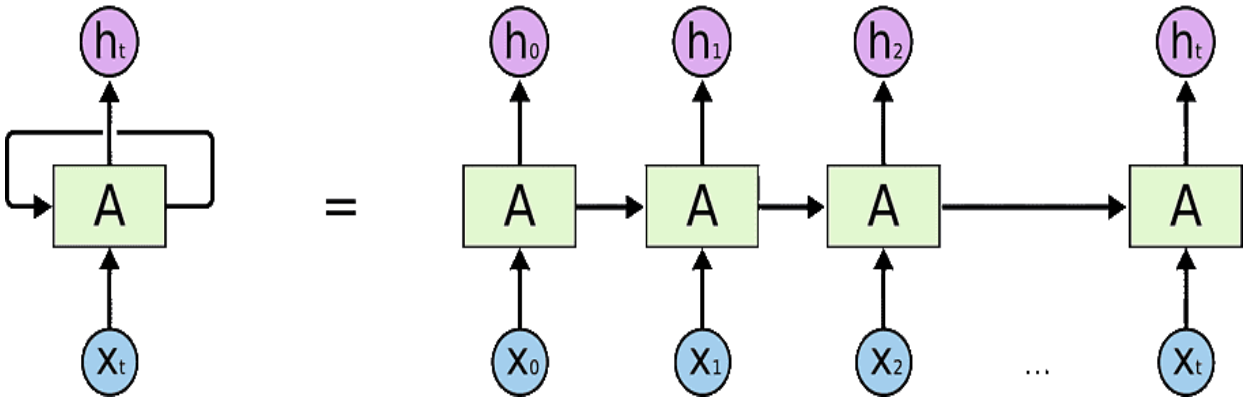

Рис. 3.12. Архітектура рекурентної нейронної мережі,

де: x_t - вхідні вузли, A - приховані вузли, h_t – вихідні вузли, t = 128.

У результаті тестування (рис. 3.13. та рис 3.14.) видно, що тестова точність становить близько 85%, так само, як і точність при навчанні. Точність тесту вища, ніж у згорткової нейронної мережі, та значно випереджає у точності звичайну нейронну мережу. Крім того, можна помітити, що між точністю тренування і точністю тесту дуже мала різниця, що означає, що модель не є надмірно навченою, тобто її ваги розставлені вірно.

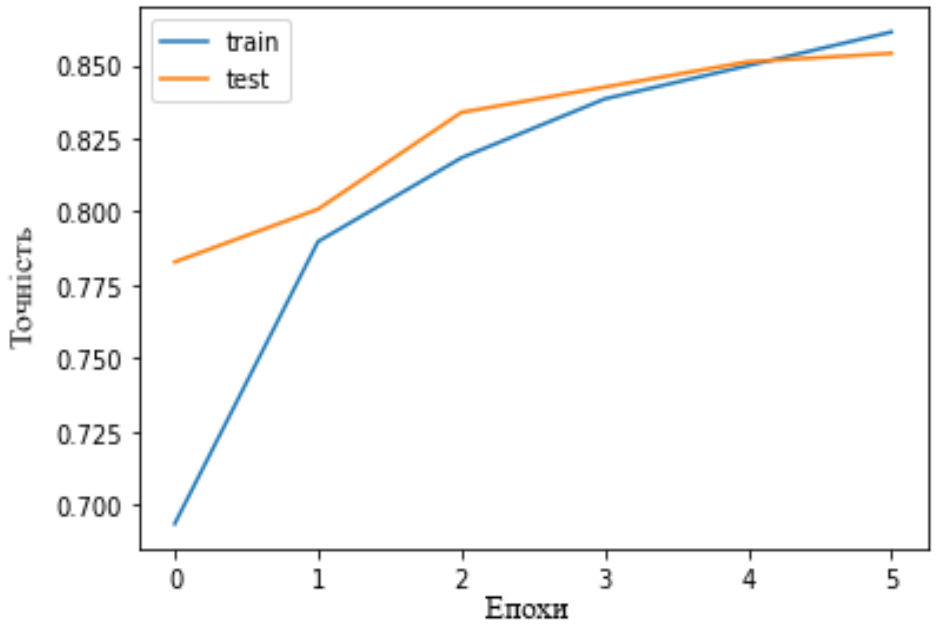

Рис. 3.13. Точність RNN.

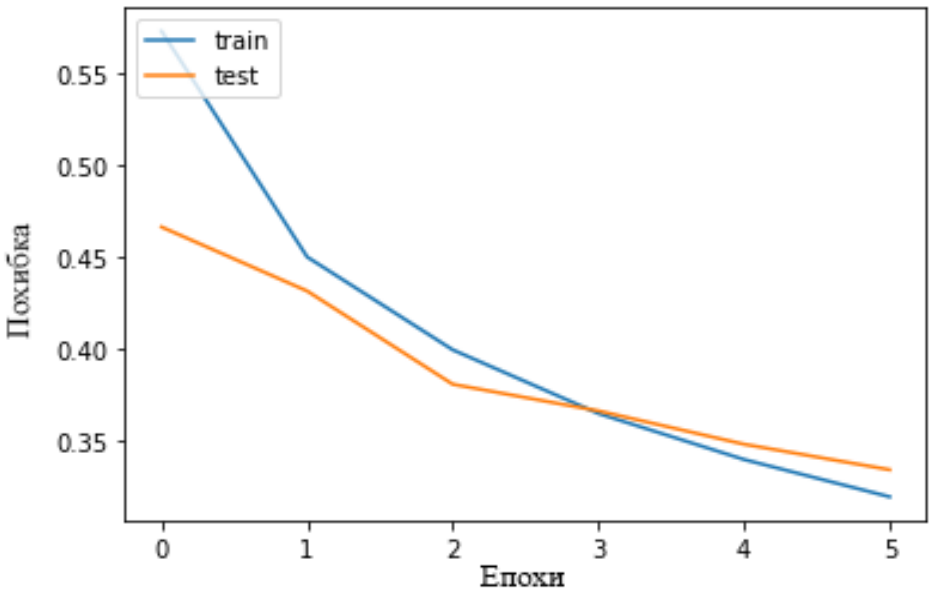

Рис. 3.14. Похибка RNN.

Висновки

Результати тестування показують, що найкраще справилась рекурентна нейронна мережа. ЇЇ точність класифікації становить близько 85%, як під час навчання, так і під час тестування, що є досить непоганим результатом, враховуючи не надто великий набір даних для навчання. Трохи гірше себе

показала згорткова мережа, яка під час навчання підтвердила точність 92%, а під час перевірки 84%. Така велика розбіжність вказує на те, що ШНМ не змогла добре підібрати параметри і точність на справжніх рецензіях може бути значно меншою. Найгірший результат у мережі прямого поширення – 82% при навчанні та 74% під час перевірки, показуючи такі ж проблеми як і в згорткової мережі, лише в більшому масштабі.

Отже, як видно з тестування для виконання завдання класифікації текстових даних найкраще себе показала рекурентна нейронна мережа, яку варто і потрібно використовувати для завдань такого типу. Згорткову мережу також можна застосувати для таких завдань, але потрібно більше даних для тренування. Звичайну нейронну мережу краще не застосовувати в такому випадку і віддати перевагу двом попереднім.

ВИСНОВКИ

Класифікація тексту – одне з найпоширеніших завдань з обробки текстових даних. У кваліфікаційній роботі розглянуто, як можна вирішити таке завдання, використовуючи три різні типи нейронних мереж, щоб класифікувати тональність рецензій щодо різних фільмів.

В роботі розглянуто історію створення, різноманітні архітектури та способи навчання штучних нейронних мереж. Зроблено огляд сучасних технологій для проектування та навчання ШНМ: TensorFlow, Scikit-learn, Keras.Також бібліотеки обробки природної мови: SpaCy та Text Blob. Показано їх способи використання, переваги та недоліки.

В результаті розроблено нейронну мережу для тональної класифікації текстових даних, на основі трьох архітектур нейронних мереж. Проведено тестування версій програми та проаналізовано результати. Вказано на недоліки використання тієї чи іншої мережі.

Можна сказати, що аналіз та обробка неструктурованих текстових даних надзвичайно важливий аспект сучасного світу, що дозволить оптимізувати та покращити десятки галузей, починаючи від медицини та закінчуючи освітою. Як показало тестування з даним завданням найкраще справляються рекурентні нейронні мережі.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. McCulloch, W. Pitts. Bulletin of Mathematical Biophysics. V. 5. Illinois, pp. 110-135. (1943)
2. Hebb, D.O. The Organization of Behavior: A Neuropsychological Theory. New York, NY: John Wiley & Sons, pp. 80-96 (1949).
3. M. L. Minsky and S. A. PAPERT, Perceptrons. MIT Press, Cambridge, MA. Expanded version of the original 1968 edition, pp. 40-55 (1988).
4. Kohonen, Teuvo. "Self-Organized Formation of Topologically Correct Feature Maps". Biological Cybernetics, pp. 59–69 (1982).
5. Memisevic, R. and Hinton, G. E. Unsupervised learning of image transformations. Computer Vision and Pattern Recognition, pp. 29-38 (2007).
6. LeCun, Yann; Léon Bottou; Yoshua Bengio; Patrick Haffner "Gradient-based learning applied to document recognition". Proceedings of the IEEE, pp. 82-86 (1998).
7. Goller, Christoph; Küchler, Andreas. Learning task-dependent distributed representations by backpropagation through structure, pp. 120-128 (1996).
8. A. Graves, M. Liwicki, S. Fernandez, R. Bertolami, H. Bunke, J. Schmidhuber. A Novel Connectionist System for Improved Unconstrained Handwriting Recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 31, no. 5, pp. 34-48 (2009).
9. Mandic, Danilo P. & Chambers, Jonathon A. Recurrent Neural Networks for Prediction: Learning Algorithms, Architectures and Stability. Wiley, pp. 50-54 (2001).
10. Mohri, Mehryar; Rostamizadeh, Afshin; Talwalkar, Ameet, Foundations of Machine Learning. The MIT Press, pp. 137-142 (2012).
11. Документація бібліотеки SpaCy [Електронний ресурс] – Режим доступу: https://spacy.io/usage/spacy-101

12. Документація бібліотеки Text Blob [Електронний ресурс] – Режим доступу: https://textblob.readthedocs.io/en/dev/quickstart.html#quickstart
13. Документація бібліотеки TensorFlow [Електронний ресурс] – Режим доступу: https://www.tensorflow.org/overview
14. Документація бібліотеки Scikit-learn [Електронний ресурс] – Режим доступу: https://scikit-learn.org/stable/user_guide.html
15. Бібліотека Keras [Електронний ресурс] – Режим доступу: https://keras.io/

```
import plac
import random
import pathlib
import cytoolz
import numpy
from keras.models import Sequential, model_from_json
from keras.layers import LSTM, Dense, Embedding, Bidirectional
from keras.layers import TimeDistributed
from keras.optimizers import Adam
import thinc.extra.datasets
from spacy.compat import pickle
import spacy


class SentimentAnalyser(object):
    @classmethod
    def load(cls, path, nlp, max_length=100):
        with (path / "config.json").open() as file_:
            model = model_from_json(file_.read())
        with (path / "model").open("rb") as file_:
            lstm_weights = pickle.load(file_)
        embeddings = get_embeddings(nlp.vocab)
        model.set_weights([embeddings] + lstm_weights)
        return cls(model, max_length=max_length)

    def __init__(self, model, max_length=100):
        self._model = model
        self.max_length = max_length

    def __call__(self, doc):
        X = get_features([doc], self.max_length)
        y = self._model.predict(X)
        self.set_sentiment(doc, y)

    def pipe(self, docs, batch_size=1000):
        for minibatch in cytoolz.partition_all(batch_size, docs):
            minibatch = list(minibatch)
            sentences = []
            for doc in minibatch:
                sentences.extend(doc.sents)
            Xs = get_features(sentences, self.max_length)
            ys = self._model.predict(Xs)
            for sent, label in zip(sentences, ys):
                sent.doc.sentiment += label - 0.5
            for doc in minibatch:
                yield doc

    def set_sentiment(self, doc, y):
        doc.sentiment = float(y[0])


def get_labelled_sentences(docs, doc_labels):
    labels = []
    sentences = []
    for doc, y in zip(docs, doc_labels):
        for sent in doc.sents:
```

```
            sentences.append(sent)
            labels.append(y)
    return sentences, numpy.asarray(labels, dtype="int32")


def get_features(docs, max_length):
    docs = list(docs)
    Xs = numpy.zeros((len(docs), max_length), dtype="int32")
    for i, doc in enumerate(docs):
        j = 0
        for token in doc:
            vector_id = token.vocab.vectors.find(key=token.orth)
            if vector_id >= 0:
                Xs[i, j] = vector_id
            else:
                Xs[i, j] = 0
            j += 1
            if j >= max_length:
                break
    return Xs


def train(
    train_texts,
    train_labels,
    dev_texts,
    dev_labels,
    lstm_shape,
    lstm_settings,
    lstm_optimizer,
    batch_size=100,
    nb_epoch=5,
    by_sentence=True,
):

    print("Loading spaCy")
    nlp = spacy.load("en_vectors_web_lg")
    nlp.add_pipe(nlp.create_pipe("sentencizer"))
    embeddings = get_embeddings(nlp.vocab)
    model = compile_lstm(embeddings, lstm_shape, lstm_settings)

    print("Parsing texts...")
    train_docs = list(nlp.pipe(train_texts))
    dev_docs = list(nlp.pipe(dev_texts))
    if by_sentence:
        train_docs, train_labels = get_labelled_sentences(train_docs,
train_labels)
        dev_docs, dev_labels = get_labelled_sentences(dev_docs, dev_labels)

    train_X = get_features(train_docs, lstm_shape["max_length"])
    dev_X = get_features(dev_docs, lstm_shape["max_length"])
    model.fit(
        train_X,
        train_labels,
        validation_data=(dev_X, dev_labels),
        epochs=nb_epoch,
        batch_size=batch_size,
    )
    return model
```

```
def compile_lstm(embeddings, shape, settings):
    model = Sequential()
    model.add(
        Embedding(
            embeddings.shape[0],
            embeddings.shape[1],
            input_length=shape["max_length"],
            trainable=False,
            weights=[embeddings],
            mask_zero=True,
        )
    )
    model.add(TimeDistributed(Dense(shape["nr_hidden"], use_bias=False)))
    model.add(
        Bidirectional(
            LSTM(
                shape["nr_hidden"],
                recurrent_dropout=settings["dropout"],
                dropout=settings["dropout"],
            )
        )
    )
    model.add(Dense(shape["nr_class"], activation="sigmoid"))
    model.compile(
        optimizer=Adam(lr=settings["lr"]),
        loss="binary_crossentropy",
        metrics=["accuracy"],
    )
    return model


def get_embeddings(vocab):
    return vocab.vectors.data


def evaluate(model_dir, texts, labels, max_length=100):
    nlp = spacy.load("en_vectors_web_lg")
    nlp.add_pipe(nlp.create_pipe("sentencizer"))
    nlp.add_pipe(SentimentAnalyser.load(model_dir, nlp, max_length=max_length))

    correct = 0
    i = 0
    for doc in nlp.pipe(texts, batch_size=1000):
        correct += bool(doc.sentiment >= 0.5) == bool(labels[i])
        i += 1
    return float(correct) / i


def read_data(data_dir, limit=0):
    examples = []
    for subdir, label in (("pos", 1), ("neg", 0)):
        for filename in (data_dir / subdir).iterdir():
            with filename.open() as file_:
                text = file_.read()
            examples.append((text, label))
    random.shuffle(examples)
    if limit >= 1:
```

```
        examples = examples[:limit]
    return zip(*examples)


@plac.annotations(
    train_dir=("Location of training file or directory"),
    dev_dir=("Location of development file or directory"),
    model_dir=("Location of output model directory",),
    is_runtime=("Demonstrate run-time usage", "flag", "r", bool),
    nr_hidden=("Number of hidden units", "option", "H", int),
    max_length=("Maximum sentence length", "option", "L", int),
    dropout=("Dropout", "option", "d", float),
    learn_rate=("Learn rate", "option", "e", float),
    nb_epoch=("Number of training epochs", "option", "i", int),
    batch_size=("Size of minibatches for training LSTM", "option", "b", int),
    nr_examples=("Limit to N examples", "option", "n", int),
)
def main(
    model_dir=None,
    train_dir=None,
    dev_dir=None,
    is_runtime=False,
    nr_hidden=64,
    max_length=100,
    dropout=0.5,
    learn_rate=0.001,
    nb_epoch=5,
    batch_size=256,
    nr_examples=-1,
):
    if model_dir is not None:
        model_dir = pathlib.Path(model_dir)
    if train_dir is None or dev_dir is None:
        imdb_data = thinc.extra.datasets.imdb()
    if is_runtime:
        if dev_dir is None:
            dev_texts, dev_labels = zip(*imdb_data[1])
        else:
            dev_texts, dev_labels = read_data(dev_dir)
        acc = evaluate(model_dir, dev_texts, dev_labels, max_length=max_length)
        print(acc)
    else:
        if train_dir is None:
            train_texts, train_labels = zip(*imdb_data[0])
        else:
            print("Read data")
            train_texts, train_labels = read_data(train_dir, limit=nr_examples)
        if dev_dir is None:
            dev_texts, dev_labels = zip(*imdb_data[1])
        else:
            dev_texts, dev_labels = read_data(dev_dir, imdb_data,
limit=nr_examples)
        train_labels = numpy.asarray(train_labels, dtype="int32")
        dev_labels = numpy.asarray(dev_labels, dtype="int32")
        lstm = train(
            train_texts,
            train_labels,
            dev_texts,
            dev_labels,
```

```
            {"nr_hidden": nr_hidden, "max_length": max_length, "nr_class": 1},
            {"dropout": dropout, "lr": learn_rate},
            {},
            nb_epoch=nb_epoch,
            batch_size=batch_size,
        )
        weights = lstm.get_weights()
        if model_dir is not None:
            with (model_dir / "model").open("wb") as file_:
                pickle.dump(weights[1:], file_)
            with (model_dir / "config.json").open("w") as file_:
                file_.write(lstm.to_json())


if __name__ == "__main__":
    plac.call(main)
```