НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ
УНІВЕРСИТЕТ УКРАЇНИ “КИЇВСЬКИЙ
ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ
ІГОРЯ СІКОРСЬКОГО” НАВЧАЛЬНО-
НАУКОВИЙ КОМПЛЕКС
"ІНСТИТУТ ПРИКЛАДНОГО
СИСТЕМНОГО АНАЛІЗУ"
КАФЕДРА СИСТЕМ АВТОМАТИЗОВАНОГО ПРОЕКТУВАННЯ

«До захисту допущено»

Завідувач кафедри

(підпис) Мухін В. Є.

“___”_____2022 р.

Дипломний проект

на здобуття ступеня бакалавра

зі спеціальності (спеціалізації): 122 Комп'ютерні науки

на тему: “Алгоритми та програмна реалізація на мові Python методів глибокого навчання для класифікації документів та часових послідовностей”

Виконав: студент 4 курсу, групи ДА-82, Зайцев Кирило Юрійович (підпис)

Керівник: проф., д.т.н, Рогоза В. С. (підпис)

Консультант: (назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент: (посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному

проекті немає запозичень з праць

інших авторів без відповідних посилань.

Студент (підпис)

Київ – 2022

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ
УНІВЕРСИТЕТ УКРАЇНИ “КИЇВСЬКИЙ
ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ
ІГОРЯ СІКОРСЬКОГО” НАВЧАЛЬНО-
НАУКОВИЙ КОМПЛЕКС
"ІНСТИТУТ ПРИКЛАДНОГО
СИСТЕМНОГО АНАЛІЗУ"
КАФЕДРА СИСТЕМ АВТОМАТИЗОВАНОГО ПРОЕКТУВАННЯ

Рівень вищої освіти – перший (бакалаврський)

Спеціальність (спеціалізація) 122 Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри
(підпис) Мухін В. Є.
«___»_____2022 р.

ЗАВДАННЯ

на дипломний проект (роботу) студенту

Зайцеву Кирилу Юрійовичу

1. Тема проекту (роботи): “Алгоритми та програмна реалізація на мові Python методів глибокого навчання для класифікації документів та часових послідовностей”.
Керівник проекту (роботи): Рогоза В. С., проф., д.т.н.,
затверджені наказом по університету від «___»_____ 20__ р. №_____
2. Строк подання студентом проекту (роботи)

3. Вихідні дані до проекту (роботи)
"Review of deep learning: concepts, architectures, challenges, applications, Springer Journal of Big Data"; "Deep learning for time series classification: a review, Springer Nature whitepaper"; "GloVe: Global Vectors for Word Representation, Stanford whitepaper"; "Attention Is All You Need, NIPS

whitepaper"; "Universal Language Model Fine-tuning for Text Classification, ACL 2018"; "LSTM Fully Convolutional Networks for Time Series Classification, IEEE whitepaper"; "A recent overview of the state-of-the-art elements of text classification, Expert Systems with Applications Journal"; “Large-Scale Arabic Text Classification Using MapReduce , ACIT whitepaper“.

4. Зміст (дипломної роботи) пояснювальної записки (перелік завдань, які потрібно розробити)

1) Ознайомитись з методами глибокого машинного навчання.
2) Дослідити особливості використання нейронних мереж для вирішення задач класифікації документів та часових послідовностей.
3) Провести аналіз сучасних досліджень у напрямку класифікації документів та часових послідовностей.
4) Програмно реалізувати низку архітектур нейронних мереж та проаналізувати результати їх застосування у задачах класифікації тексту та часових послідовностей.
5) Розглянути парадигму MapReduce як спосіб оптимізації швидкодії обчислень на прикладі використання глибоких нейронних мереж у задачах класифікації тексту.

5. Перелік графічного (ілюстративного) матеріалу (із зазначенням обов’язкових креслеників, плакатів, презентацій тощо)

6. Консультанти розділів проекту (роботи)

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

7. Дата видачі завдання

Календарний план

№ з/п	Назва етапів виконання дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Ознайомитись з методами глибокого машинного навчання	02-06.05.22	
2	Дослідити особливості використання нейронних мереж для вирішення задач класифікації документів та часових послідовностей	09-13.05.22	
3	Провести аналіз сучасних досліджень у напрямку класифікації документів та часових послідовностей	16-20.05.22	
4	Програмно реалізувати низку архітектур нейронних мереж та проаналізувати результати їх застосування у задачах класифікації тексту та часових послідовностей	23-27.05.22	

5	Розглянути парадигму MapReduce як спосіб оптимізації швидкодії обчислень на прикладі використання глибоких нейронних мереж у задачах класифікації тексту	30.05.22-03.06.22

Студент: Зайцев К. Ю. (підпис)

Керівник проекту (роботи): Рогоза В. С. (підпис)

ЗМІСТ

Терміни і скорочення у тексті роботи....8
Вступ....9
1 Застосування методів глибокого навчання у задачах класифікації документів та часових послідовностей....11
1.1 Основні поняття теорії глибокого машинного навчання....11
1.1.1 Застосування машинного навчання та нейронних мереж у сучасному світі....11
1.1.2 Обґрунтування переходу до глибокого навчання....12
1.1.3 Огляд ключових елементів теорії глибокого навчання....14
1.1.3.1 Глибока нейронна мережа....14
1.1.3.2 Нелінійна активація нейронів....15
1.1.3.3 Зворотне поширення похибки....16
1.1.3.4 Оптимізаційний процес. Алгоритм стохастичного градієнтного спуску....16
1.2 Аналіз сучасних архітектур нейронних мереж....18
1.2.1 Загальна таксономія алгоритмів ГН....18
1.2.2 Класичні методи побудови нейронних мереж для задач класифікації 19
1.2.2.1 Багатошаровий перцептрон....19
1.2.2.2 Згорткова нейронна мережа....21
1.2.2.3 RNN....22
1.2.2.3.1 Особливості роботи RNN у задачах з послідовною структурою даних....24
1.2.2.3.2 Сучасна варіація рекурентної архітектури....27
1.2.3 Універсальні НМ....30
1.3 Огляд задач класифікації даних з послідовною структурою....32
1.3.1 Класифікація текстових даних....33
1.3.2 Класифікація часових послідовностей....33
1.4 Вибір архітектур нейронних мереж для класифікації документів та часових послідовностей....34
2 Порівняльний аналіз методів глибокого навчання на деяких наборах даних 36
2.1 Вибір наборів даних для експериментів....36
2.1.1 Текстові набори даних....36
2.1.2 Датасети часових рядів....37

2.2 Цільові метрики та вибір порівнюваних характеристик процесу навчання обраних алгоритмів....38
2.3 Результати порівняльного аналізу....39
2.3.1 Результати класифікації текстових даних....39
2.3.2 Результати класифікації часових послідовностей....41
2.4 Особливості реалізації обраних архітектур нейронних мереж. .42
3 Прискорення роботи моделей глибокого навчання засобами розподілених обчислень на прикладі парадигми MapReduce....48
3.1 Пришвидшення роботи алгоритмів глибокого навчання....48
3.2 Основні поняття розподілених обчислень. Парадигма MapReduce....51
3.3 Оптимізація швидкості класифікації текстових даних за допомогою MapReduce....53
4 Функціонально-вартісний аналіз програмного продукту....57
4.1 Постановка задачі техніко-економічного аналізу....58
4.1.1 Обґрунтування функцій програмного продукту....58
4.1.2 Варіанти реалізації основних функцій....59
4.2 Обґрунтування системи параметрів програмного продукту....62
4.2.1 Опис параметрів....62
4.2.2 Кількісна оцінка параметрів....62
4.2.3 Аналіз експертного оцінювання параметрів....65
4.3 Аналіз рівня якості варіантів реалізації функцій....70
4.4 Економічний аналіз варіантів розробки програмного продукту72
4.5 Висновки....76
Висновки....78
Перелік джерел посилання....80
Додаток А....85
Деталі алгоритму GloVe....85
Додаток Б Архітектура мережі класу “Трансформер”....87
Додаток В Реалізація трансформеру, рекурентної та згорткової мереж

для класифікації часових рядів у PyTorch...91

ТЕРМІНИ І СКОРОЧЕННЯ У ТЕКСТІ РОБОТИ

ML (англ. Machine learning) – напрям в області штучного інтелекту, який займається статистичними методами аналізу даних.

DL (англ. Deep learning) – клас методів машинного навчання, що використовують глибокі нейронні мережі.

Датасет — набір прикладів, що складаються з ознак та, у випадку навчання з вчителем, міток.

ГН — глибоке навчання.

НМ — нейронна мережа.

Токен — цілочисельне представлення слова.

Ембедінг (англ. embedding) — векторна репрезентація слова.

Гіперпараметр (англ. hyperparameter) — параметр, який задається перед початком оптимізації та визначає роботу деякої складової оптимізаційного процесу.

ВСТУП

Класифікація даних з послідовною структурою є важливою і складною проблемою в аналізі даних. Зі збільшенням доступності даних часових рядів та текстових документів були запропоновані сотні алгоритмів аналізу. Серед цих методів лише деякі розглядають глибокі нейронні мережі (DNN) для виконання цього завдання. Це дивно, оскільки в останні роки глибоке навчання знайшло дуже успішні застосування. DNN зробили революцію в області комп'ютерного зору, особливо з появою нових глибших архітектур, таких як залишкові та згорткові нейронні мережі. Окрім зображень, послідовні дані, такі як текст та аудіо, також можна обробляти за допомогою DNN для досягнення найсучаснішої продуктивності для класифікації документів та розпізнавання мовлення. У цій роботі розглядається поточна продуктивність алгоритмів глибокого навчання для класифікації часових рядів та текстів, представляючи емпіричне дослідження низки архітектур DNN.

Техніки глибокого навчання для аналізу даних з послідовною семантичною структурою набирають популярності у дослідницьких напрямах. Вони забезпечують автоматичне вилучення ознак, а також розширені можливості представлення даних та кращу продуктивність, ніж традиційні методи, засновані на ознаках (тобто, поверхневі методи). Традиційні підходи засновані на складних отриманих вручну ознаках, і цей процес пошуку ознак є фундаментальною проблемою у вирішенні оптимізаційних задач. Ці давно сформульовані підходи все ж дають надійні якірні точки для метрик оцінки, а їх передбачувані можливості можна використовувати в поєднанні з методами глибокого навчання, для забезпечення можливості пояснення передбачень останніх.

Таким чином, основною метою даної роботи є проведення дослідження деяких алгоритмів глибокого навчання та їх застосування до задач класифікації даних з послідовною структурою. Розглядаються

сучасні підходи для покращення характеристик алгоритмів даного класу у оптимізаційному процесі та при передбаченні на нерозмічених даних.

1 ЗАСТОСУВАННЯ МЕТОДІВ ГЛИБОКОГО НАВЧАННЯ У ЗАДАЧАХ КЛАСИФІКАЦІЇ ДОКУМЕНТІВ ТА ЧАСОВИХ ПОСЛІДОВНОСТЕЙ

1.1 Основні поняття теорії глибокого машинного навчання

1.1.1 Застосування машинного навчання та нейронних мереж у сучасному світі

Методи машинного навчання активно досліджуються з середини XX століття. Історично, дослідження в цій області мало мінливий характер, і за успіхами в створенні алгоритмів [1] йшли “зими штучного інтелекту” [36], [1].

Початок першого десятиліття XXI ст. виявився поворотним моментом в історії машинного навчання, і це пояснюється трьома синхронними тенденціями, які разом дали помітний синергетичний ефект [1].

Перша тенденція це великі дані. Обсяг даних став настільки великим, що нові підходи були викликані практичною необхідністю а не через цікавість вчених.

Другою тенденцією є зниження витрат на паралельні обчислення та пам’ять. Ця тенденція була виявлена 2004 р Google представила свою технологію MapReduce, а потім його аналог з відкритим кодом Hadoop (2006) і разом вони дали можливість поширювати переробку величезних обсяги даних між простими процесорами. В той же самий Разом, Nvidia зробила прорив на ринку графічних процесорів: якщо раніше в ігровому сегменті він міг конкурувати з AMD/ATI, то в сегменті графічних процесорів, які можуть бути використовуваний для цілей машинного навчання, виявилося, що це а монополіст. І в той же час вартість оперативної пам'яті має значно зменшився, що відкривало можливість для робота з великими обсягами даних в пам'яті і, як а В результаті з'являється безліч нових типів баз даних,

включаючи NoSQL. Нарешті, у 2014 році фреймворк Apache Spark для розподіленої обробки неструктурованих і слабко структурованих даних, що зробило використання алгоритмів машинного навчання можливим в умовах великого масштабу датасетів.

Третя тенденція – розробка нових алгоритмів глибоке машинне навчання, успадкування та розвиток ідеї персептрону в поєднанні з успішним наук PR-кампанія. Після багатьох років вивчення багатошаровості нейронні мережі концепція технологія глибокої нейронної мережі (DNN).

На сьогодні, ці тренди обумовлюють широкий спектр використання алгоритмів машинного навчання та, особливо, нейронних мереж. До основних сфер застосування ML відносять [2], [3] наступні:

- інтелектуальне прийняття рішень;
- кібербезпека;
- інтернет речей та розумне місто;
- медицина;
- інтелектуальна логістика;
- тощо.

Проте переважну більшість застосунків домінують саме методи, які за основу використовують навчання глибоких нейронних мереж. У наступній секції розглянуто передумови ефективності глибокого навчання порівняно з класичними методами.

1.1.2 Обґрунтування переходу до глибокого навчання

У сфері машинного навчання DL, завдяки своєму успіху в індустрії, в даний час є одним з найважливіших напрямків досліджень. Постійна поява нових досліджень у галузях глибокого та розподіленого навчання пояснюється трендами у зборі даних та апаратних засобах для навчання, описаних у 1.1.1.

Глибоке навчання є похідною від одношарової нейронної мережі, але значно перевершує цю архітектуру за рахунок можливості навчання з

використанням нелінійних функцій. Водночас, DL використовує технології перетворення та графів, щоб побудувати багаторівневі моделі навчання. Останні розроблені методи DL мають найкращі результати з-поміж інших методів, перевершуючи результати, отримані людьми на деяких задачах, включаючи обробку аудіо та мовлення, обробку візуальних даних, обробку природної мови (NLP), та інших [4], [5].

Зазвичай ефективність класичного алгоритму ML сильно залежить від цілісності представлення вхідних даних. Було показано, що відповідне представлення даних забезпечує кращі результати у порівнянні з поганим представленням даних. Таким чином, значним дослідницьким напрямком у ML протягом багатьох років була інженерія ознак, яка послужила основою для численних досліджень.

Цей підхід спрямований на побудову ознак із необроблених даних. Крім того, він надзвичайно специфічний для конкретної галузі і часто вимагає значних людських зусиль. Наприклад, у контексті комп'ютерного зору було введено та порівняно кілька типів ознак, таких як гістограма орієнтованих градієнтів (HOG) [15], масштабно-інваріантне перетворення ознак (SIFT) [16] та пакет слів (BoW). [17].

В цей же час, вивчення найбільш репрезентативних ознак досягається автоматично в усіх алгоритмах DL. Це дозволяє отримувати високі результати, використовуючи найменшу можливу кількість людських зусиль і доменних знань [9]. Ці алгоритми мають багаторівневу архітектуру представлення даних, в якій перші шари вивчають базові, низькорівневі функції, а останні шари абстрагують їх

до рівня понять та об'єктів реального світу. При цьому, цей тип архітектури алгоритмів прагне імітувати процеси, що відбувається в основних сенсорних областях людського мозку. Так, аналізуючи машинний зір, можна провести пряму паралель між мозком людини та алгоритмом. Людський мозок приймає на вхід доступну інформацію та класифікує об'єкти, пропускаючи "сирі" дані через декілька шарів абстракції, що прямо відображається на принцип роботи глибокої нейронної мережі.

1.1.3 Огляд ключових елементів теорії глибокого навчання

1.1.3.1 Глибока нейронна мережа

Глибока нейронна мережа (англ. artificial neural network, ANN) ґрунтується на принципі, за яким працює біологічна нервова система. В одній з піонерських робіт глибокого навчання, Уоррен МакКаллок і Уолтер Піттс запропонували першу математичну модель біологічного нейрона [6]. Формально ANN можна визначити як елемент обробки, який отримує набір вхідних елементів, позначених як $X = x_1,x_2,...x_n$, які відповідно модифікуються для серії ваг $W = w_1, w_2, ..., w_n$. Різні значення, які змінюються ваговими коефіцієнтами, додаються разом у так званий чистий вхід (Чистий вхід є результатом додавання добутків кожного вхідного значення на його відповідну вагу, додаючи значення зміщення або поріг нейрона, позначається b, що визначається при його активації). Активація нейрона залежить від функції активації, яка діє на мережевий вхід, який також регулює вихід нейрона. Як видно на малюнку 1, що і сума, і функція активації представляють тіло клітини нейрона, в цьому місці здійснюються відповідні математичні обчислення і результати передаються на вихід мережі. Загалом, ANN ділиться на три шари [7]: вхідний, прихований і вихідний (див. рисунок 1), де для цього прикладу прихований шар має три нейрони.

Глибокі нейронні мережі з не менш як одним прихованим шаром є універсальними апроксиматорами функцій [8] і можуть моделювати будь-яке як завгодно складне відношення від входів моделі до передбачень.

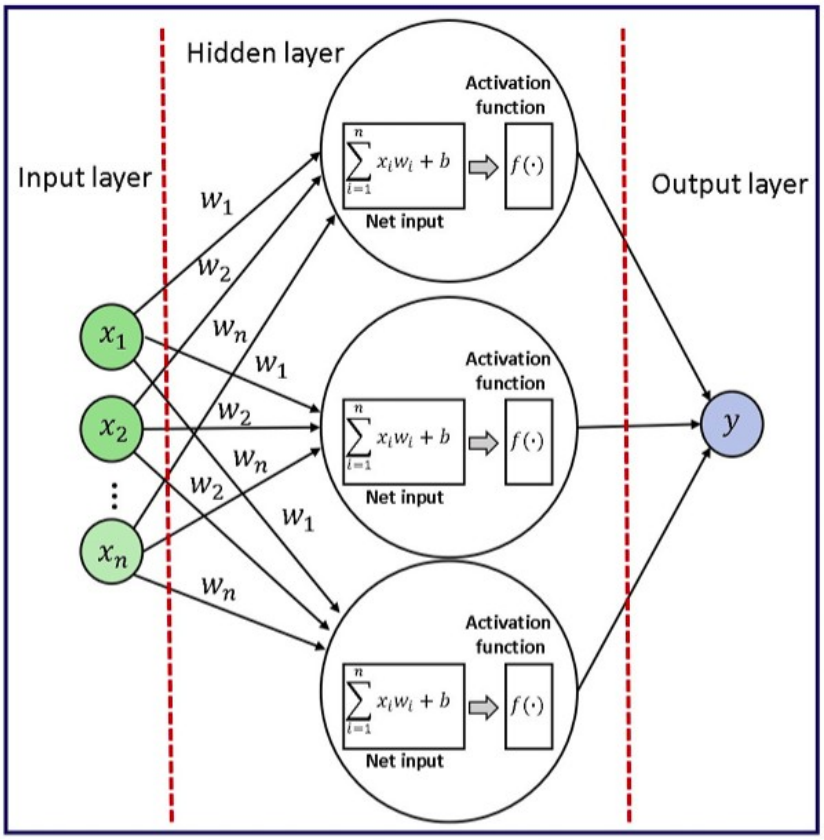

Рисунок 1 - Схема нейронної мережі з одним прихованим шаром

1.1.3.2 Нелінійна активація нейронів

Ключовим засобом для глибокої НМ є нелінійне моделювання, яке досягла значного прогресу в останні роки [9]. Так, керовані даними, моделі глибокого навчання можуть вивчати будь-яку форму нелінійності безпосередньо з даних без будь-яких явних припущень або попередніх знань про нелінійність. Ця властивість робить моделі особливо ефективними у вивченні властивостей кодування сенсорних стимулів у нервовій системі ([10]), чия анатомічна та функціональна організація залишається спекулятивною,

особливо для природні подразники.

1.1.3.3 Зворотне поширення похибки

Оптимізаційний процес в класичних нейронних мережах полягає в ітеративній мінімізації функції втрат за рахунок змін у зв'язках між нейронами мережі — вагами. Процедура багаторазового зворотного поширення похибки [11] змінює ваги так, щоб мінімізувати міру різниці між фактичним вихідним вектором мережі та бажаним вихідним вектором. У результаті коригування ваги внутрішні «приховані» одиниці, які не є частиною входу або виходу, стають представляти важливі характеристики області завдання, а закономірності в завданні фіксуються взаємодією цих одиниць. Здатність створювати корисні нові функції відрізняє зворотне поширення від попередніх, простіших методів.

1.1.3.4 Оптимізаційний процес. Алгоритм стохастичного градієнтного спуску

Розглядають дві цілі досліджень алгоритмів оптимізації [37]: швидший показник збіжності оптимізатора та кращі показники цільових метрик. Водночас, показники метрик, можуть сильно відрізнятися від втрат в процесі оптимізації та часто вимірюються на даних, які відкладаються на початку тренування моделі. Швидший метод не обов’язково є кращим у загальному випадку і не обов’язково покращує необхідну

метрику.

Розглянутий у попередньому пункті алгоритм зворотного поширення похибки на практиці можна реалізувати різними способами. Одним з базових методів є стохастичний градієнтний спуск (англ. SGD). На ітерації t, алгоритм полягає у зміні ваг моделі за формулою:

$$\theta_{t+1} = \theta_t - \alpha \nabla F_i(\theta_t), \quad (1)$$

де θ- ваги моделі, α- коефіцієнт навчання, $\nabla F_i(\theta_t)$- градієнт функції втрат.

На практиці, весь набір прикладів перемішується випадковим чином на початку кожної епохи, а потом ділиться на міні-блоки. На кожній ітерації, один блок завантажується у пам'ять для обчислень (обчислення градієнту та виконання модифікації ваг).

Причинами використання SGD замість GD є обмеження пам'яті та швидша збіжність. Один GPU або CPU не може завантажити всі приклади в свою пам’ять для обчислення повного градієнта, тому завантаження міні-блоку прикладів на кожній ітерації є альтернативою. Тим не менш, навіть з обмеженням пам’яті, оригінальний GD можна реалізувати шляхом накопичення всіх міні-блокових градієнтів без оновлення параметрів на кожній ітерації. У порівнянні з GD, реалізованим таким чином, перевагою SGD є більш висока швидкість збіжності [37]. Втім SGD не обов’язково швидший за GD, якщо всі зразки можна обробляти на одній машині паралельно, але в середовищі з обмеженою пам’яттю SGD часто набагато швидше, ніж GD. Кількість вибірок в одному міні-блоці залежить від доступного розміру пам’яті, а також залежить від кількості параметрів у моделі та інших алгоритмічних вимог (наприклад, проміжний вихід на кожному рівні). Наприклад, графічний процесор з об’ємом пам’яті 11 гігабайт може обробляти лише 512 зразків одночасно, якщо

використовується AlexNet для ImageNet, і може обробляти лише 64 зразки за один раз при використанні ResNet50 для ImageNet 10 [37].

1.2 Аналіз сучасних архітектур нейронних мереж

1.2.1 Загальна таксономія алгоритмів ГН

Алгоритми глибокого навчання можна поділити на класи за типом даних, для яких вони призначені (рис. 2). Слідуючи [12], виокремлюють три категорії алгоритмів ГН:

a) зі вчителем — алгоритми даного класу навчаються на розмічених наборах даних, тобто наявні дані розмічено згідно вирішуваної задачі;

b) без вчителя — використовувані дані не мають міток, а отже, у більшості випадків, алгоритми не мають зворотного зв'язку про якість передбачень;

c) гібридні алгоритми — даний клас представлений поєднанням представників a) та b) для вирішення задач як зі вчителем, так і без, включаючи навчання з підкріпленням.

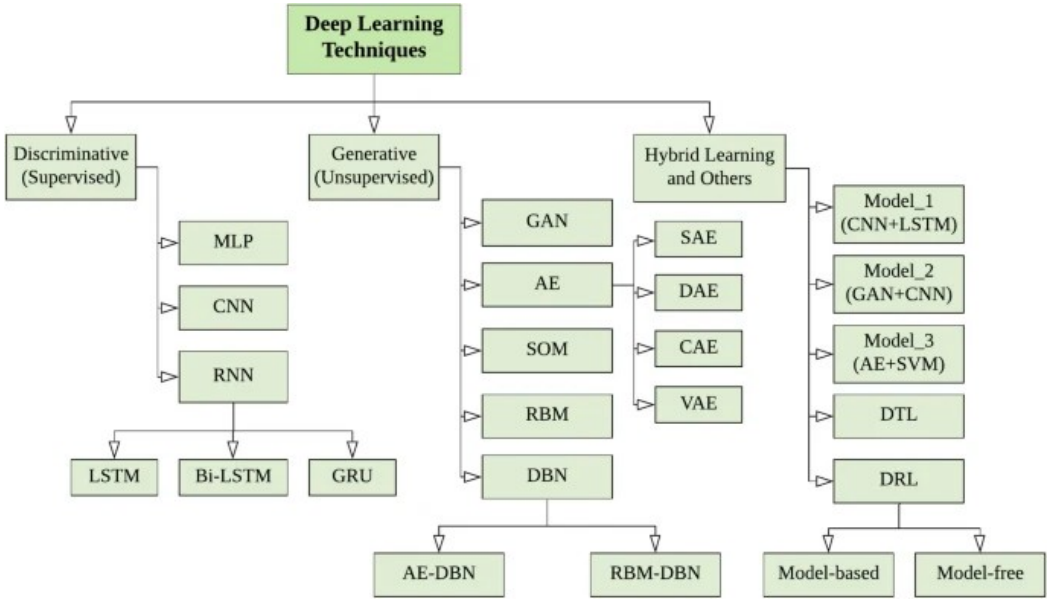

Рисунок 2 - Таксономія методів ГН, поділених на три основні категорії (i) глибокі мережі для навчання на розмічених даних, (ii) глибокі мережі для неконтрольованого або генеративного навчання та (iii) глибокі мережі для гібридного навчання

1.2.2 Класичні методи побудови нейронних мереж для задач класифікації

1.2.2.1 Багатошаровий перцептрон

Багатошаровий персептрон (англ. Multi-layer Perceptron, MLP), підхід до навчання зі вчителем [13], є типом штучної нейронної мережі з прямим зв’язком (ANN). Він також відомий як базова архітектура глибоких нейронних мереж (DNN) або глибоке навчання. Типова MLP - це повнозв’язна мережа, яка складається з вхідного рівня, який отримує вхідні дані, вихідного рівня, який приймає рішення або передбачення щодо вхідного сигналу, і одного або кількох прихованих шарів між цими двома, які вважаються обчислювальним механізмом мережі. Вихід мережі MLP визначається за допомогою різноманітних функцій активації, також відомих як функції передачі, таких як ReLU (Rectified Linear Unit), Tanh, Sigmoid і Softmax [13]. Типову архітектуру MLP наведено на рис. 3.

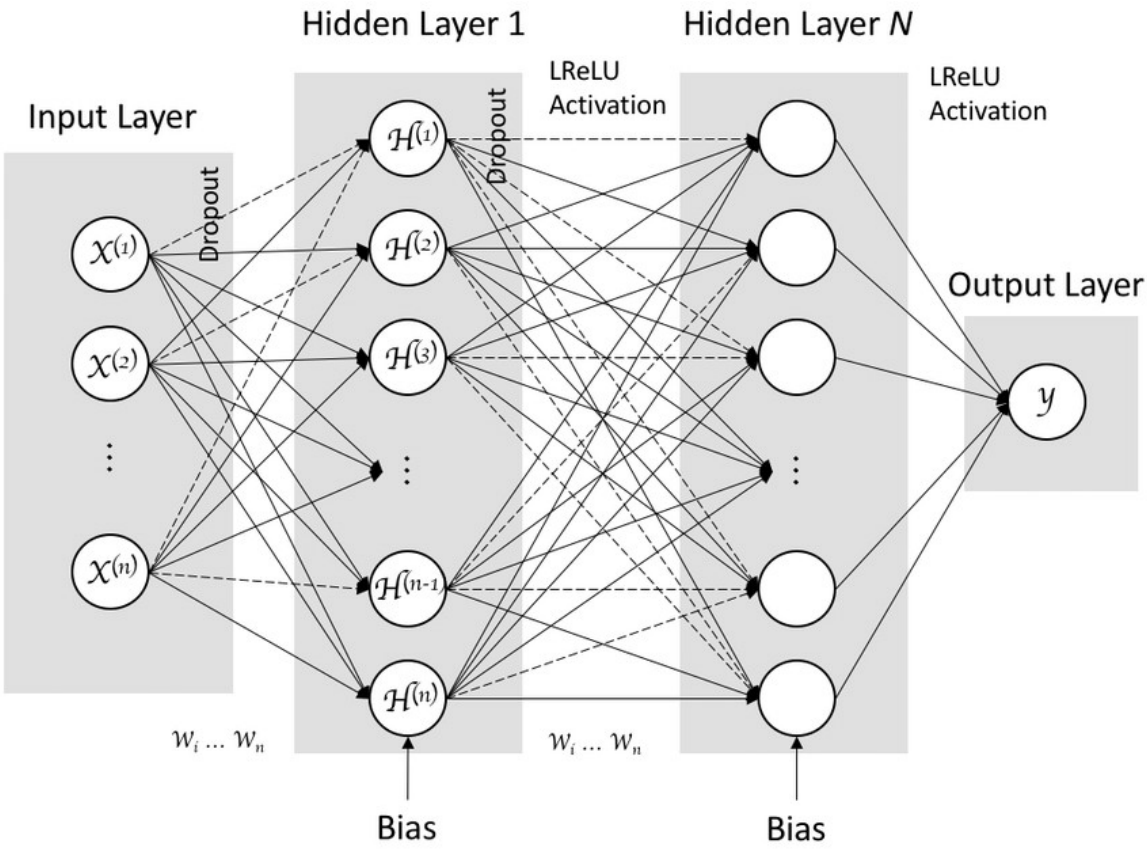

Рисунок 3 - Класична повнозв'язна мережа з двома прихованими шарами

Для навчання MLP найбільш широко використовується алгоритм зворотного поширення похибки, який розглянуто у 1.1.3. Під час навчального процесу застосовуються різні підходи оптимізації, такі як стохастичний градієнтний спуск (SGD), обмежена пам’ять BFGS (L-BFGS) та адаптивна оцінка моменту (Adam). MLP вимагає налаштування кількох гіперпараметрів, таких як кількість прихованих шарів, нейронів та ітерацій, що може зробити розв’язання складної моделі дорогим з точки зору кількості обчислень. Однак завдяки частковому підгонці MLP пропонує перевагу вивчення нелінійних моделей в режимі реального часу або онлайн [13].

1.2.2.2 Згорткова нейронна мережа

Згорткова нейронна мережа (CNN або ConvNet) [14] є популярною дискримінаційною архітектурою глибокого навчання, яка навчається безпосередньо з вхідних даних без необхідності вилучення людських ознак. На малюнку 4 показано приклад CNN, що включає кілька згорток і шарів об'єднання. В результаті CNN покращує дизайн традиційних ANN, таких як упорядковані мережі MLP. Кожен рівень в CNN виокремлює оптимальні ознаки репрезентації даних (вхідного зображення, послідовності, виходу попереднього шару згорткової мережі, тощо), яку було передано на даному кроці.

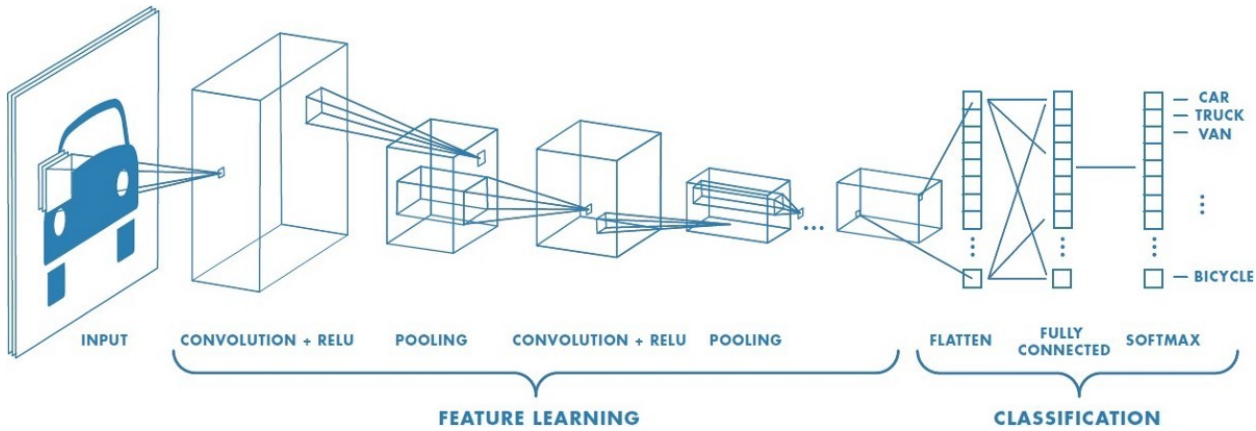

Рисунок 4 - Класична архітектура згорткової мережі

CNN спеціально призначені для роботи з різноманітними двовимірними формами і, таким чином, широко використовуються для візуального розпізнавання, аналізу медичних зображень, сегментації зображень, обробки природної мови та багатьох інших [14]. Можливість автоматичного виявлення основних функцій із вхідних даних без необхідності втручання людини робить її потужнішою, ніж традиційна мережа. Існує кілька варіантів CNN [15], до яких належать: VGG, AlexNet, Xception, Inception, ResNet тощо, які можна використовувати в різних застосунках відповідно до їхніх навчальних можливостей.

1.2.2.3 RNN

Рекурентна нейронна мережа (RNN) — це ще одна популярна нейронна мережа, яка використовує послідовні дані або дані часового ряду та подає вихідні дані попереднього кроку як вхідні дані на поточний етап [16]. Як і прямий зв'язок і CNN, рекурентні мережі навчаються на вхідних даних навчання, однак відрізняються наявністю механізму «пам'яті», що дозволяє їм обробляти поточний вхід і вихід, використовуючи інформацію з попередніх вхідних даних. На відміну від типової DNN, яка передбачає, що входи та виходи незалежні один від одного, вихід RNN залежить від попередніх елементів у послідовності (рис. 5). Однак стандартні рекурентні мережі мають проблему зникаючих градієнтів, що робить вивчення довгих послідовностей даних складним.

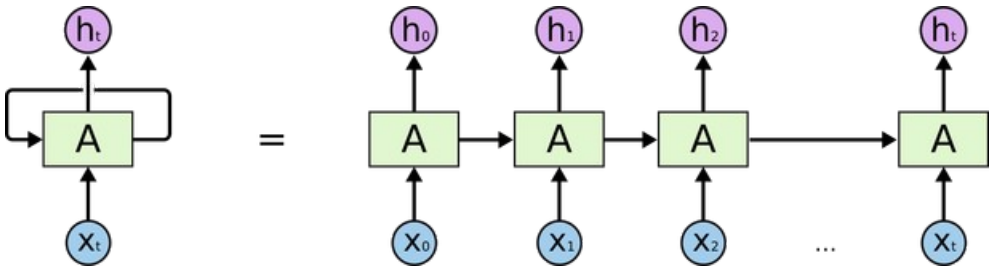

Рисунок 5 - Одношарова рекурентна мережа

Далі описано кілька популярних варіантів рекурентної мережі, яка мінімізує проблеми та добре працює в багатьох реальних доменах додатків.

а) Довготривала пам’ять (LSTM) Це популярна форма архітектури

RNN, яка використовує спеціальні блоки для вирішення проблеми зникаючого градієнта, яку представили Хохрайтер та ін. [17]. Комірка пам'яті в блоці LSTM може зберігати дані протягом тривалого періоду, а потік інформації в комірку та з неї керується трьома вентилями. Наприклад, 'Forget Gate' визначає, яка інформація з комірки попереднього стану буде запам'ятатися і яка інформація буде видалена, яка більше не є корисною, тоді як 'Input Gate' визначає, яка інформація повинна увійти в стан клітинки, а 'Output Gate' ' визначає і керує виходами. Оскільки вона вирішує питання навчання рекурентної мережі, мережа LSTM вважається однією з найуспішніших RNN.

b) Двонапрямлені RNN/LSTM. Двонапрямлені RNN з’єднують два прихованих шари, які працюють у протилежних напрямках, до одного виходу, що дозволяє їм приймати дані як з минулого, так і з майбутнього. Двонапрямлені RNN, на відміну від традиційних рекурентних мереж, навчені передбачати як позитивні, так і негативні напрямки часу одночасно. Двонапрямлений LSTM, часто відомий як BiLSTM, є розширенням стандартного LSTM, що може підвищити продуктивність моделі щодо питань класифікації послідовностей [18]. Це модель обробки послідовності, що складається з двох LSTM: один приймає вхід вперед, а інший — назад. Двонапрямлений LSTM, зокрема, є популярним вибором у задачах обробки природної мови.

c) Рекурентні блоки зі стробуванням (GRU). Затворний повторюваний блок (GRU) є ще одним популярним варіантом рекурентної мережі, який використовує методи стробування для контролю та керування потоком інформації між клітинами нейронної мережі, представлений Чо та ін. [19]. GRU схожий на LSTM, однак має менше параметрів, оскільки має шлюз скидання та шлюз оновлення, але не має вихідного шлюза, як показано на рис. 8. Таким чином, ключова відмінність між

GRU та LSTM полягає в тому, що GRU має два вентилі (шлюзи скидання та оновлення), тоді як LSTM має три вентилі (а саме вхідні, вихідні та забуття). Структура GRU дозволяє йому адаптивно фіксувати залежності від великих послідовностей даних, не відкидаючи інформацію з попередніх частин послідовності. Таким чином, GRU є дещо більш обтічним варіантом, який часто пропонує порівнянну продуктивність і значно швидше обчислюється [20]. Хоча було показано, що GRU демонструють кращу продуктивність на деяких менших і менш частих наборах даних [20], обидва варіанти RNN довели свою ефективність при отриманні результату.

1.2.2.3.1 Особливості роботи RNN у задачах з послідовною структурою даних

Прямий прохід через мережу (відображення послідовності входів у послідовність виходів) у звичайній RNN є практично ідентичним розглянутому для мереж прямого зв'язку. Щоб обчислити вихід yt для входу xt, потрібно значення активації для прихованого шару ht. Щоб обчислити це, множиться вхідний xt на вагову матрицю W, а прихований шар з попереднього кроку часу ht−1 на вагову матрицю U. Ці значення додаються і передаються через відповідну функцію активації g, щоб отримати значення активації для поточного прихованого шару, ht. Отримавши значення для прихованого шару, обчислюється вихідний вектор за використання функції активації [41].

$$h_t = g\left(Uh_{t-1} + Wx_t\right) \quad (2)$$

$$y_t = f\left(Vh_t\right) \quad (3)$$

Позначимо вхідний, прихований і вихідний шари як d_{in}, d_h і d_{out} відповідно. З огляду на це, маємо три матриці параметрів: $W \in R_{dh \times din}$, $U \in R_{dh \times dh}$ та $V \in R_{dout \times dh}$.

У звичайному випадку м'якої класифікації обчислення y_t складається з

обчислення функції softmax, яка забезпечує розподіл ймовірності за можливими вихідними класами.

$$y_t = softmax\left(Vh_t\right) \tag{4}$$

Той факт, що обчислення в момент t вимагає значення прихованого шару з моменту t−1, вимагає використання алгоритму інкрементного висновку, який продовжується від початку послідовності до кінця. Послідовну природу простих рекурентних мереж можна також побачити, розгорнувши мережу в часі, як показано на рис. 6. На цьому рисунку різні шари ваг копіюються для кожного кроку часу, щоб проілюструвати, що вони будуть мати різні значення з часом. Однак різні вагові матриці використовуються спільно в часі.

Як і в мережах з прямим зв’язком, використовується навчальний набір, функцію втрат і зворотне поширення, щоб отримати градієнти, необхідні для коригування ваг у цих рекурентних мережах. Як вже сказано, тепер є 3 набори ваг для оновлення: W, ваги від вхідного шару до прихованого шару, U, ваги від попереднього прихованого шару до поточного прихованого шару, і, нарешті, V, ваги від прихованого шару до вихідного шару.

На рис. 6 висвітлено два міркування, що відрізняють рекурентні мережі від мереж з прямим зв’язком. По-перше, щоб обчислити функцію втрат для виходу в момент t, потрібен прихований шар з моменту t −1. По-друге, прихований шар у момент t впливає як на вихід у момент t, так і на прихований шар у момент часу t +1 (і, отже, на

вихід і втрати в момент t +1). Звідси випливає, що для оцінки помилки, яка накопичується до h_t,потрібно знати її вплив як на поточний вихід, так і на наступні.

Алгоритм зворотного поширення у цьому випадку переходить до двопрохідного алгоритму навчання ваг у RNN. На першому проході виконується прямий висновок, обчислюючи h_t, y_t, накопичуючи втрати на кожному кроці часу, зберігаючи значення прихованого шару на кожному кроці для використання на наступному кроці часу. На другому етапі обробляється послідовність у зворотному порядку, обчислюючи необхідні градієнти на ходу, обчислюючи та зберігаючи значення помилки для використання в прихованому шарі для кожного кроку назад у часі (рис. 6). Цей загальний підхід зазвичай називають зворотним поширенням через час [38], [39], [40].

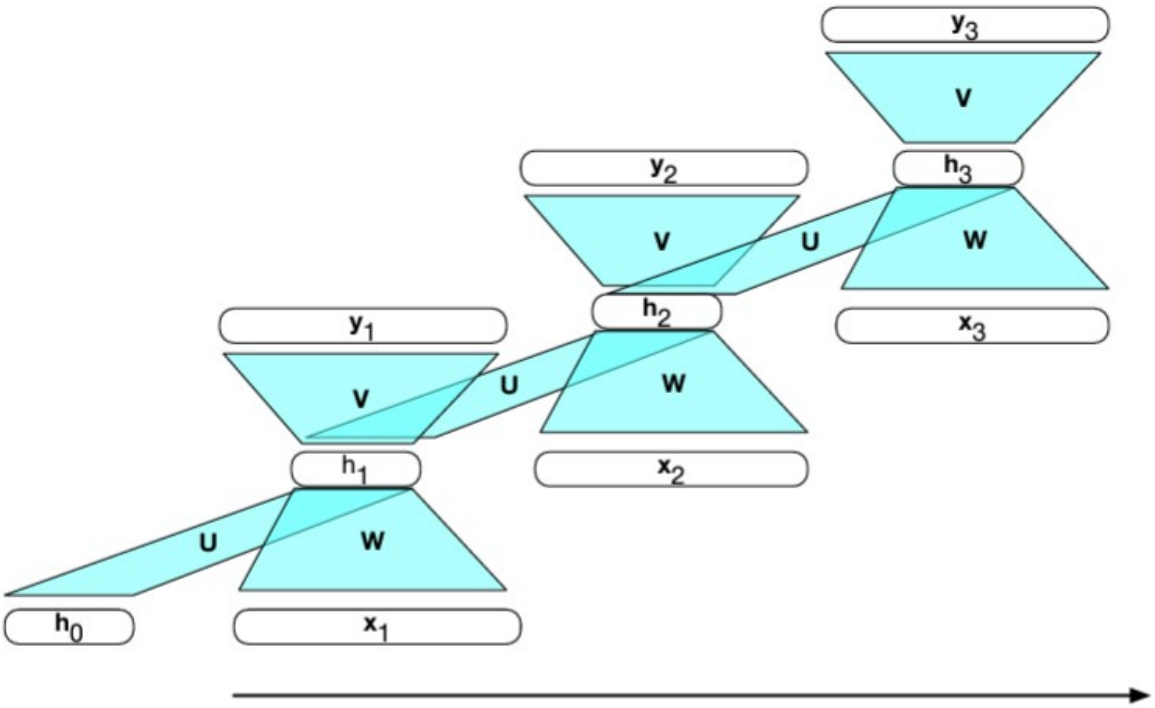

Рисунок 6 - Розгортання RNN у часі

На щастя, за наявності сучасних обчислювальних систем і відповідних обчислювальних ресурсів немає необхідності в спеціалізованому підході до навчання RNN. Як показано на рис. 6, явне розгортання рекурентної мережі в обчислювальний граф з прямим зв’язком усуває будь-які явні повторення, дозволяючи напряму навчати ваги мережі.

Для програм, які передбачають набагато довшу вхідну послідовність, наприклад, розпізнавання мовлення, обробку на рівні символів або потокове безперервне введення, розгортання всієї вхідної послідовності може бути неможливим. У цих випадках можна розгорнути вхідні дані в керовані сегменти фіксованої довжини та розглядати кожен сегмент як окремий навчальний елемент.

1.2.2.3.2 Сучасна варіація рекурентної архітектури

На практиці досить важко навчити звичайну RNN для виконання завдань, які вимагають, щоб мережа використовувала інформацію, віддалену від поточної точки у послідовності. Незважаючи на доступ до всієї попередньої послідовності, інформація, закодована в прихованих станах, як правило, є досить локальною, більш актуальною для останніх частин вхідної послідовності та останніх передбачень [41]. І все ж віддалена інформація має вирішальне значення для багатьох мовних програм. Розглянемо наступний приклад у контексті мовного моделювання.

Призначити високу ймовірність слідування авіакомпанії просто, оскільки авіакомпанія забезпечує сильний локальний контекст для окремої угоди. Однак призначити відповідну ймовірність до були досить складно не тільки тому, що множина польотів досить віддалена, а й тому, що проміжний контекст включає одиничні складові. В ідеалі мережа повинна мати можливість зберігати віддалену інформацію про множинні рейси, поки вона не знадобиться, при цьому правильно обробляючи проміжні частини послідовності.

Однією з причин нездатності RNN передавати важливу інформацію є те,

що приховані шари і, відповідно, вагові коефіцієнти, які визначають значення в прихованому шарі, просять виконувати два завдання одночасно: надавати інформацію, корисну для поточного рішення, а також оновлення та перенесення інформації, необхідної для майбутніх рішень.

Друга складність з навчанням RNN виникає через необхідність зворотного поширення сигналу помилки назад у часі. Під час зворотного проходження навчання приховані шари підлягають повторним множенням, що визначаються довжиною послідовності. Частим результатом цього процесу є те, що градієнти в кінцевому підсумку зводяться до нуля, а ситуація називається проблемою зникаючих градієнтів.

Щоб вирішити ці проблеми, були розроблені складніші архітектури мережі, щоб чітко керувати завданням підтримки відповідного контексту з часом, дозволяючи мережі навчитися забувати інформацію, яка більше не потрібна, і запам’ятовувати інформацію, необхідну для прийняття рішень.

Найбільш часто використовуваним таким розширенням до RNN є мережа довготривалої пам’яті (LSTM) [17]. LSTM поділяють проблему управління контекстом на дві підпроблеми: видалення інформації, яка більше не потрібна з контексту, і додавання інформації, яка може знадобитися для подальшого прийняття рішень. Ключем до вирішення обох проблем є навчитися керувати цим контекстом, а не жорстко кодувати стратегію в архітектурі. LSTM досягають цього, спочатку додаючи явний контекстний рівень до архітектури (на додаток до

звичайного повторюваного прихованого шару) і за допомогою використання спеціалізованих нейронних блоків, які використовують вентилі для керування потоком інформації в блоки, які виходять із них. складаються з мережевих рівнів. Ці вентилі реалізуються за допомогою використання додаткових ваг, які діють послідовно на вхідному, попередньому прихованому шарі та попередніх рівнях контексту. Шлюзи в LSTM мають шаблонний дизайн; кожен складається з шару прямого зв'язку, за яким слідує функція активації сигмоїда, за якою слідує скалярне множення з шаром, на який діє шлюз.

Вибір сигмоїди як функції активації виникає через її тенденцію підштовхувати свої вихідні дані до 0 або 1. Поєднання цього з точковим множенням має ефект, подібний до ефекту двійкової маски. Значення в шарі, на який діє шлюз, які збігаються зі значеннями біля 1 в масці, передаються майже без змін; значення, що відповідають нижчим значенням, по суті стираються. Перші ворота, які ми розглянемо, це ворота забуття. Мета цього шлюзу – видалити інформацію з контексту, яка більше не потрібна. Шлюз забуття обчислює зважену суму прихованого шару попереднього стану та поточного вхідного сигналу та передає це через сигмоїду. Ця маска поелементно множиться на вектор контексту, щоб видалити інформацію з контексту, яка більше не потрібна.

У вигляді обчислювального графу, розглянуті операції зображено на рис. 7.

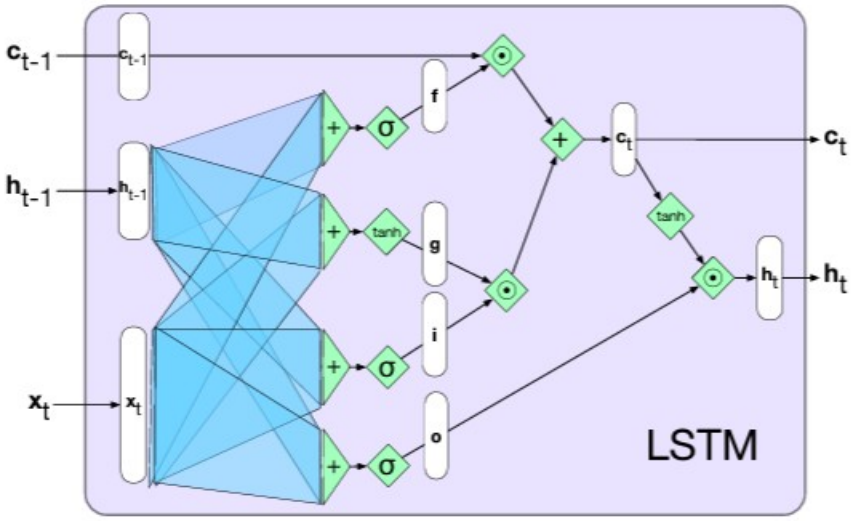

Рисунок 7 - Обчислювальний граф LSTM-клітини

1.2.3 Універсальні НМ

Рекурентні нейронні мережі послідовно обробляють дані, оновлюючи свій стан з кожною новою точкою даних, і вже давно є де-факто вибором для завдань моделювання послідовності. Однак їхнє послідовне обчислення робить їх повільними в навчанні. Нещодавно було продемонстровано, що архітектури з само-увагою (англ. self-attention) досягають чудових результатів у деяких задачах моделювання послідовності, таких як машинний переклад, з додатковою перевагою, що вони одночасно обробляють усі вхідні дані в послідовності, що призводить до легкого розпаралелювання та прискорення навчання. Однак, незважаючи на ці успіхи, популярні моделі само-уваги, такі як Transformer, не можуть узагальнити багато простих завдань, які рекурентні моделі легко виконують, наприклад. копіювання рядків або навіть простий логічний висновок, коли довжина рядка або формули перевищує спостережувану під час навчання.

Втім, архітектури типу Transformer, нещодавно змогли перевершити RNN для цілого ряду завдань моделювання послідовностей [21].

Модель Transformer, зокрема, повністю спирається на механізм власної

уваги [22] для обчислення серії контекстно-інформованих уявлень символів у векторному просторі на її вході та виході, які потім використовуються для прогнозування розподілу над наступними символами, оскільки модель передбачає вихідну послідовність символ за символом. Цей механізм не тільки легко розпаралелювати, але оскільки представлення кожного символу також безпосередньо залежить від уявлень усіх інших символів, це призводить до ефективного глобального сприйнятливого поля в усій послідовності. Це на відміну від напр. згорткові архітектури, які зазвичай мають лише обмежене сприйнятливе поле. Примітно, однак, що Transformer з його фіксованим стеком різних шарів уникає індуктивного зміщення RNN до навчання ітеративним або рекурсивним перетворенням.

Нещодавно модель Transformer увійшла у візуальний домен застосунків машинного навчання [23]. Прориви мереж “Transformer” у сфері обробки природної мови (NLP) викликали великий інтерес у спільноті комп’ютерного зору, щоб адаптувати ці моделі для завдань зору та мультимодального навчання.

В той же час, моделі гібридного класу, описані у секції 1.3.2.1, як і звичайні моделі для навчання з вчителем, можуть у окремих випадках бути використані не у своєму цільовому домені. Так, у [34] було натреновано згорткову мережу для застосування у сфері обробки природної мови з результатами, порівняними з рекурентними мережами.

1.3 Огляд задач класифікації даних з послідовною структурою

Класифікація послідовностей — це завдання прогнозування міток класів для даних у вигляді послідовностей. Воно має центральне значення в багатьох сферах, таких як класифікація документів, геномний аналіз, клінічна історія пацієнта та ін. [35]. Наприклад, класифікація документів за різними тематичними категоріями є проблемою для бібліотекознавства, особливо для сучасних цифрових бібліотек. Геномна класифікація допомагає дослідникам краще зрозуміти деякі захворювання. Класифікація часових рядів ЕКГ визначає, чи є людина здоровою або має захворювання серця. Машинне навчання, особливо глибоке навчання, набуває широкого поширення в класифікації послідовностей, коли одна модель вивчає залежність між всіма початковими входами послідовності і кінцевою міткою.

Рекурентні нейронні мережі, трансформери та згорткові нейронні мережі — це три основні методи аналізу послідовних даних. RNN використовують свої внутрішні стани для обробки послідовності крок за кроком. Незважаючи на успіх коротких послідовностей, традиційні RNN не можуть масштабуватися до дуже довгих послідовностей. Одна з причин полягає в тому, що їм важко тренуватися через проблеми з вибухаючим або зникаючим градієнтом. Крім того, передбачення кожного кроку часу має чекати завершення передбачення всіх його попередників, що ускладнює розпаралелювання RNN. Трансформери – це сімейство моделей, що спираються на механізм власної уваги. Вони мають квадратичну складність часу та пам’яті до довжини вхідної послідовності, оскільки вони обчислюють парні скалярні добутки. Для зниження цієї вартості потрібні складні методи оптимізації мережі. На відміну від цього, CNN здатні обробляти дуже довгі послідовності. Згортковий шар використовує розріджені зв’язки і не має повторюваних вузлів. Тому CNN легше тренувати та розпаралелювати. Крім того, розширені згортки можуть експоненціально збільшувати сприйнятливі

поля, дозволяючи CNN використовувати менше шарів для захоплення довгострокових залежностей. Часові згорткові мережі (англ. Time convolutional networks, TCN) забезпечують високу результативність у задачах класифікації послідовностей.

1.3.1 Класифікація текстових даних

Завдання класифікації тексту полягає у віднесенні документа до однієї або кількох категорій, виходячи з семантичного змісту документа. Класифікація документів (або тексту) виконується у двох режимах:
Фаза навчання та фаза прогнозу (або класифікації).
В свою чергу етап навчання можна розділити на три види:

- контрольована класифікація документів виконується за допомогою зовнішнього механізму, як правило, зворотного зв'язку людини, який надає необхідну інформацію для правильної класифікації документів,
- напів-контрольована класифікація документів, суміш між контрольованою та неконтрольованою класифікацією: деякі документи або частини документів позначаються зовнішньою допомогою,
- неконтрольована класифікація документів повністю виконується без посилання на зовнішню інформацію.

В рамках даної роботи, задача класифікації тексту формулюється наступним чином: для заданого набору пар (текст, мітка) потрібно передбачити коректний клас (мітку) з наступними умовами:

- з-поміж N класів, коректний клас лише один
- кожна вхідна пара даних містить коректну мітку

1.3.2 Класифікація часових послідовностей

На відміну від описаного у попередній секції, випадок часових послідовностей передбачає впорядкованість даних у часовій розмірності.

Розглянуті етапи класифікаційного процесу діють і в цьому випадку. Варто зауважити, що задача класифікації не є аналогічною задачі передбачення часових рядів. У випадку останньої вирішується регресійна задача передбачення значення, яке набуває ряд у час t+1, використовуючи попередні значення 1..t. Класифікація часових рядів полягає у знаходженні відповідності між членами ряду за час 1..t та одною (або декількома) з фіксованих міток. Наприклад, типова задача в даному випадку є зіставлення сигналу, що отримується, до типу приладу, який його продукує (рис. 8).

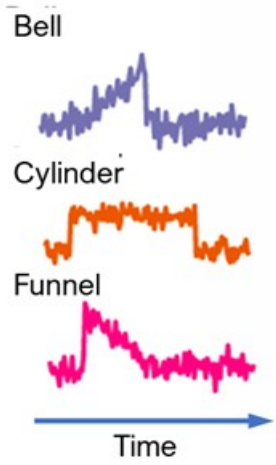

Рисунок 8 - Приклад задачі класифікації часових рядів

1.4 Вибір архітектур нейронних мереж для класифікації документів та часових послідовностей

Розглянуті у 1.2.2.2-3 та 1.2.3 архітектури представляють

найбільший інтерес для проведення дослідження. До обраних архітектур пропонується ввести наступні модифікації, враховуючи особливості задачі використання одної архітектури для класифікації тексту та часових послідовностей:

- для дослідження доцільно взяти LSTM архітектуру рекурентних мереж, що обумовлюється використанням мережі в умовах довгих часових послідовностей (до 380 елементів), хоча переважна частина текстових даних є звичайними реченнями англомовного тексту (до 50 слів);
- в згортковій архітектурі, яка є широковживаною для задач з часовими рядами, необхідно внести зміни по відношенню до текстових даних. Так, початковий шар мережі має вміти обробляти текст, який, в свою чергу, пропонується перевести у векторну репрезентацію, використавши ембедінг матрицю (англ. embedding matrix);
- трансформер можна використати аналогічно згортковій мережі. Для даної моделі необхідно передбачити можливість використання натренованої ембедінг матриці.

2 Порівняльний аналіз методів глибокого навчання на деяких наборах даних

2.1 Вибір наборів даних для експериментів

Для експериментів з класифікації документів та часових послідовностей було обрано по три набори даних для кожної задачі. Вибір здійснювався згідно об'єму та якості розмітки даних.

2.1.1 Текстові набори даних

На основі [25] та [26] було обрано наступні набори даних для експериментів з класифікацією документів:

a) IMDb Movie Reviews - набір даних для аналізу настроїв, що складається з 50 000 відгуків з бази даних Інтернет-фільмів (IMDb), розмічених як позитивні або негативні. Набір даних містить однакову кількість позитивних і негативних відгуків. Розглядаються лише відгуки, які однозначно відносяться до того чи іншого класу. Негативна рецензія має оцінку ≤ 4 з 10, а позитивна — ≥ 7 з 10. На фільм включено не більше 30 рецензій. Набір даних містить додаткові нерозмічені дані.

b) GoEmotions - корпус із 58 тисяч ретельно підібраних коментарів, отриманих з платформи Reddit, з людськими анотаціями, що відмічають кожен коментар як одну з 27 категорій емоцій або як нейтральний.

Кількість прикладів: 58 009.

Кількість міток: 27 + нейтральна.

Максимальна довжина послідовності в наборах навчальних та тестових даних: 30.

Категорії представлених емоцій: захоплення, веселість, гнів, роздратування, схвалення, турбота, розгубленість, цікавість, бажання, розчарування, несхвалення, огида, збентеження, хвилювання, страх,

вдячність, горе, радість, любов, нервозність, оптимізм, гордість, усвідомлення, полегшення, каяття, смуток, здивування.

c) Disaster Tweets — задача бінарної класифікації, де передбачено, чи стосується даний твіт справжню катастрофу чи ні.

2.1.2 Датасети часових рядів

На основі [27] та [28] було обрано наступні набори даних для експериментів з класифікацією часових рядів:

a) MotionSense - цей датасет включає дані часових рядів, згенеровані датчиками акселерометра та гіроскопа (відношення, гравітація, userAcceleration і rotationRate). Він збирається за допомогою iPhone 6s, який зберігається в передній кишені учасника за допомогою SensingKit, який збирає інформацію з фреймворку Core Motion на пристроях iOS. Усі дані збираються з частотою дискретизації 50 Гц. Загалом 24 учасники різної статі, віку, ваги та зросту виконували 6 видів діяльності в 15 випробуваннях в однакових умовах та в однакових умовах: внизу, нагорі, ходьба, біг, сидячи та стоячи.

b) GestureWiimote - Вихідні дані включають 10 досліджуваних, кожен виконує 10 жестів 10 разів. Пристроєм для отримання жестів є пульт дистанційного керування Nintendo Wiimote з вбудованим триосьовим акселерометром. Часові ряди мають різну довжину. Немає відсутніх значень.

c) Condition monitoring of hydraulic systems - Набір даних містить необроблені дані датчика (тобто, без вилучення ознак), які

структуровані у вигляді матриць з рядками, що представляють цикли роботи, а стовпці — точки даних у циклі. Датасет було отримано експериментально за допомогою гідравлічної випробувальної установки. Цей випробувальний прилад складається з первинного робочого та вторинного контурів охолодження-фільтрації, які з'єднані через масляний бак. Система циклічно повторює постійні цикли навантаження (тривалість 60 секунд) і вимірює такі значення процесу, як тиск, об'ємна витрата і температура, в той час як стан чотирьох гідравлічних компонентів (охолоджувач, клапан, насос і акумулятор) кількісно змінюється. Оскільки кожен компонент це окрема задача класифікації, для експериментів обрано найперший з наведених компонентів.

2.2 Цільові метрики та вибір порівнюваних характеристик процесу навчання обраних алгоритмів

Метою даної роботи є порівняльний аналіз низки алгоритмів глибокого навчання. Для досягнення цієї мети пропонується використати як результативність алгоритмів, так і характеристики оптимізаційного процесу. До обраних метрик результативності належать:

- Точність — навчений класифікатор вимірюється на основі повної коректності, яка відноситься до загальної кількості випадків, які правильно передбачені навченим класифікатором під час тестування з нерозміченими даними. Оцінено точність на тестових даних та по закінченню тренування.
- ROC-AUC — Receiver Operator Characteristic (ROC) є метрикою оцінки для задач бінарної класифікації. Це крива ймовірності, яка відкладає TPR від FPR при різних порогових значеннях і по суті відокремлює «сигнал» від «шуму». Площа під кривою (англ. area under the curve,

AUC) є мірою здатності класифікатора розрізняти класи і використовується як кількісна міра кривої ROC. Дана метрика використовується для задач бінарної класифікації.

- F1-score — є метрикою для задач бінарної класифікації, яка поєднує в собі точність і відгук моделі, і визначається як середнє гармонійне значення точності (англ. precision) та відгуку (англ. recall) моделі.

Також оцінено такі показники моделей як час тренування, час передбачення на тестовій вибірці та кількість змінюваних параметрів.

2.3 Результати порівняльного аналізу

Як для текстових даних, так і для часових послідовностей, взято по декілька моделей кожного типу: рекурентна, згорткова та трансформер. Для порівняння, у кожному експерименті наведено результативність базової моделі класичного машинного навчання.

Додатково визначено наступні умови експериментів:

- перехресна валідація (англ. cross-validation) з розділенням на 4 набори даних: 3 — тренування моделі, 1 — валідація результатів
- обмеження на кількість епох тренування: 7.
- оптимізатор: AdamW.
- прискорювач: Nvidia GeForce GTX 960M.

2.3.1 Результати класифікації текстових даних

Базова модель — логістична регресія. У табл. 1 наведено результати роботи моделей на обраних завданнях класифікації тексту.

Таблиця 1 — Результативність моделей на текстових датасетах

	Датасет			
Модель	Точність (train)	Точність (test)	F1-score (avg)	ROC-AUC (avg)
	Disaster Tweets			

Кінець таблиці 1

	Датасет			
Модель	Точність (train)	Точність (test)	F1-score (avg)	ROC-AUC (avg)
Базова модель	0.91	0.74	0.74	0.5
LSTM	0.86	0.79	0.79	0.67
CNN	0.92	0.78	0.78	0.6
Transformer	0.82	0.82	0.82	0.65
	IMDb Movie Reviews			
Базова модель	0.98	0.86	0.86	0.5
LSTM	0.77	0.67	0.67	0.55
CNN	0.91	0.68	0.67	0.41
Transformer	0.82	0.82	0.82	0.42
	GoEmotions			
Базова модель	0.39	0.32	0.09	-
LSTM	0.33	0.32	0.07	-
CNN	0.44	0.3	0.13	-
Transformer	0.35	0.34	0.12	-

Деякі характеристики оптимізаційного процесу наведені у табл. 2.

Таблиця 2 — характеристики моделей на текстових датасетах

	Датасет		
Модель	Час тренування, с	Час передбачення на тестових даних, с	Кількість тренувальних параметрів
	Disaster Tweets		
LSTM	18.53	0.99	893442
CNN	12.63	0.56	53058
Transformer	639.2	10.89	593668
	IMDb Movie Reviews		
LSTM	23.39	1.1	893442
CNN	15.54	0.62	53058
Transformer	835.36	13.74	593668
	GoEmotions		
LSTM	23.77	1.5	900124
CNN	15.48	1.26	59740
Transformer	838.44	14.53	633656

2.3.2 Результати класифікації часових послідовностей

Базова модель — випадковий ліс. У табл. 3 наведено результати роботи моделей на обраних завданнях класифікації часових рядів.

Таблиця 3 — Результативність моделей на датасетах з часовими рядами

	Датасет		
Модель	Точність (train)	Точність (test)	F1-score (avg)
	MotionSense		
Базова модель	0.73	0.69	0.54
LSTM	0.87	0.7	0.62
CNN	0.83	0.69	0.62
Transformer	0.79	0.71	0.62
	GestureWiimote		
Базова модель	0.9	0.7	0.45
LSTM	0.57	0.4	0.3

Кінець таблиці 3

	Датасет		
Модель	Точність (train)	Точність (test)	F1-score (avg)
CNN	0.96	0.6	0.52
Transformer	0.61	0.4	0.3
	Condition monitoring of hydraulic systems		
Базова модель	0.982	0.981	0.981
LSTM	0.997	0.997	0.997
CNN	0.998	0.998	0.998
Transformer	0.997	0.997	0.997

Деякі характеристики оптимізаційного процесу наведені у табл. 4.

Таблиця 4 — характеристики моделей на датасетах з часовими рядами

	Датасет		
Модель	Час тренування, с	Час передбачення на тестових даних, с	Кількість тренувальних параметрів
	MotionSense		
LSTM	32.65	1.27	205574
CNN	33.46	0.98	34310
Transformer	39.17	0.98	104158
	GestureWiimote		
LSTM	2.84	0.82	388362
CNN	3.05	0.53	92426
Transformer	3.06	0.53	4384042
	Condition monitoring of hydraulic systems		
LSTM	25.18	2.02	99715
CNN	25.22	1.56	36611
Transformer	25.19	1.54	570819

2.4 Особливості реалізації обраних архітектур нейронних мереж

Обрані нейронні мережі реалізовано засобами мови програмування Python

та фреймворку для роботи з глибоким навчанням PyTorch.

Для проведення експериментів додатково використано апаратний прискорювач — графічний процесор — який задіяно для отримання результатів на датасетах повної довжини (з усіма прикладами).

Для роботи з текстовими даними, одним з ключових елементів рекурентної та згорткової мережі є перетворення вхідної послідовності токенів у стиснуту векторну репрезентацію. Це було досягнуто, використавши шар torch.nn.Embedding у такій варіації:

```
if embeddings is not None:
    self.word_embeddings = nn.Embedding.from_pretrained(
        embeddings=embeddings.float(),
padding_idx=padding_idx
    )

else:
    assert vocab_size is not None and embedding_dim is not
None
    self.word_embeddings = nn.Embedding(vocab_size,
embedding_dim)
```

Дана реалізація дозволяє використати техніку перенесення знань (англ. transfer learning) з ембедінгів, навчених на великих корпусах тексту та відкритими для використання у дослідженнях [42]. Завантаження натренованої ембедінг-матриці наведено у наступному лістингу.

```
unk_token = "<unk>"
pad_token = "<pad>"

if embedding_type == EmbeddingType.glove:
    embedding = GloVe(
        name="6B",
        dim=100,
```

```
        cache=cache,
    )
else:
    raise RuntimeError(f"unknown embedding type:
{embedding_type}")
for extra_t in [unk_token, pad_token]:
    embedding.itos.append(extra_t)
    embedding.stoi[extra_t] = len(embedding.vectors)
    embedding.vectors = torch.vstack(
        [
            embedding.vectors,
            torch.zeros(embedding.dim).unsqueeze(0),
        ]
    )
```

Хоча обрана матриця складається з понад 40 000 векторів, до неї необхідно додати нульові вектори-представлення для двох спеціальних слів - “<unk>” та “<pad>”. Для обох слів мережа використовує нульовий вектор. Перше слово відповідає за слова у реченні, які не присутні у ембедінг-матриці. Друге використовується у випадках, коли у блоці даних на вході мережі присутні послідовності різної довжини. До них, за наведеним нижче лістингом, додається дане слово для отримання послідовностей однакової довжини.

```
text_list = torch.utils.rnn.pad_sequence(
    text_list,
    batch_first=True,
    padding_value=self.embedding.stoi[pad_token],
)
```

Як рекурентна, так і згорткова мережі використовують ембедінг матрицю над вхідними даними та звичайний лінійний шар безпосередньо для класифікації.

Для проведення експериментів використано ембедінг матрицю, яка переводить текст у вектор розмірності 100. Цим обрано компроміс між надійністю представлення слів (розмірності 10, 50 можуть завадити мережі отримати повний контекст слова у тексті) та швидкодією системи (вектори у

300-вимірному просторі варто обирати для задач з класифікацією довгих та складних документів). Залежно від задачі, кількість ваг в останньому шарі змінюється за рахунок повнозв'язного зв'язку. В той же час, суттєво відрізняються основні шари моделей.

У рекурентній мережі, це два шари torch.nn.LSTM, без модифікацій (шар не двонапрямлений, тощо). Загальна архітектура мережі наведена у наступному лістингу:

```
LSTMNet(
  (word_embeddings): Embedding(400002, 100, padding_idx=400001)
  (lstm_layers): LSTM(100, 256, num_layers=2, batch_first=True)
  (clf): Linear(in_features=256, out_features=2, bias=True)
)
```

Архітектура побудованої згорткової мережі наведена у наступному лістингу:

```
CNNNet(
  (word_embeddings): Embedding(400002, 100, padding_idx=2)
  (conv1): Sequential(
    (0): Conv1d(100, 64, kernel_size=(3,), stride=(1,),
padding=same)
    (1): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU()
  )
  (conv2): Sequential(
    (0): Conv1d(64, 128, kernel_size=(4,), stride=(1,),
padding=same)
    (1): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU()
  )
  (avgp): AvgPool1d(kernel_size=(8,), stride=(8,), padding=(0,))
  (clf): Linear(in_features=256, out_features=2, bias=True)
)
```

Інший метод реалізації було використано для побудови трансформерної архітектури. З теоретичних відомостей відомо, що дану модель неможливо навчити виконувати задачу з високою результативністю, якщо тренувати її з нуля на недостатньо великому наборі даних. Тому в даному випадку доцільно використати розглянуту для випадку векторного представлення слів техніку

передачі знань, використавши навчену на аналогічній задачі модель як основну, додавши необхідний класифікатор в якості вихідного шару. Така постановка задачі відома у літературі як файн-тюнинг моделі (англ. - fine-tuning).

З цих міркувань, для текстових задач за основу обрано модель DistilBERT, натреновані ваги якої доступні у відкритому доступі. До неї додається класифікатор відповідно задачі, що вирішується.

Реалізувати розглянутий патерн можна наступним чином:

```
self.backbone =
transformers.DistilBertModel.from_pretrained(
    "distilbert-base-uncased"
)
self.head = nn.Sequential(
     nn.Linear(in_features=bert_hidden_dim,
out_features=classifier_dim, bias=True),
     nn.ReLU(),
     nn.Dropout(0.3),
     nn.Linear(classifier_dim, num_labels),
)
```

, де типом основної мережі є "distilbert-base-uncased", що відповідає натренованій моделі DistilBERT на корпусі англомовного тексту малого регістру.

Варто зазначити, що дана модель вже має вбудований механізм переведення послідовного набору даних у стисле векторне представлення. Через це, використання ембедінг шару недоцільне в даному випадку. Повна архітектура мережі наведена у додатку Б.

Для вирішення класифікації часових рядів не всі моделі зазнали суттєвих змін. Рекурентна та згорткова мережі зберегли свою структуру практично в повному обсязі (їх реалізація для цього випадку

наведена у додатку В), за винятком ембедінг-шару. Трансформер-архітектуру побудовано згідно рекомендацій у [43], без використання техніки перенесення знань як для випадку текстових даних. Відповідну реалізацію наведено у додатку В.

За рахунок великої розмірності послідовностей у розглянутих задачах (до 370 елементів у векторі), оптимізаційний процес суттєво відрізнявся від випадку класифікації тексту. Так, для вирішення поставлених задач було адаптовано низку гіперпараметрів, таких як коефіцієнт навчання, сила регуляризації ваг, розклад зменшення коефіцієнту навчання, тощо. Цим вдалось вирішити проблему нестабільної процедури оптимізації, яка виникла при використанні однакових параметрів процесу для текстових даних та часових послідовностей.

3 Прискорення роботи моделей глибокого навчання засобами розподілених обчислень на прикладі парадигми MapReduce

3.1 Пришвидшення роботи алгоритмів глибокого навчання

Обробка великомасштабних даних є проблемою через обмеження алгоритмів машинного навчання з точки зору масштабованості. Наприклад, коли обчислювальна складність алгоритму перевищує основну пам’ять серверу, алгоритм не буде добре масштабуватися через дане обмеження. В цьому випадку необхідно перейти до розподілених алгоритмів машинного навчання. Розподілені алгоритми ML є невід'ємною частиною великомасштабного навчання через їхню здатність розподіляти процеси навчання на кількох робочих станціях, щоб зробити швидкість обробки цих даних прийнятною для подальшого прийняття рішень.

Розподілене машинне навчання відноситься до паралельних алгоритмів і систем машинного навчання, які призначені для підвищення продуктивності, підвищення точності та масштабування до більших розмірів вхідних даних. Збільшення розміру вхідних даних для багатьох алгоритмів може значно зменшити помилку навчання і часто може бути більш ефективним, ніж використання більш складних методів [29]. Розподілене машинне навчання дозволяє компаніям, дослідникам і окремим особам приймати зважені рішення та робити значущі висновки на основі великих обсягів даних.

Масштабування нейронних мереж по відношенню до кількості навчальних прикладів, кількості параметрів моделі, або і того, й іншого, може значно підвищити кінцеву точність класифікації [30]. Ці результати призвели до сплеску інтересу до масштабування алгоритмів машинного навчання та передбачення, що використовуються для цих моделей, а також до вдосконалення використовуваних процедур оптимізації.

Так, виконання алгоритмів машинного навчання в розподіленому середовищі спочатку передбачає концептуальне перетворення однопотокових

алгоритмів у паралельні алгоритми. Цей крок часто може бути найскладнішим, оскільки він залежить від алгоритму і вимагає, щоб користувач добре розумів основний алгоритм. Другий крок включає фактичну реалізацію паралельних алгоритмів, що вимагає від користувача розуміння семантики та часу виконання системи, щоб досягти правильного та ефективного паралельного виконання.

За рахунок складності завдання розробки паралельних алгоритмів та їх використання в індустрії, більшість сучасних досліджень з використанням розподіленого машинного навчання використовують стандартні способи паралелізму: паралелізм на даних та на моделі.

Паралелізм даних. Паралелізм даних — це техніка розпаралелювання, яка передбачає розбиття даних на блоки. У паралельних розподілених обчисленнях даних ми спочатку ділимо дані на кілька розділів, причому кількість розділів дорівнює кількості робочих машин (тобто обчислювальних вузлів). Потім ми дозволяємо кожному працівнику володіти одним незалежним розділом і дозволяємо їм виконувати обчислення над цими даними. Оскільки у нас є кілька вузлів, які сканують дані паралельно, ми повинні мати можливість сканувати більше даних, ніж при використанні одного вузла — ми збільшуємо пропускну здатність за допомогою розподілених паралельних обчислень.

У розподіленому машинному навчанні, де наша мета — прискорити зближення навчання моделі з використанням кількох вузлів, застосування паралелізму даних є досить інтуїтивним: ми дозволяємо кожному працівнику виконувати навчання (тобто

стохастичний градієнтний спуск) на власному розділі даних і генеруємо набір оновлення параметрів (тобто градієнтів) на ньому. Потім ми дозволяємо всім вузлам синхронізувати стан своїх параметрів мережевим зв’язком, поки вони не досягнуть консенсусу.

Паралелізм даних ефективний, коли розмір навчальних даних більший, оскільки можливо сканувати дані набагато швидше за рахунок додаткових вузлів.

Модельний паралелізм. У порівнянні з паралелізмом даних, паралелізм моделі є більш складним і неоднозначним поняттям. Взагалі кажучи, при паралелізмі моделі замість розділення даних ми намагаємося розділити саму модель машинного навчання, щоб розподілити робоче навантаження на кількох обчислювальних працівників. Наприклад, скажімо, що ми вирішуємо задачу факторізації матриці, де матриця дуже велика, і ми хочемо вивчити кожен параметр цієї величезної матриці. Щоб застосувати паралелізм моделі, ми повинні розділити матрицю на багато маленьких блоків (підматриць), а потім дозволити кожному працівнику подбати про декілька. Таким чином, ми можемо використовувати додаткову оперативну пам’ять кількох вузлів, якщо ОЗП на одному вузлі недостатньо для збереження всіх параметрів у матриці. Оскільки різні вузли мають різні робочі навантаження, які відображаються на різні блоки матриці, ми повинні отримати прискорення, коли вони обчислюють паралельно.

Паралелізм моделі застосовний у випадках, коли розмір моделі завеликий для одного обчислювального вузла, оскільки даний метод дозволяє розділяти моделі та використовувати додаткову пам’ять за рахунок кількох вузлів.

Недолік даного методу полягає у складності визначити кроки моделі, де можна використати паралелізм. Оскільки кожна модель машинного навчання має свої особливості, не існує принципового способу реалізації паралелізму

моделей.

3.2 Основні поняття розподілених обчислень. Парадигма MapReduce

До появи розподілених фреймворків користувачі повинні були створювати рукописні рішення, в яких вони несли виключну відповідальність за явний контроль усіх аспектів виконання. Цей схильний до помилок і трудомісткий процес включав керування розподілом даних, паралелізацією, синхронізацією та відмовостійкістю, що призвело до тривалого циклу розробки, коли користувачі мали труднощі з налагодженням існуючих алгоритмів та впровадженням нових. Однак нові рамки програмування, такі як MapReduce [31], значно спростили процес розподілених обчислень і дозволили користувачам легко масштабувати алгоритми до більших наборів даних. Ці фреймворки надають примітивам добре визначену семантику паралелізації разом із розподіленим середовищем виконання та файловою системою, що дозволяє користувачам зосередитися на реалізації алгоритмів, а не на управлінні низькорівневими деталями.

MapReduce є фреймворком (і разом з тим архітектурою) для обробки даних і був розроблений Google [32] для обробки даних у розподіленому середовищі. Архітектура складається з кількох етапів і використовує концепції з функціонального програмування. По-перше, всі дані розбиваються на кортежі (пари ключ-значення) під час фази map. Ця фаза може виконуватися повністю паралельно, оскільки немає залежності даних між відображенням функції на два різні значення в наборі. Під час фази перемішування (англ. shuffle), ці кортежі переміщуються між вузлами і передаються далі. Це необхідно, оскільки подальша агрегація відбувається на основі ключа, і необхідно переконатися, що всі кортежі, що належать одному ключу, обробляються одним і тим же вузлом для коректності результатів. На наступній фазі зведення (англ. reduce) виконується агрегація кортежів для створення єдиного вихідного значення на заданий ключ. Цей етап не можна розпаралелювати, оскільки кожен крок reduce залежить від попереднього кроку.

Основна перевага цього фреймворку полягає в тому, що дані можуть бути розділені між великою кількістю машин, тоді як завдання однієї фази не мають залежності від даних і тому можуть виконуватися повністю паралельно. Ці самі машини можуть бути вузлами в кластері сховища GFS (або подібних), так що замість переміщення даних у програму, програму можна перемістити до даних для врахування локальності даних і вищої продуктивності. Програма, на відміну від великої кількості даних, може бути легко передана по мережі потрібному вузлу. Крім того, відповідно до концепції масштабування, MapReduce реалізує відмовостійкість у програмному забезпеченні, відстежуючи стан робочих вузлів за допомогою спеціальних повідомлень і перепланування завдань, які не вдалося виконати на "здорові" вузли [33].

Як правило, розмір одиничного завдання дорівнює розміру окремого блоку у вхідному наборі даних, тому збій вузла повинен впливати лише на частину загальної програми, і система здатна відновлюватися при збої обробки цього блоку. MapReduce не дозволяє спілкуватися між робочими вузлами на етапі map. Натомість він дозволяє перехресний зв'язок лише під час фази змішування, між фазами map та reduce, для зменшення бар'єрів синхронізації та збільшення паралелізму.

MapReduce як фреймворк є власністю Google. Однак архітектура, яка стоїть за цим, була відтворена у фреймворку Hadoop з відкритим вихідним кодом. Він використовує HDFS там, де MapReduce використовує GFS, але схожий за своєю загальною архітектурою.

Разом з файловою системою, Hadoop став центральним інструментом роботи з великими даними на початку 21 ст., а парадигма MapReduce й досі широко використовується в низці рішень у індустрії.

3.3 Оптимізація швидкості класифікації текстових даних за допомогою MapReduce

В дослідженні [30] прийшли до висновку, що MapReduce, розроблений для паралельної обробки даних, погано підходить для ітераційних обчислень, властивих глибокому навчанню. Звідси, дослідження оптимізаційного процесу не є потенційним кандидатом на пришвидшення засобами MapReduce. Водночас, процес передбачення на великій кількості нерозмічених даних можна розглянути як випадок паралелізму даних з 3.1 та використати для цього MapReduce.

Параметри експерименту. Експеримент проводився з використанням датасету IMDb Movie Reviews, що складається з 50 тисяч прикладів. Для більш детального опису див. 2.1.1. Експериментальна задача — порахувати кількість позитивних та негативних відгуків користувачів IMDb. Кожен робочий процес у MapReduce отримує копію нейронної мережі (з класу рекурентних мереж) та працює над відповідною номеру процесу частиною даних. В рамках експерименту використано засоби мови Python для розподілених обчислень з використанням багатьох процесів. Кожен процес — окремий робітник в “кластері”, незалежний від інший.

Отримані результати можна зіставити тим, отриманим у середовищі з багатьма фізичними обчислювальними вузлами, оскільки в обох середовищах є справедливими такі умови, як: затрати на керування робітниками, незбалансований паралелізм відносно об'єму даних, та ін.

Схему роботи розробленої програми представлено на рис. 9.

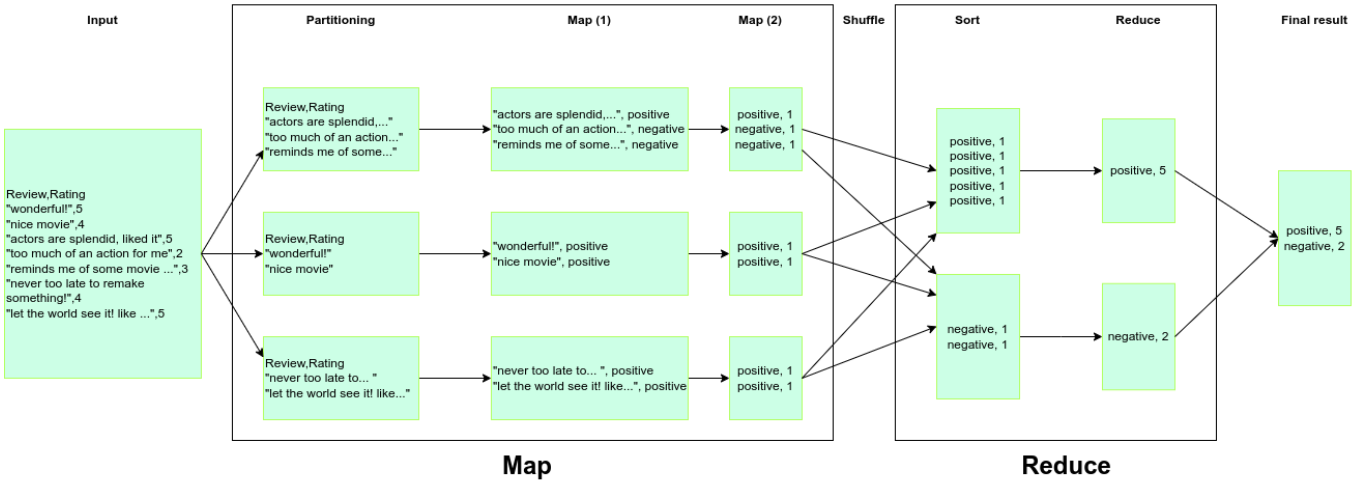

Рисунок 9 - Розроблена схема алгоритму MapReduce для пришвидшення передбачення моделі

Алгоритм роботи розробленої реалізації фреймворку MapReduce складається з трьох фаз:

1) map — локальна для робітника копія моделі робить передбачення для частини даних, які записуються у загальну файлову систему. Передбачення в даному випадку є категорією відгуку користувачів на деякий фільм: позитивний або негативний. Ключову логіку даного кроку наведено у наступному фрагменті коду.

```
def chunks_mapper(pid, chunk):
    pred_cats = []

    for i, line in enumerate(chunk):
        reader = csv.reader([line.strip()])
        for row in reader:
            reviews = dict(zip(columns, row))["Review"]
            tokens = vocab(tokenizer(reviews))
            tokens = torch.tensor(tokens).unsqueeze(dim=0)
            pred = model(tokens).squeeze()
            pred_cat_idx = torch.argmax(pred, dim=0).numpy()
            pred_cat = cats[pred_cat_idx]
            pred_cats.append(pred_cat)
    with open(f"{args.out_path}_{pid}.txt", "w+") as f:
        for cat in pred_cats:
            f.write(f"{cat}\t1\n")
```

У наведеному фрагменті, ‘pid’ - унікальний номер робітника в “кластері”,

chunk — виділений робітнику блок з даними, model — скопійована між робітниками навчена модель глибокого навчання. Відповідно до діаграми, результатом даної фази є набір файлів, збережених на диск робітником з відповідним ідентифікатором.

2) shuffle & sort — записані попередньою фазою дані зчитуються та сортуються у пам'яті.

3) reduce — дані агрегуються та рахується кількість позитивних та негативних відгуків, яка зберігається у вихідну хеш-таблицю.

Таблиця 5 — Порівняння звичайного передбачення з розподіленим за допомогою MapReduce

	Кількість робочих процесів	Час, с	% прискорення
	Кількість прикладів у датасеті = 20 500		
Звичайне передбачення	1	15.28	-
Модель з MapReduce	4	8.81	73.44
	8	9.34	63.69
	16	10.27	48.78
	32	9.84	55.28
	64	9.47	61.42
	128	9.63	58.69
	Кількість прикладів у датасеті = 41 000		
Звичайне передбачення	1	28.54	-
Модель з MapReduce	4	17.71	61.15
	8	19.09	49.5
	16	19.26	48.15
	32	19.48	46.54
	64	20.11	41.89

	128	19.47	46.58
	Кількість прикладів у датасеті = 61 500		
Звичайне передбачення	1	42.2	-
Модель з MapReduce	4	25.53	65.32
	8	27.84	51.59
	16	29.21	44.5
	32	30.55	38.13
	64	28.94	45.83
	128	29.17	44.68

Отримані у таблиці 5 результати свідчать про можливість прискорення передбачення моделі глибокого навчання до 75%, використовуючи 4 незалежних робітники у схемі паралелізму даних. Отже, парадигму MapReduce можна розглядати в якості способу пришвидшення часу передбачення моделі, а отже, часу прийняття рішення на основі великого набору даних.

Слід зазначити, що у мульти-серверному середовищі результати відрізнятимуться при фіксації інших параметрів експерименту за рахунок різниці у апаратурі, затримки на мережеву комунікацію серверів, тощо.

4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У даному розділі проведена оцінка основних функціональних та вартісних характеристик програмного продукту, розробленого для вирішення задачі фільтрування агресивних висловлювань в соціальних мережах.

Програмний продукти призначено для використання на персональних комп'ютерах під управлінням будь-якої операційної системи.

Нижче наведено аналіз різних варіантів реалізації модулю з метою вибору оптимальної стратегії створення програмного продукту, враховуючи при цьому як економічні фактори, так і на характеристики продукту, що впливають на продуктивність роботи і на його сумісність з апаратним забезпеченням. Для цього було використано апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз – це технологія оцінки реальної вартості продукту або послуги незалежно від організаційної структури компанії. Прямі та побічні витрати розподіляються по продуктам та послугам у залежності від потрібних обсягів ресурсів на кожному етапі виробництва. Виконані на цих етапах дії у контексті метода ФВА називаються функціями.

Мета ФВА полягає у забезпеченні найбільш оптимального розподілу ресурсів, що виділені на виробництво продукції або надання послуг, на прямі та непрямі витрати. У даному випадку – аналізу функцій програмного продукту й виявлення усіх витрат на реалізацію цих функцій.

Метод ФВА починається з визначення послідовності функцій, необхідних для виробництва продукту. Спочатку йдуть всі можливі функції, які розподіляються по двом групам: ті, що впливають на вартість продукту і ті, що не впливають. Також на цьому етапі оптимізується сама послідовність скороченням кроків, що не впливають на витрати.

Для кожної функції визначається повний обсяг річних витрат та кількість робочих часів. На основі даних оцінок визначається кількісна характеристика

джерел витрат. Після визначення джерел витрат проводиться кінцевий розрахунок витрат на виробництво продукту.

4.1 Постановка задачі техніко-економічного аналізу

Метод ФВА був застосований для проведення техніко-економічний аналізу розробки системи для виявлення та фільтрування агресивних висловлювань в коментарях в соціальних мережах. Оскільки основні проектні рішення стосуються всієї системи, кожна окрема підсистема має їм задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмного продукту, призначеного обробки та фільтрування даних.

До продукту були визначені наступні технічні вимоги:

1) можливість виконання на комп’ютерах без відеокарти;
2) зручність та простота аналізу експериментів;
3) мінімальні витрати на впровадження програмного продукту;
4) можливість зручного налаштування та масштабування під інші набори даних.

4.1.1 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка програми, що дозволяє проводити експерименти над низкою нейронних мереж з наборами даних різного походження. Виходячи з конкретної мети, можна виділити наступні основні функції програмного продукту:

F_1 – вибір мови програмування;

F_2 – вибір бібліотек для машинного навчання;

F_3 – вибір середовища розробки.

Кожна з основних функцій може мати декілька варіантів реалізації.

Функція F_1:

1) Python
2) R

Функція F_2:

1) Pytorch
2) Tensorflow

Функція F_3:

1) PyCharm CE
2) Visual Studio Code

4.1.2 Варіанти реалізації основних функцій

Варіанти реалізації основних функцій наведені у морфологічній карті системи на рисунку . Морфологічна карта відображує всі можливі комбінації варіантів реалізації функцій, які складають повну множину варіантів ПП.

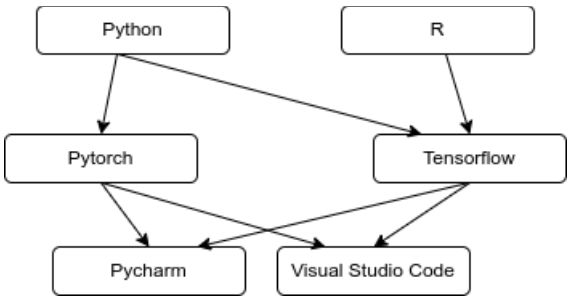

Рисунок 10 – Морфологічна карта варіантів реалізації функцій

На основі цієї карти побудовано позитивно-негативну матрицю варіантів основних функцій (табл. 6).

Таблиця 6 – Позитивно-негативна матриця варіантів основних функцій

Функція	Варіант реалізації	Переваги	Недоліки
F_1	А	Займає менше часу на написання коду, кросплатформений, може бути використаний для будь-яких мереж	Займає більше часу для виконання коду
	Б	Кросплатформений, займає менше часу на виконання коду	Займає більше часу на написання коду
F_2	А	Безкоштовність, розгорнута документація, постійно оновлюється	Підтримка лише одної мови
	Б	Безкоштовність, розгорнута документація, підтримка багатьох мов	Відсутність спільноти для звітування проблем
F_3	А	Гарна підтримка, легкість у використанні, швидкодія	Тільки для Python
	Б	Безкоштовний, легковісний, кросплатформений засіб	Необхідність довгих налаштувань для оптимізації роботи

На основі аналізу позитивно-негативної матриці робимо висновок, що при

розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція F_1:

Оскільки реалізація програмного коду для вирішення поставленої задачі є концептуально складною, необхідно обрати мову, що спрощує написання коду для виконавця, тому варіант Б має бути відкинутий.

Функція F_2:

Вибір фреймворку машинного навчання не відіграє велику роль у даному програмному продукту, тому вважаємо варіанти А та Б гідними розгляду.

Функція F_3:

Оскільки, програмний продукт реалізується мовою Python, але варіант Б легший у використанні та має більше можливостей до кастомізації, обираємо його.

Таким чином, будемо розглядати такі варіанти реалізації програмного продукту:

- F_1(А) – F_2(А) – F_3(Б)
- F_1(А) – F_2(Б) – F_3(А)
- F_1(А) – F_2(Б) – F_3(Б)
- F_1(А) – F_2(А) – F_3(А)

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.2 Обґрунтування системи параметрів програмного продукту

4.2.1 Опис параметрів

На підставі даних про основні функції, що повинен реалізувати програмний продукт, вимог до нього, визначаються основні параметри виробу, що будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб схарактеризувати програмний продукт, будемо використовувати наступні параметри:

X1 – швидкодія операцій мови програмування в залежності від обраної серверної технології;

X2 – об’єм пам’яті для збереження даних персонального комп’ютера, необхідний для збереження та обробки даних під час виконання програми;

X3 – час, який витрачається на виконання коду;

X4 – потенційний об’єм програмного коду, який необхідно створити безпосередньо розробнику.

4.2.2 Кількісна оцінка параметрів

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію програмного продукту як показано у таблиці 7.

Таблиця 7 – Основні параметри програмного продукту

Опис параметру	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	X1	мс/оп	3400	12000	20000
Об’єм пам’яті для збереження даних	X2	Мб	128	64	16
Час виконання коду	X3	с	1000	600	200
Потенційний об’єм програмного коду	X4	строк коду	3000	1500	800

За даними таблиці 7 будуються графічні характеристики параметрів (див. Рис. 11-14).

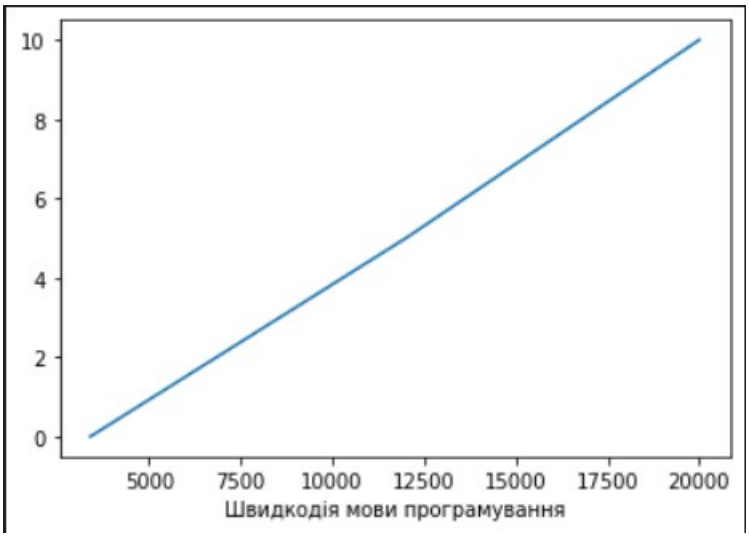

Рисунок 11 – Швидкодія мови програмування

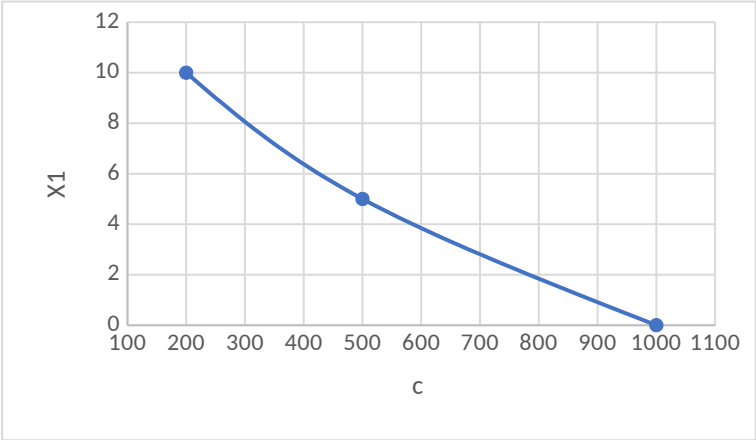

Рисунок 12 – Об’єм пам’яті для збереження даних

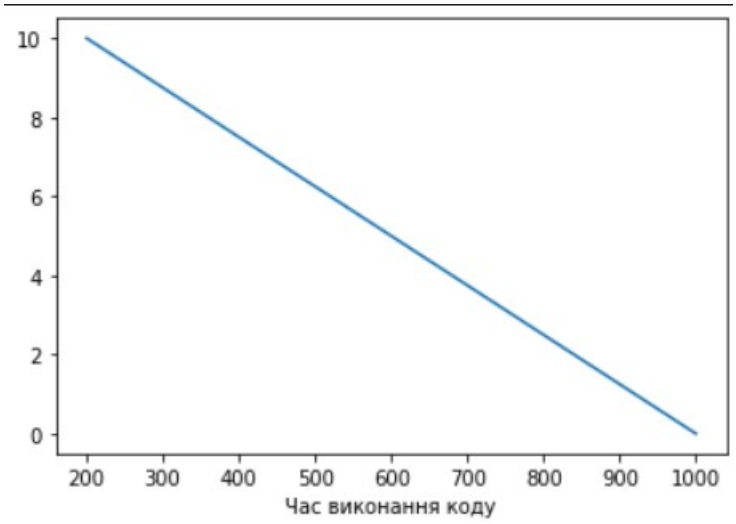

Рисунок 13 – Час виконання коду

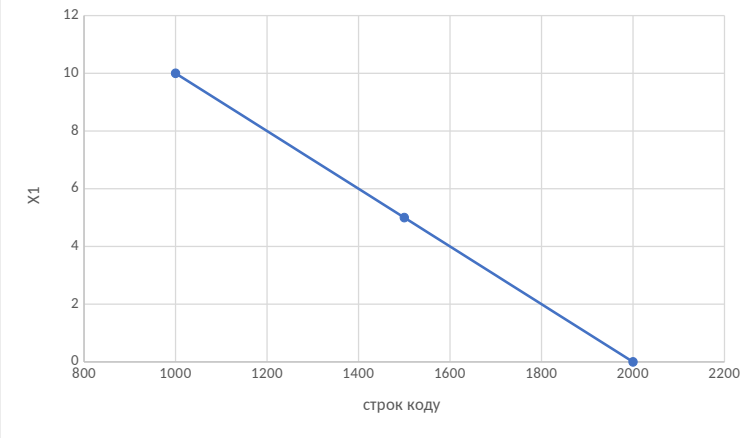

Рисунок 14 – Потенційний об’єм програмного коду

4.2.3 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який має найбільш зручний інтерфейс та зрозумілу взаємодію з користувачем

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 5 людей. Визначення коефіцієнтів значимості передбачає:

1) визначення рівня значимості параметра шляхом присвоєння різних рангів;
2) перевірку придатності експертних оцінок для подальшого використання;
3) визначення оцінки попарного пріоритету параметрів;
4) обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 5.

Таблиця 8 – Результати ранжування показників

Параметр	Ранг параметра за оцінкою експерта					Сума рангів	Відхилення, Δ_i	Δ_i^2
	1	2	3	4	5			
X1	3	4	3	4	4	18	+5,5	30,25
X2	2	2	1	1	2	8	–4,5	20,25
X3	1	1	2	2	1	7	–5,5	30,25
X4	4	3	4	3	3	17	+4,5	20,25
Разом	10	10	10	10	10	50	0	101

Для перевірки ступеню достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} = 50,$$

де r_{ij} – ранг i-го параметра, визначений j-м експертом; N – число експертів.

б) середня сума рангів T:

$$T = \frac{1}{n} R_i = 12{,}5.$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T\,.$$

г) загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{n} \Delta_i^2 = 101.$$

д) коефіцієнт узгодженості (конкордації):

$$W = \frac{12\,S}{N^2\left(n^3 - n\right)} = \frac{12 \cdot 101}{5^2\left(4^3 - 4\right)} = 0{,}808 > W_k = 0{,}67.$$

Ранжирування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67. Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 6. Числове значення, що визначає ступінь переваги *i*-го параметра над *j*-тим, a_{ij} визначається за формулою:

$$a_{ij} = \begin{cases} 1{,}5 & x_i > x_j; \\ 1{,}0 & x_i = x_j; \\ 0{,}5 & x_i < x_j. \end{cases}$$

Таблиця 9 – Результати ранжування параметрів

Параметри	Експерти					Підсумкова оцінка	Числове значення коефіцієнтів переваги
	1	2	3	4	5		
X1,X2	>	>	>	>	>	>	1,5
X1,X3	>	>	>	>	>	>	1,5
X1,X4	<	>	>	<	>	>	1,5
X2,X3	>	<	<	>	>	>	1,5
X2,X4	<	<	<	<	<	<	0,5
X3,X4	<	<	<	<	<	<	0,5

З отриманих числових оцінок переваги складемо матрицю $A=\|a_{ij}\|$. Для кожного параметра розрахунок вагомості K_{B_i} проводиться за наступною формулою:

$$K_{B_i}=\frac{b_i}{\sum_{i=1}^{n} b_i},$$

де $b_i=\sum_{j=1}^{N} a_{ij}$ – вагомість i-го параметра за результатами оцінок всіх

експертів;

a_{ij} –коефіцієнт переваги *i*-го на *j*-тим параметром.

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступною формулою:

$$K_{B_i} = \frac{b_i^{'}}{\sum_{i=1}^{n} b_i^{'}},$$

де $b_i^{'} = \sum_{j=1}^{N} a_{ij} b_j.$

Як видно з таблиці 7, різниця значень коефіцієнтів вагомості після другої ітерації не перевищує 2%, тому додаткові ітерації не потрібні.

Таблиця 10 – Розрахунок вагомості параметрів

i	j				Перша ітерація		Друга ітерація	
	X1	X2	X3	X4	B_i	K_{B_i}	B_i^1	$K_{B_i}^1$
X1	1,0	1,5	1,5	1,5	5,5	0,344	21,25	0,36
X2	0,5	1,0	1,5	0,5	3,5	0,219	12,25	0,208
X3	0,5	0,5	1,0	0,5	2,5	0,156	9,25	0,157
X4	0,5	1,5	1,5	1,0	4,5	0,281	16,25	0,275

Разом					16,0	1,0	59,0	1,0

4.3 Аналіз рівня якості варіантів реалізації функцій

Рівень якості кожного варіанту виконання основних функцій визначається окремо. Абсолютні значення параметрів X1(швидкодія мови програмування) та X2 (об'єм пам'яті для збереження даних) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра X3 (час розрахунків) обрано не найгіршим (не максимальним), тобто це значення відповідає варіанту б) 600 або в) 200 с.

Абсолютне значення параметра X4 (кількість строк коду) обрано не найгіршим, тобто це значення відповідає варіанту б) 1500 або в) 1000 строк коду.

Коефіцієнт технічного рівня якості для кожного варіанта реалізації ПП розраховується за формулою:

$$K_{TP} = \sum_{i=1}^{n} K_{B_i} B_i,$$

де n – кількість параметрів;
K_{B_i} – коефіцієнт вагомості і–го параметра;
B_i – оцінка i–го параметра в балах.

Розрахунок показників рівня якості представлено відповідно в таблиці 8.

Таблиця 11 – Розрахунок показників якості

Основні функції	Варіант реалізації	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F_1	А	20000	10	0,36	3.6
F_2	А	64	5	0,208	1.04
F_2	Б	16	10	0,208	2.08
F_3	А	600	5	0,157	1,57
	Б	1000	0	0,157	0
F_4	А	3000	0	0,275	2,75
	Б	1500	5	0,275	1,375

За цими даними визначаємо рівень якості кожного з варіантів:

– $F_1(А) – F_2(А) – F_3(Б) – F_4(Б) = 3.6 + 1.04 + 1.57 + 1.375 = 7.597$

– $F_1(А) – F_2(Б) – F_3(А) – F_4(А) = 3.6 + 2.08 + 0 + 2.75 = 8.43$

– F_1(А) – F_2(Б) – F_3(Б) – F_4(А) = 3.6 + 2.08 + 1.57 + 2.75 = 8.0

– F_1(А) – F_2(А) – F_3(А) – F_4(А) = 3.6 + 1.04 + 0 + 2.75 = 7.39

Отже, найкращим є перший варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.4 Економічний аналіз варіантів розробки програмного продукту

Для визначення вартості розробки програмного продукту спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;
2. Розробка алгоритму збору даних.

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовує інформацію у вигляді даних, а завдання 2 використовує стандартні методи збору даних. Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань. Загальна трудомісткість обчислюється за формулою.

$$T_O = T_P \cdot K_П \cdot K_{СК} \cdot K_М \cdot K_{СТ} \cdot K_{СТ.М}$$

де T_P – трудомісткість розробки програмного продукту;
$K_П$ – коригуючий коефіцієнт;
$K_{СК}$ – коефіцієнт на складність вхідної інформації;
$K_М$ – коефіцієнт рівня мови програмування;
$K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;
$K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення.

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: $T_P = 78$ людино-днів. Коригуючий коефіцієнт, який враховує вид

вхідної інформації для першого завдання: $K_П = 1,7$. Коригуючий коефіцієнт, який враховує складність контролю вхідної та вихідної інформації рівний $K_{СК} = 1$. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $K_{СТ} = 0,8$. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 78 \cdot 1.7 \cdot 0.8 = 106,08$$

людино-днів. Проведемо аналогічні розрахунки для другого завдання, в якому використовується алгоритм третьої групи складності зі ступенем новизни Б, тобто $T_Р = 25$ людино-днів, $K_П = 0,9, K_{СК} = 1$, $K_{СТ} = 0,8$:

$$T_2 = 25 \cdot 0.9 \cdot 0.8 = 18$$

людино-днів. Оскільки загальна трудомісткість усіх варіантів реалізації збігаються, їх можна об’єднати в одну групу.

Загальна трудомісткість складає:

$$T_О = (106,08 + 18) \cdot 8 = 996,64$$

людино-годин; В розробці беруть участь програміст з окладом 20000 грн., один аналітик в області даних з окладом 25000 грн, та один інженер даних з окладом 18000 грн. Визначимо середню зарплату за годину за формулою:

$$C_ч = \frac{M}{T_m \cdot t} грн.,$$

де M – місячний оклад працівників; T_m – кількість робочих днів на місяць; t – кількість робочих годин в день.

$$C_ч = \frac{20000 + 25000 + 18000}{3 \cdot 20 \cdot 8} = 131{,}25\ грн.$$

Тоді, розрахуємо заробітну плату за формулою:

$$C_{ЗП} = C_ч \cdot T_i \cdot K_Д,$$

де $C_ч$ – величина погодинної оплати праці програміста; T_i – трудомісткість відповідного завдання; $K_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

$$C_{ЗП} = 131{,}25 \cdot 979{,}2 \cdot 1{,}2 = 127\,315{,}59\ грн.$$

Відрахування на соціальний внесок становить 22%:

$$C_{СВ} = C_{ЗП} \cdot 0{,}22 = 127\,315{,}59 \cdot 0{,}22 = 28\,009{,}43\ грн.$$

Тепер визначимо витрати на оплату однієї машино-години. Оскільки одна ЕОМ обслуговується одним інженером апаратного забезпечення з окладом 15000 грн. та коефіцієнтом зайнятості $K_З = 0{,}2$ то для однієї машини отримаємо:

$$C_Г = 12 \cdot M \cdot K_З = 12 \cdot 15\,000 \cdot 0{,}2 = 36\,000\ грн.$$

З урахуванням додаткової заробітної плати:

$$C_{ЗП} = C_Г \cdot (1 + K_З) = 36\,000 \cdot (1 + 0{,}2) = 43\,200\ грн.$$

Відрахування на соціальний внесок становить 22%:

$$C_{СВ}=C_{ЗП}\cdot 0{,}22=43\,200\cdot 0{,}22=9\,504\,грн.$$

Амортизаційні відрахування розраховуємо за формулою при амортизації 25% та вартості ЕОМ – 30 000 грн.:

$$C_A=K_{ТМ}\cdot K_A\cdot Ц_{ПР}=1{,}15\cdot 0{,}25\cdot 20000=8\,625\,грн.$$

де $K_{ТМ}$ – коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;
K_A – річна норма амортизації;
$Ц_{ПР}$ – договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо за формулою:

$$C_P=K_{ТМ}\cdot K_P\cdot Ц_{ПР}=1{,}15\cdot 0{,}05\cdot 30000=1\,725\,грн.$$

де K_P– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$T_{ЕФ}=(Д_K-Д_B-Д_C-Д_P)\cdot t\cdot K_B, T_{ЕФ}=(365-104-12-16)\cdot 8\cdot 0{,}9=1\,667{,}6\,год.$$

де $Д_K$ – календарна кількість днів у році;
$Д_B, Д_C$ – відповідно кількість вихідних та святкових днів;
$Д_P$ – кількість днів планових ремонтів устаткування;
t – кількість робочих годин в день;
K_B – коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$C_{ЕЛ}=T_{ЕФ}\cdot N_C\cdot K_З\cdot Ц_{ЕЛ}=1\,677{,}6\cdot 0{,}65\cdot 0{,}2\cdot 4{,}29=935{,}59\,грн.$$

де N_C – середньо-споживча потужність приладу;
$K_З$ – коефіцієнтом зайнятості приладу;
$Ц_{ЕЛ}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$C_H = Ц_{ПР} \cdot 0{,}67 = 30\,000 \cdot 0{,}67 = 20100\,грн.$$

Тоді, річні експлуатаційні витрати будуть складати:

$$C_{ЕК} = C_{ЗП} + C_{СВ} + C_A + C_P + C_{ЕЛ} + C_H, C_{ЕК} = 43\,200 + 9504 + 8\,625 + 1\,725 + 828{,}06 + 20\,100 = 83\,982{,}06\,грн.$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$C_{МГ} = \frac{C_{ЕК}}{T_{ЕФ}} = \frac{83\,982{,}06}{1\,677{,}6} = 50.38\,грн/год.$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу складають:

$$C_M = C_{МГ} \cdot T = 50.38 \cdot 996{,}64 = 50\,178{,}48\,грн.$$

Накладні витрати складають 67% від заробітної плати:

$$C_H = C_{ЗП} \cdot 0{,}67 = 127\,315{,}59 \cdot 0{,}67 = 85\,301{,}45\,грн.$$

Отже, вартість розробки програмного продукту за варіантами становить:

$$C_{ПП} = C_{ЗП} + C_{СВ} + C_M + C_H, C_{ПП} = 127\,315{,}59 + 9504 + 50\,178{,}48 + 85\,301{,}45 = 272\,299{,}52\,грн.$$

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$K_{ТЕР} = \frac{K_K}{C_{ПП}} = \frac{7{,}16}{272\,299{,}52} = 2{,}63 \cdot 10^{-5}$$

4.5 Висновки

В результаті виконання економічного розділу були систематизовані і закріплені теоретичні знання в галузі економіки та

організації виробництва використанням їх для техніко-економічного обґрунтування розробки методом функціонально-вартісного аналізу.

В даному розділі проведено повний функціонально-вартісний аналіз ПП, який було розроблено в рамках дипломного проекту. Процес аналізу можна умовно розділити на дві частини. В першій з них проведено дослідження ПП з технічної точки зору: було визначено основні функції ПП та сформовано множину варіантів їх реалізації; на основі обчислених значень параметрів, а також експертних оцінок їх важливості було обчислено коефіцієнт технічного рівня, який і дав змогу визначити оптимальну з технічної точки зору альтернативу реалізації функцій ПП. Другу частину ФВА присвячено вибору із альтернативних варіантів реалізації найбільш економічно обґрунтованого.

На основі даних про зміст основних функцій, які повинен реалізувати програмний продукт, були визначені чотири найбільш перспективні варіанти реалізації продукту.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є другий варіант реалізації програмного продукту. який дає максимальну величину коефіцієнта техніко-економічного рівня, вартість витрат для нього становить $C_{ПП}=272\,299,52$ грн.

Цей варіант передбачає:

- мову програмування Python;
- бібліотеку машинного навчання Pytorch;
- зберігання експериментальних даних у файл;
- використання графіків та таблиць;
- середовище розробки Visual Studio Code;

ВИСНОВКИ

У роботі досліджено проблему використання нейронних мереж для класифікації текстових даних та часових послідовностей. Для цього було обрано представників низки архітектур різних класів нейронних мереж, які реалізовано використовуючи сучасний інструментарій мови Python. Оскільки використання нейронних мереж передбачає значну кількість обчислень, що зі збільшенням кількості даних стає неможливо використати для прийняття рішень, розглянуто механізм пришвидшення роботи нейронних мереж на етапі передбачення за допомогою парадигми розподілених обчислень MapReduce.

За результатами можна сказати, що всі розглянуті класи нейронних мереж є придатними до використання у сценаріях класифікації текстів та часових рядів у реальному світі. Проте таке використання означає прийняття низки компромісних рішень, які визначать баланс між точністю, складністю та часом роботи моделі.

Згорткова архітектура, яка виявилась найбільш схильною до запам'ятовування тренувальних даних, має значно меншу кількість параметрів і, за використання відповідних технік регуляризації ваг, потенційно є найкращим кандидатом для роботи в реальному часі. Водночас трансформер-архітектура є найпотужнішою з розглянутих, і може вивчити складні відображення між послідовністю та відповідною їй міткою значно краще від інших мереж. Однак, це призводить до неприйнятного у низці сценаріїв часу роботи, який в десятки разів може перевищувати показник класичної рекурентної мережі, як на етапі тренування, так і на етапі передбачення. Це обмеження значно звужує використання трансформерів до сценаріїв, де затримка моделі не є критичною у прийнятті рішення. В свою чергу, частина цих випадків має справу з великими даними, які можуть стати проблемою з урахуванням значної затримки у разі звичайного передбачення з даною моделлю на всьому датасеті.

Втім, вирішити проблему великої затримки передбачення можливо, використовуючи методи розподілених обчислень. Фреймворк MapReduce

дозволяє втілити паралелізм над даними у випадку передбачення важкою нейронною мережею, значно зменшивши загальну затримку до рівня затримки на декількох блоках цих даних. Так у мульти-вузловому середовищ, модель копіюється на сервери, які оброблюють відповідні партиції даних, що дозволяє досягати прискорення доки такі властивості розподіленого середовища як мережева комунікація не стають основною причиною затримки.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Alexander L. Fradkov, Early History of Machine Learning, IFAC-PapersOnLine, Volume 53, Issue 2, 2020, Pages 1385-1390, ISSN 2405-8963.
2. Sarker, I.H. Machine Learning: Algorithms, Real-World Applications and Research Directions. SN COMPUT. SCI. 2, 160 (2021).
3. Machine Learning: The New 'Big Thing' for Competitive Advantage - Scientific Figure on ResearchGate. Режим доступу: https://www.researchgate.net/figure/ML-applications-in-different-industries_tbl2_327215281.
4. Adeel A, Gogate M, Hussain A. Contextual deep learning-based audio-visual switching for speech enhancement in real-world environments. Inf Fusion. 2020;59:163–70.
5. Tian H, Chen SC, Shyu ML. Evolutionary programming based deep learning feature selection and network construction for visual data classification. Inf Syst Front. 2020;22(5):1053–66.
6. Piccinini, Gualtiero. (2004). The First Computational Theory of Mind and Brain: A Close Look at Mcculloch and Pitts's “Logical Calculus of Ideas Immanent in Nervous Activity”. Synthese. 141. 10.1023/B:SYNT.0000043018.52445.3e.
7. Sánchez-DelaCruz, Eddy & Lara Alabazares, David. (2019). Deep learning: concepts and implementation tools.
8. Hornik K., Stinchcombe M., White H. (1989) Multilayer feedforward networks are universal approximators Neural Networks 2:359–366.
9. LeCun Y., Bengio Y., Hinton G. (2015) Deep learning Nature 521:436–444.
10. Klindt D., Ecker A., Euler T, Bethge M. (2017) Neural system identification

for large populations separating “what” and “where.”, Advances in Neural Information Processing Systems. pp. 3509–3519.

11. Rumelhart, D., Hinton, G. & Williams, R. Learning representations by back-propagating errors. Nature 323, 533–536 (1986).
12. Sarker, I.H. Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions. SN COMPUT. SCI. 2, 420 (2021).
13. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, et al. Scikit-learn: machine learning in python. J Mach Learn Res. 2011;12:2825–30.
14. LeCun Y, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. Proc IEEE. 1998;86(11):2278–324.
15. Alzubaidi, L., Zhang, J., Humaidi, A.J. et al. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. J Big Data 8, 53 (2021).
16. Dupond S. A thorough review on the current advance of neural network structures. Annu Rev Control. 2019;14:200–30.
17. Hochreiter S, Schmidhuber J. Long short-term memory. Neural Comput. 1997;9(8):1735–80.
18. Siami-Namini S, Tavakoli N, Namin AS. The performance of lstm and bilstm in forecasting time series. In: 2019 IEEE International Conference on Big Data (Big Data), 2019; p. 3285–292. IEEE.
19. Cho K, Van MB, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, Bengio Y. Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078, 2014.
20. Chung J, Gulcehre C, Cho KH, Bengio Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555, 2014.
21. Jonas Gehring, Michael Auli, David Grangerier, Denis Yarats, and Yann N. Dauphin . Convolutions for sequential data. CoRR, abs/1705.03122, 2017.

22. Parikh, Ankur P., et al. "A decomposable attention model for natural language inference." arXiv preprint arXiv:1606.01933 (2016).
23. Khan, Salman, et al. "Transformers in vision: A survey." ACM Computing Surveys (CSUR) (2021).
24. Dietterich, T.G. (2002). Machine Learning for Sequential Data: A Review. In: Caelli, T., Amin, A., Duin, R.P.W., de Ridder, D., Kamel, M. (eds) Structural, Syntactic, and Statistical Pattern Recognition. SSPR /SPR 2002. Lecture Notes in Computer Science, vol 2396. Springer, Berlin, Heidelberg.
25. Список використовуваних у дослідженнях текстових датасетів – Papers with code. (загол. з екрана). URL : https://paperswithcode.com/datasets?task=text-classification. (дата звернення 05.05.2022).
26. Текстові датасетів – Huggingface (загол. з екрана). URL : https://huggingface.co/datasets. (дата звернення 05.05.2022).
27. Список використовуваних у дослідженнях наборів даних з числовими рядами – Papers with code. (загол. з екрана). URL : https://paperswithcode.com/datasets?task=time-series-classification. (дата звернення 05.05.2022).
28. Датасети з часовими рядами – TimeSeriesClassification (загол. з екрана). URL : https://www.timeseriesclassification.com. (дата звернення 05.05.2022).
29. Halevy A, Norvig P, Pereira F. The unreasonable effectiveness of data. IEEE Intell Syst. 2009;24(2): 8–12.
30. Large Scale Distributed Deep Networks, by Jeffrey Dean, Greg

Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Quoc V. Le, Andrew Y. Ng.

31. Dean J, Ghemawat S. Mapreduce: simplified data processing on large clusters. In: Proceedings of the 6th Conference on Symposium on Operating Systems Design andImplementation(OSDI'04). USENIXAssociation; 2004.
32. Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. Communications of the ACM, 51(1), 107-113.
33. Verbraeken, Joost, et al. "A survey on distributed machine learning." ACM Computing Surveys (CSUR) 53.2 (2020): 1-33.
34. Widiastuti, Nelly. (2019). Convolution Neural Network for Text Mining and Natural Language Processing. IOP Conference Series: Materials Science and Engineering. 662. 052010. 10.1088/1757-899X/662/5/052010.
35. Cheng, Lei, et al. "Classification of Long Sequential Data using Circular Dilated Convolutional Neural Networks." arXiv preprint arXiv:2201.02143 (2022).
36. M. Minsky, and S. Papert (1969) Perceptrons: An Introduction to Computational Geometry MIT Press, Cambridge, MA, USA.
37. Sun, Ruoyu. "Optimization for deep learning: theory and algorithms." arXiv preprint arXiv:1912.08957 (2019).
38. Werbos, P. 1974. Beyond regression: new tools for prediction and analysis in the behavioral sciences. Ph.D. thesis, Harvard University.
39. Rumelhart, D. E., G. E. Hinton, and R. J. Williams. 1986. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, Parallel Distributed Processing, volume 2, pages 318–362. MIT Press.
40. Werbos, P. J. 1990. Backpropagation through time: what it does and how to do it. Proceedings of the IEEE, 78(10):1550–1560.

41. Speech and Language Processing. Daniel Jurafsky & James H. Martin, arXiv pre-print, 2022.
42. Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global Vectors for Word Representation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.
43. Класифікація часових рядів з Трансформером – Keras. (загол. з екрана). URL : https://keras.io/examples/timeseries/timeseries_transformer_classification/. (дата звернення 09.05.2022).

ДОДАТОК А

Деталі алгоритму GloVe

Модель GloVe навчається на ненульових записах глобальної матриці одночасного зустрічання пар слово-слово, яка відображає, як часто слова зустрічаються одне з одним у заданому корпусі. Заповнення цієї матриці вимагає одного проходу через весь корпус для збору статистичних даних. Для великих корпусів цей крок може бути дорогим з точки зору обчислень, але це одноразова процедура. Подальші ітерації навчання відбуваються набагато швидше, оскільки кількість ненульових записів матриці зазвичай набагато менша за загальну кількість слів у корпусі.

GloVe є логарифмично-білінійною моделлю зі зваженою функцією найменших квадратів в якості цілі оптимізації. Основним принципом, що лежить в основі моделі, є спостереження, що співвідношення ймовірностей спільного зустрічання двох слів може нести певну форму інформації.

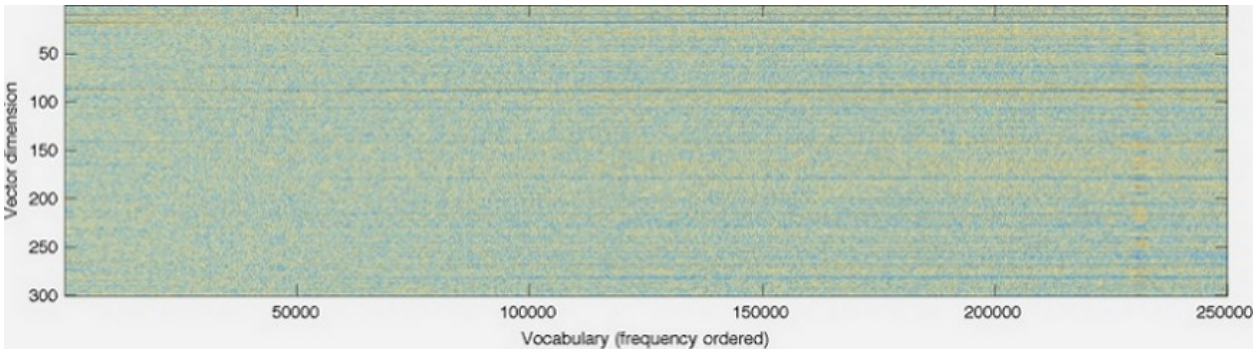

Рисунок 1 - Графічне представлення натренованої матриці GloVe (розмірність 100 використана для експериментів

Мета навчання GloVe полягає в тому, щоб вивчити вектори слів таким чином, щоб їх скалярний добуток дорівнював логарифму ймовірності слів, що зустрілися разом у корпусі. У зв'язку з тим, що логарифм дробу дорівнює різниці логарифмів, ця ціль пов'язує (логарифм) відношення ймовірностей спільної появи слів з векторними різницями у векторному просторі слів. Оскільки ці співвідношення можуть кодувати певну форму значення, ця інформація також кодується як векторні різниці. З цієї причини отримані вектори слів можна успішно використовувати для задач, які передбачають математичні операції над звичайними словами.

ДОДАТОК Б

Архітектура мережі класу “Трансформер”

Для задачі бінарної класифікації тексту, з основною моделлю DistilBERT, архітектура має наступну структуру:

```
Transformer(
  (hf_model): DistilBertModel(
    (embeddings): Embeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (transformer): Transformer(
      (layer): ModuleList(
        (0): TransformerBlock(
          (attention): MultiHeadSelfAttention(
            (dropout): Dropout(p=0.1, inplace=False)
            (q_lin): Linear(in_features=768, out_features=768,
bias=True)
            (k_lin): Linear(in_features=768, out_features=768,
bias=True)
            (v_lin): Linear(in_features=768, out_features=768,
bias=True)
            (out_lin): Linear(in_features=768, out_features=768,
bias=True)
          )
          (sa_layer_norm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
          (ffn): FFN(
            (dropout): Dropout(p=0.1, inplace=False)
            (lin1): Linear(in_features=768, out_features=3072,
bias=True)
            (lin2): Linear(in_features=3072, out_features=768,
bias=True)
            (activation): GELUActivation()
          )
          (output_layer_norm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
        )
        (1): TransformerBlock(
          (attention): MultiHeadSelfAttention(
```

```
            (dropout): Dropout(p=0.1, inplace=False)
            (q_lin): Linear(in_features=768,
out_features=768, bias=True)
            (k_lin): Linear(in_features=768,
out_features=768, bias=True)
            (v_lin): Linear(in_features=768,
out_features=768, bias=True)
            (out_lin): Linear(in_features=768,
out_features=768, bias=True)
          )
          (sa_layer_norm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
          (ffn): FFN(
            (dropout): Dropout(p=0.1, inplace=False)
            (lin1): Linear(in_features=768,
out_features=3072, bias=True)
            (lin2): Linear(in_features=3072,
out_features=768, bias=True)
            (activation): GELUActivation()
          )
          (output_layer_norm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
        )
        (2): TransformerBlock(
          (attention): MultiHeadSelfAttention(
            (dropout): Dropout(p=0.1, inplace=False)
            (q_lin): Linear(in_features=768,
out_features=768, bias=True)
            (k_lin): Linear(in_features=768,
out_features=768, bias=True)
            (v_lin): Linear(in_features=768,
out_features=768, bias=True)
            (out_lin): Linear(in_features=768,
out_features=768, bias=True)
          )
          (sa_layer_norm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
          (ffn): FFN(
            (dropout): Dropout(p=0.1, inplace=False)
            (lin1): Linear(in_features=768,
out_features=3072, bias=True)
            (lin2): Linear(in_features=3072,
out_features=768, bias=True)
            (activation): GELUActivation()
```

```
          )
          (output_layer_norm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
        )
        (3): TransformerBlock(
          (attention): MultiHeadSelfAttention(
            (dropout): Dropout(p=0.1, inplace=False)
            (q_lin): Linear(in_features=768, out_features=768,
bias=True)
            (k_lin): Linear(in_features=768, out_features=768,
bias=True)
            (v_lin): Linear(in_features=768, out_features=768,
bias=True)
            (out_lin): Linear(in_features=768, out_features=768,
bias=True)
          )
          (sa_layer_norm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
          (ffn): FFN(
            (dropout): Dropout(p=0.1, inplace=False)
            (lin1): Linear(in_features=768, out_features=3072,
bias=True)
            (lin2): Linear(in_features=3072, out_features=768,
bias=True)
            (activation): GELUActivation()
          )
          (output_layer_norm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
        )
        (4): TransformerBlock(
          (attention): MultiHeadSelfAttention(
            (dropout): Dropout(p=0.1, inplace=False)
            (q_lin): Linear(in_features=768, out_features=768,
bias=True)
            (k_lin): Linear(in_features=768, out_features=768,
bias=True)
            (v_lin): Linear(in_features=768, out_features=768,
bias=True)
            (out_lin): Linear(in_features=768, out_features=768,
bias=True)
          )
          (sa_layer_norm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
          (ffn): FFN(
            (dropout): Dropout(p=0.1, inplace=False)
            (lin1): Linear(in_features=768, out_features=3072,
bias=True)
            (lin2): Linear(in_features=3072, out_features=768,
```

```
bias=True)
            (activation): GELUActivation()
          )
          (output_layer_norm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
        )
        (5): TransformerBlock(
          (attention): MultiHeadSelfAttention(
            (dropout): Dropout(p=0.1, inplace=False)
            (q_lin): Linear(in_features=768, out_features=768,
bias=True)
            (k_lin): Linear(in_features=768, out_features=768,
bias=True)
            (v_lin): Linear(in_features=768, out_features=768,
bias=True)
            (out_lin): Linear(in_features=768, out_features=768,
bias=True)
          )
          (sa_layer_norm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
          (ffn): FFN(
            (dropout): Dropout(p=0.1, inplace=False)
            (lin1): Linear(in_features=768, out_features=3072,
bias=True)
            (lin2): Linear(in_features=3072, out_features=768,
bias=True)
            (activation): GELUActivation()
          )
          (output_layer_norm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
        )
      )
    )
  )
  (head): Sequential(
    (0): Linear(in_features=768, out_features=128, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.3, inplace=False)
    (3): Linear(in_features=128, out_features=2, bias=True)
  )
)
```

ДОДАТОК В

Реалізація трансформеру, рекурентної та згорткової мереж для класифікації часових рядів у PyTorch

Обидві мережі реалізовано аналогічно випадку класифікації тексту, за винятком вилучення ембедінг-матриці для вивчення векторної репрезентації слів у задачах з текстом.

Рекурентна мережа складається з декількох шарів LSTM-клітин та класифікаційного шару:

```
class LSTMNet(torch.nn.Module):
    def __init__(
        self,
        input_size,
        hidden_dim,
        out_dim,
        num_lstm_layers=2,
    ):
        super().__init__()
        self.hidden_dim = hidden_dim
        self.num_lstm_layers = num_lstm_layers
        self.out_dim = out_dim
        self.lstm_batch_first = False
        self.lstm1 = nn.LSTM(
            input_size,
            hidden_dim,
            batch_first=self.lstm_batch_first,
            num_layers=self.num_lstm_layers,
        )
        self.clf = nn.Linear(hidden_dim, out_dim)

    def forward(self, features):
        lstm_out, (lstm_hidden, _) = self.lstm1(features)
```

```
        lstm_out = lstm_out[:, -1, :].squeeze()
        out = self.clf(lstm_out)
        return out
```

Згорткова мережа побудована з двох блоків наступного вмісту: згортковий шар, блокова нормалізація та нелінійна активація. За цими блоками слідують pooling шар та шар для передбачення.

```
class CNNNet(torch.nn.Module):
    def __init__(
        self,
        input_size: tuple,
        last_hidden_dim,
        out_dim,
    ):
        super().__init__()
        self.last_hidden_dim = last_hidden_dim
        self.out_dim = out_dim
        self.in_channels, self.seq_length = input_size
        self.avgp_kernel_size = 8
        self.hidden_dim_before_clf = (
            self.seq_length // self.avgp_kernel_size
        ) * self.last_hidden_dim

        self.conv1 = nn.Sequential(
            nn.Conv1d(
                in_channels=self.in_channels,
                out_channels=64,
                kernel_size=3,
                padding="same",
            ),
            nn.BatchNorm1d(num_features=64),
            nn.ReLU(),
        )
        self.conv2 = nn.Sequential(
            nn.Conv1d(
                in_channels=64,
                out_channels=self.last_hidden_dim,
                kernel_size=4,
                padding="same",
            ),
            nn.BatchNorm1d(num_features=self.last_hidden_dim),
            nn.ReLU(),
        )
        self.avgp =
```

```
nn.AvgPool1d(kernel_size=self.avgp_kernel_size)

        self.clf = nn.Linear(self.hidden_dim_before_clf, out_dim)

    def forward(self, features):
        x = self.conv1(features)
        x = self.conv2(x)
        x = self.avgp(x)
        x = x.view(-1, self.hidden_dim_before_clf)
        out = self.clf(x)
        return out
```

Мережу трансформер реалізовано з використанням torch.nn.TransformerEncoderLayer шару для виокремлення ознак та звичайної повнозв'язної мережі для вирішення задачі класифікації (згідно рекомендацій у [43]):

```
class Transformer(nn.Module):
    def __init__(
        self, input_size, out_dim, nhead=4, num_layers=1,
dropout=0.1, *args, **kwargs
    ):
        super().__init__()
        self.in_channels, self.seq_length = input_size
        self.model_type = "Transformer"

        self.src_mask = None
        self.pos_encoder = PositionalEncoding(self.seq_length)
        self.encoder_layer = nn.TransformerEncoderLayer(
            d_model=self.seq_length, nhead=nhead, dropout=dropout
        )
        self.transformer_encoder = nn.TransformerEncoder(
            self.encoder_layer, num_layers=num_layers
        )
        self.decoder = nn.Sequential(
            nn.Linear(self.seq_length, self.seq_length),
            nn.ReLU(),
            nn.Linear(self.seq_length, self.seq_length),
            nn.ReLU(),
            nn.Linear(self.seq_length, out_dim),
        )

    def forward(self, src):
        if self.src_mask is None or self.src_mask.size(0) !=
len(src):
            device = src.device
            mask =
```

```
self._generate_square_subsequent_mask(len(src)).to(device)
            self.src_mask = mask

        src = self.pos_encoder(src)
        output = self.transformer_encoder(src,
self.src_mask)
        output = self.decoder(output)
        output = torch.squeeze(output, dim=1)
        return output

    def _generate_square_subsequent_mask(self, sz):
        mask = (torch.triu(torch.ones(sz, sz)) ==
1).transpose(0, 1)
        mask = (
            mask.float()
            .masked_fill(mask == 0, float("-inf"))
            .masked_fill(mask == 1, float(0.0))
        )
        return mask
```