

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування інституту, назва факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА РЕАЛІЗАЦІЯ ПРОГРАМНОЇ ТА АПАРАТНОЇ
СКЛАДОВИХ КЛАСИФІКАТОРА ТЕКСТА НА ОСНОВІ НЕЙРОННОЇ МЕРЕЖІ
RESEARCH AND IMPLEMENTATION OF SOFTWARE AND HARDWARE
COMPONENTS OF NEURAL NETWORK-BASED TEXT CLASSIFIER

Виконав: студент 2 курсу, групи КНТ-111м

Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

Діденко А.Є.

(прізвище та ініціали)

Керівник Табунщик Г.В.

(прізвище та ініціали)

Рецензент Шитікова О.В.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Інститут, факультет ФКНТ
 Кафедра програмних засобів
 Ступінь вищої освіти магістр
 Спеціальність 121 Інженерія програмного забезпечення
 (код і найменування)

Освітня програма (спеціалізація) Інженерія програмного забезпечення
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
 “ ” 20 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

Діденка Артема Євгеновича
 (прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та реалізація програмної та апаратної
складових класифікатора тексту на основі нейронної мережі. Research and
Implementation of Software and Hardware Components of Neural Network-Based Text
Classifier

керівник проєкту (роботи) Табунщик Галина Володимирівна, к.т.н., професор,
 (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “04” листопада 2022 року № 363

2. Строк подання студентом проєкту (роботи) 17 грудня 2022 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне
завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно
 розробити) 1. Аналіз предметної області. 2. Огляд і аналіз методів та засобів
класифікації текстів. 3. Розробка програмних засобів класифікації текстів.

4. Розробка апаратних засобів прискорення класифікації текстів. 5. Тестування
розробленої системи. 6. Керівництво програміста. Висновки. Перелік джерел
посилання. Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	Табунщик Г.В, професор		
Нормоконтролер	Камінська Ж.К., асистент		

7. Дата видачі завдання “05” вересня 2022 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області	2-5 тижні	Розділ 1
3	Розробка архітектури системи	6-8 тиждень	Розділ 2
4	Розробка системи	9-11 тижні	Розділ 3
5	Тестування та експериментальне дослідження системи	12 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	13 тиждень	Додатки
7	Захист роботи	14 тиждень	

Студент

_____ Діденко А.Є.
(підпис) (прізвище та ініціали)

Керівник проєкту (роботи)

_____ Табунщик Г.В.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
102 с., 44 рис., 8 табл., 3 дод., 55 джерел.

АПАРАТНИЙ ПРИСКОРЮВАЧ, ГЛИБИННЕ НАВЧАННЯ,
КЛАСИФІКАЦІЯ ТЕКСТІВ, ШТУЧНА НЕЙРОННА МЕРЕЖА.

Об'єкт дослідження – програмні та апаратні засоби класифікації текстів.

Предмет дослідження – методи програмно-апаратної класифікації текстів на основі нейронних мереж.

Мета роботи – прискорення процесу класифікації текстів з використанням нейронних мереж.

Матеріали, методи та технічні засоби: хмарний сервіс AWS з використанням апаратних засобів екземпляру EC2 F1, мови програмування Python та OpenCL, фреймворк глибинного навчання PyTorch.

Результати. Досліджено методи програмної та апаратної класифікації текстів, розроблено та навчено нейронну мережу для задачі класифікації текстів, розроблено апаратний прискорювач роботи нейронної мережі, прискорено процес класифікації текстів з використанням нейронних мереж.

Висновки. Створено програмну та апаратну складові класифікатора текстів на основі нейронної мережі.

Галузь використання – класифікація статей в наукометричних базах, класифікація книг в бібліотечних базах.

ABSTRACT

Explanatory note to the master's final qualifying work: 102 pages, 44 figures, 8 tables, 3 appendixes, 55 sources.

ARTIFICIAL NEURAL NETWORK, DEEP LEARNING, HARDWARE ACCELERATOR, TEXT CLASSIFICATION.

Object of study is software and hardware tools of text classification.

Subject of study is methods of software and hardware text classification based on neural networks.

The purpose of the work is acceleration of the process of text classification using neural networks. Materials, methods and tools AWS cloud service using EC2 F1 instance hardware, Python and OpenCL programming languages, PyTorch deep learning framework.

Results. The methods of software and hardware text classification are investigated, a neural network for the task of text classification is developed and trained, a hardware accelerator for the neural network is developed, the process of text classification using neural networks is accelerated.

Conclusions. Created software and hardware components of the text classifier based on a neural network.

The field of use is classification of articles in scientometric databases, classification of books in library databases.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	8
Вступ.....	9
1 Аналіз предметної області.....	10
1.1 Обробка природної мови.....	10
1.2 Класифікація текстів.....	19
1.3 Штучні нейронні мережі.....	25
1.4 Нейромережеві методи класифікації текстів.....	31
1.5 Використання FPGA у задачах апаратного прискорення.....	36
1.7 Висновки за розділом 1.....	44
2 Огляд і аналіз методів та засобів класифікації текстів.....	45
2.1 Вибір методу розробки на FPGA.....	45
2.2 Вибір програмного і апаратного середовища.....	46
2.3 Вибір мови програмування для створення нейромережі.....	50
2.4 Вибір фреймворку глибинного навчання.....	53
2.5 Висновки за розділом 2.....	56
3 Розробка програмних засобів класифікації текстів.....	57
3.1 Розробка нейромережі.....	57
3.2 Навчання нейромережі.....	62
3.3 Висновки за розділом 3.....	63
4 Розробка апаратних засобів прискорення класифікації текстів.....	64
4.1 Розробка ядер для FPGA.....	64
4.2 Розробка хост-програми.....	65
4.3 Висновки за розділом 4.....	67
5 Тестування розробленої системи.....	68
5.1 Тестування та перевірка результатів.....	68
5.2 Висновки за розділом 5.....	69
6 Керівництво програміста.....	70

6.1 Вимоги до технічних та програмних засобів	70
6.2 Налаштування робочого середовища.....	70
6.3 Висновки за розділом 6	71
Висновки	72
Перелік джерел посилання	73
Додаток А Технічне завдання	79
Додаток Б Текст програми	84
Додаток В Слайди презентації.....	93

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

- FPGA – field programmable gate array;
ГПМ – генерація природної мови;
ЗНМ – згорткова нейронна мережа;
ОПМ – обробка природної мови;
РНМ – рекурентна нейронна мережа;
ШНМ – штучна нейронна мережа.

ВСТУП

Задача класифікації текстів виникає при роботі з неструктурованими об'ємами даних, де текстову інформацію необхідно співвіднести до категорій, чи то певна тема статі з наукової бази даних, чи то емоційне забарвлення відгуку в коментарях. Класифікація текстів є важливою складовою інформаційного пошуку і дозволяє виявляти необхідну інформацію у неструктурованих даних.

Методи глибинного навчання, такі як нейронні мережі, використовуються у задачах класифікації текстів і показують свою ефективність у порівнянні зі звичайними методами машинного навчання.

Враховуючи ріст об'єму текстової інформації у світі, виникає необхідність у розробці та покращенні існуючих методів класифікації текстів з метою зменшення часу обробки та супровідних затрат, таких як обчислювальні ресурси та енергоспоживання. З цією метою вдаються не тільки до оптимізації програм обробки природної мови, а й до вдосконалень та використання нових обчислювальних пристроїв, таких як апаратні прискорювачі. Існуючі на даний момент мікросхеми програмованої логіки, як наприклад FPGA, знаходять широке використання саме у таких задачах апаратного прискорення операцій.

Актуальність роботи полягає у дослідженні та розробці програмних і апаратних засобів побудови та прискорення нейромережових класифікаторів текстів з використанням нейронних мереж.

У випускній кваліфікаційній роботі розроблюється програма нейромережового класифікатора текстів, створюються і навчаються штучні нейронні мережі з різною конфігурацією, порівнюються їх характеристики та результати роботи, описуються можливості для подальшого покращення отриманих результатів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Обробка природної мови

1.1.1 Загальне визначення

Обробка природної мови (ОПМ) – це сфера досліджень у комп'ютерних науках та штучному інтелекті, що пов'язана з обробкою різноманітних мов, якими спілкуються люди, таких як українська або англійська. Процес обробки в основному включає в себе трансляцію природної мови у дані (числа), які може використовувати комп'ютер для вивчення навколишнього світу. Таке розуміння навколишнього світу зазвичай використовується для генерації тексту, написаного природною мовою, що відображає розуміння цього світу [1].

Оскільки ця область досліджень та розробок швидко розвивається, то не існує одного узгодженого визначення. Деякі аспекти ОПМ можна дати у наступному визначенні: ОПМ – це набір теоретичних обчислювальних технік для аналізу та представлення природних текстів на одному або більше рівнях лінгвістичного аналізу з метою досягнення людиноподібної обробки мови для ряду задач та можливих застосувань [2].

Декілька уточнень мають бути зроблені до попереднього визначення, а саме:

- існує безліч різноманітних методів та технік, що можуть бути використані для конкретного типу аналізу мови;
- природні тексти можуть бути представлені будь-якою мовою, жанром та видом. Єдина вимога полягає в тому, що вони мають бути написані мовою, якою спілкуються люди між собою;
- згадка рівнів лінгвістичного аналізу посиляється на той факт, що при відтворенні або сприйнятті мови людина використовує безліч видів обробки мови. Вважається, що люди зазвичай використовують всі ці рівні,

оскільки кожен рівень передає різні види тлумачення. Але різні системи ОПМ використовують різні рівні, або їх комбінації;

– людиноподібність у обробці мови розкриває той факт, що ця дисципліна прагне до людського рівня розуміння та відтворення мови [2].

Система природної мови має використовувати значні знання про структуру мови, включаючи те, що таке слова, як вони поєднуються у речення, що вони означають, який зміст привносять до речення і т.д. Однак, ми не можемо повністю покладатися на лінгвістику без урахування іншого аспекту, що робить людину розумною – загальних знань про світ та здібностей до міркувань. Так, наприклад, для того, щоб відповісти на питання або взяти участь у розмові, людина повинна мати не тільки достатньо про структуру мови, на якій розмовляє, але й про світ взагалі та зокрема про контекст розмови [3].

Двома основними напрямками ОПМ є розуміння та генерація природної мови (ГПМ). На рис. 1.1 зображено складові частини ОПМ [4].

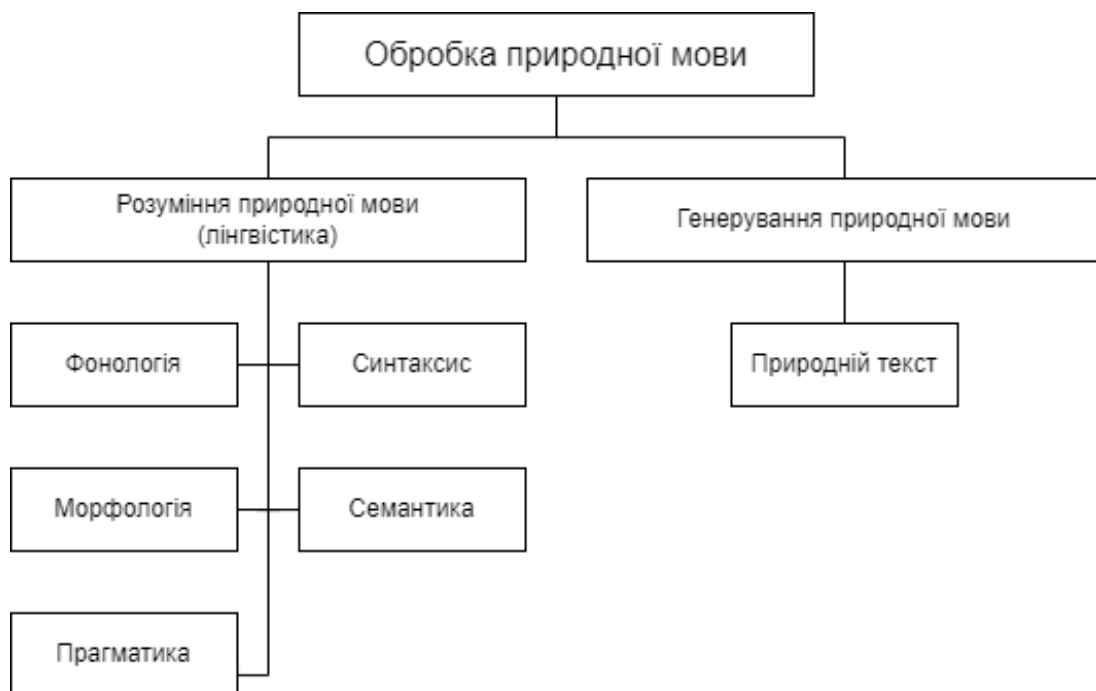


Рисунок 1.1 – Компоненти ОПМ [4]

1.1.2 Розуміння природної мови

У сфері розуміння природної мови досліджуються методи, що дозволяють комп'ютеру розуміти природну мову та аналізувати її, виокремлюючи концепції, сутності, емоції та ключові слова [4].

Дано визначення представленим у схемі на рис. 1.1 складовим, що необхідні для розуміння природної мови:

- фонологія є розділом лінгвістики, який вивчає набори звуків, це вчення про звуки, що властиві системі мови. Фонологія включає семантичне використання звуку для кодування значення будь-якої людської мови;

- морфологія вивчає те, як слова будуються з більш основних одиниць, які називаються морфемами [3]. Морфема — це примітивна одиниця мови, що несе в собі значення, як, наприклад, значення слова friendly виводиться зі значення іменника friend і суфікса -ly, який перетворює іменник на прикметник;

- синтаксис вивчає те, як слова можуть поєднуватись для формування правильного речення, визначає, яку роль відіграє кожне слово в структурі реченні та які фрази є частинами яких інших фраз;

- семантика вивчає значення слів і те, як ці значення поєднуються в реченнях, щоб сформувати значення речення. Семантика вивчає значення незалежно від контексту, тобто значення, яке має речення незалежно від контексту, у якому воно вживається;

- прагматика вивчає, як речення використовуються в різних ситуаціях і як це використання впливає на інтерпретацію речення [4].

Більшість з перелічених складових формують рівні ОПМ. Так, спочатку виконується лексичний рівень, на якому система інтерпретує значення окремих слів, та проводиться аналіз структури слів з точки зору їх лексичного значення і частин мови, до яких вони відносяться. На лексичному рівні застосовуються такі техніки очищення і вилучення інформації, як

видалення стоп-слів, стеммінг, лематизація, тощо. Після лексичного рівня, на синтаксичному рівні слова групуються у словосполучення, а словосполучення групуються у речення. Синтаксичний рівень досліджує порядок слів, стоп-слова, морфологію і вживання слів, що лексичний рівень не враховує. На семантичному рівні найважливішим завданням є визначення правильного значення речення. Щоб зрозуміти речення, люди покладаються на знання про мову і концепції, присутні в цьому реченні, але машини не можуть розраховувати на ці методи. Семантична обробка визначає можливі значення речення шляхом обробки його логічної структури для виявлення найбільш значущих слів, щоб зрозуміти взаємодію між словами або різними поняттями в реченні. У той час як синтаксичний і семантичний рівні мають справу з одиницями довжиною в речення, дискурсивний рівень має справу з більш ніж одним реченням. Він займається аналізом логічної структури шляхом встановлення зв'язків між словами та реченнями, які забезпечують її узгодженість, фокусується на властивостях тексту, які передають значення, шляхом інтерпретації зв'язків між реченнями і виявлення лінгвістичних структур з текстів на декількох рівнях. Прагматичний рівень фокусується на знаннях або змісті, які надходять ззовні змісту документа, тобто знання з реального світу. Фактично, на прагматичному рівні аналізуються речення, які безпосередньо не вимовляються. Коли речення не є конкретним і контекст не надає ніякої конкретної інформації про це речення, виникає так звана прагматична неоднозначність [4].

1.1.3 Генерація природної мови

ГПМ – це процес створення фраз, речень і абзаців, які є змістовними з точки зору розуміння людини. ГПМ відбувається в чотири етапи: визначення цілей, планування того, як цілі можуть бути досягнуті шляхом оцінки ситуації та доступних комунікативних джерел, і реалізація планів у вигляді тексту [4].

З технічної точки зору, майже всі прикладні системи ГПМ виконують наступні три завдання:

– визначення змісту полягає у прийнятті рішення, яка інформація повинна бути повідомлена користувачеві, а планування змісту – у тому, як ця інформація має бути структурована. Ці завдання зазвичай виконуються одночасно;

– планування речень полягає у вирішенні, як інформація буде розподілена між окремими реченнями та абзацами, і які засоби зв'язку (наприклад, займенники, маркери дискурсу) слід додати, щоб зробити текст плавним;

– реалізація полягає у граматично правильному формуванні окремих речень [5].

1.1.4 Загальна послідовність обробки природної мови

Підсумовуючи вище сказане, можна сформулювати послідовність, у якій виконується загальна задача ОПМ (рис. 1.2) [3].



Рисунок 1.2 – Послідовність виконання ОПМ [3]

ОПМ за своєю природою є сукупністю процесів перетворення, що ставлять у відповідність одному представленню інше. Наприклад, процес, який ставить у відповідність реченню його синтаксичну структуру та логічну форму, називається парсером, що використовує лексику, тобто знання про слова і їх значення, та граматику, тобто набір правил, що визначають правильні конструкції [3].

Процес контекстуальної інтерпретації перетворює синтаксичну структуру і логічну форму речення в його кінцеве значення. Цей процес включає такі питання, як аналіз об'єктів, що визначенні іменниками і займенниками, аналіз часових аспектів інформації, виявлення присутніх намірів, а також інтелектуальна обробка, необхідна для належної інтерпретації речення в межах предметної області. Для остаточного представлення речення у вигляді його значення використовуються також знання контексту дискурсу (визначеного реченнями, що передували поточному) і знання з предметної області [3].

Після цього обмірковує задачу у межах предметної області. Якщо користувачу необхідно надати відповідь, то значення відповіді передається компоненту генерації тексту системи. Він використовує знання контексту дискурсу, а також інформацію про граматику і лексику, щоб спланувати форму і структуру відповіді, яка потім буде відображена у вигляді слів в процесі реалізації [3].

Слід звернути увагу на те, що ті ж самі знання, такі як словниковий запас, граматика та ін., використовуються і під час генерації відповіді. Так наприклад, синтаксична структури відповіді створюються за допомогою тієї ж граматики. Граматика, що може бути використана або для ідентифікації структури речення, або для реалізації структури у вигляді послідовності слів, називається двонаправленою граматиною [3].

1.1.5 Лінгвістичні аналіз

Розглянемо так званий лінгвістичний аналіз та його засоби, що мають важливе значення у процесі ОПМ.

Лінгвістичний аналіз тексту, як правило, проводиться крок за кроком. Попередньо оброблений документ спочатку розбивається на абзаци, абзаци – на речення, а речення – на окремі слова. Згодом слова в реченні тежуються (tagged) частинами мови та іншими ознаками, перш ніж саме речення піддається парсингу (тобто піддано граматичному аналізу). Таким чином, парсери, як правило, базуються на таких засобах, як розділювачі (sentence delimiters), токенизатори (tokenizers), стеммери (stemmers) і теггери (taggers) частин мови (POS). Але не всі програми потребують повного набору таких інструментів. Наприклад, всі пошукові системи виконують крок токенизації, але не всі виконують тежування частинами мови [6].

Попередня обробка тексту (text preprocessing) є завданням з перетворення первинного текстового файлу, по суті послідовності бітів, у чітко визначену послідовність лінгвістичних одиниць: символів, що представляють окремі графеми, слів, що складаються з одного або декількох символів, і речень, що складаються з одного або декількох слів. Попередня обробка тексту є важливою частиною будь-якої системи з ОПМ, оскільки визначені на цьому етапі символи, слова і речення є фундаментальними одиницями, що передаються на всі подальші етапи процесу обробки [8]. Так, наприклад, для попередньої обробки текстів для задач на Field Programmable Gage Array (FPGA) існують спеціалізовані програмні методи, що описані у роботі [7].

Попередню обробку тексту можна розділити на два етапи: відбір документів (document triage) і сегментацію тексту (text segmentation). Відбір документів – це процес перетворення набору файлів у чітко визначені текстові документи. Процес відбору відбувається у декілька етапів. Спочатку визначається тип кодування символів, потім визначається мова документу, і,

нарешті, відбувається секціонування тексту, при якому виділяється фактичний зміст файлу без урахування таких елементів, як зображення, таблиці, посилання, HTML-теги. Результатом етапу відбору документів є чітко визначений текстовий корпус, організований за мовою і придатний для сегментації тексту та подальшого аналізу. Сегментація тексту, у свою чергу, – це процес перетворення чітко визначеного текстового масиву на складові його слова та речення [8].

Сегментація речень (sentence segmentation) – процес, що необхідний для розділення тексту на речення для подальшої обробки. Він полягає у визначенні одиниць обробки, що складаються з одного або декількох слів. Це завдання передбачає визначення меж між словами в різних реченнях [8]. Точне визначення меж речень є непростим завданням, оскільки розділові знаки, що позначають кінець речення, часто бувають неоднозначними. Наприклад, крапка може позначати десяткову крапку, аббревіатуру, кінець речення або аббревіатуру в кінці речення. Аналогічно, речення починаються з великої літери, але не всі слова з великої літери починають речення, навіть якщо вони стоять після крапки. Щоб розрізнити розділові знаки, розділювачі речень часто вдаються до використання регулярних виразів, виняткових правил, емпіричних методів, тощо [6].

Процес сегментації речень іноді потребує допомоги токенизаторів (також відомих як лексичні аналізатори та сегментатори слів) для розпізнавання розділових знаків [6]. Токенізація означає поділ вхідного тексту, який для комп'ютера є лише одним довгим рядком символів, на окремі одиниці, які називаються токенами. Ці токени потім подаються на наступні етапи ОПМ, такі як морфологічний аналіз, тегування частинами мови і парсинг. Для того, щоб виконати токенизацію, необхідно описати, які набори символів вважати токеном і які виконувати дії (наприклад, розривати їх або з'єднувати їх разом) при кожній їх появі у вхідному файлі. Шаблони наборів, які зазвичай потрібні для розпізнавання токенів, відносяться до класу формальних мов, тобто до регулярних виразів. Необхідно також

визначитися з типом роздільників токенів: чи це буде прогалина, новий рядок або інше [9].

Стеммери є морфологічними аналізаторами, які пов'язують варіанти одного і того ж терму з його кореневою формою. Корінь можна розглядати як форму слова, яку зазвичай можна знайти в словнику [6]. Один з поширених підходів полягає у видаленні афіксів з вхідних слів, таким чином залишаючи їх основу. Якщо видалити афікс належним чином, то всі варіанти форм слова будуть перетворені в одну стандартну форму. Оскільки процес спрямований на перетворення вхідного тексту для задач інформаційного пошуку, основа слова не обов'язково повинна бути лінгвістично правильною лемою або коренем. Прийнято вважати, що видалення закінчень слів (іноді його називають суфіксацією) є прийнятним способом стеммінгу. Видалення префіксі у свою чергу може бути корисним у деяких предметних областях, таких як хімія, хоча широкого застосування воно не набуло [10].

Існує два типи морфологічних аналізаторів для стеммінгу: флективний (inflectional), у якому присутні синтаксичні зв'язки між словами однієї частини мови, і дериваційний (derivational), у якому присутні лексичні зв'язки між словами, що можуть належати до різних частин мови [6].

Теги – це описові символи, які присвоюються (простим або складеним) словам у тексті. Часто ці теги називають тегами частинам мови, оскільки вони, як правило, вказують на частину мови (іменник, дієслово, прийменник тощо), до якої належить слово. Тегований текст корисний для багатьох цілей, наприклад, для досліджень текстових корпусів, програм обробки мовленн, автоматичного розпізнавання понять, перекладу, тощо. У випадку ОПМ, тегування необхідне для подальшого парсингу речень [9].

1.2 Класифікація текстів

1.2.1 Постановка задачі класифікації текстів

Класифікація текстів, як і будь-яка задача класифікації відноситься до способів навчання з учителем. Навчання з учителем має на увазі наявність як вхідних так і вихідних даних. У випадку класифікації текстів вхідними даними є вибірки тексту, а вихідними – категорії, до яких необхідно віднести ту чи іншу вибірку тексту.

Задачу класифікації можна формалізувати наступним чином. Нехай у нас є множина навчальних записів $D=\{X_1, X_2, \dots, X_N\}$, така, що кожному запису ставиться у відповідність значення класу з множини k різних дискретних значень, індексованих як $\{1 \dots k_0\}$. Навчальні дані використовуються для побудови моделі класифікації, яка пов'язує ознаки в певному записі з однією з міток класу. Для даного запису, для якого клас невідомий, навчальна модель використовується для прогнозування мітки класу. У жорсткій версії задачі класифікації певна мітка присвоюється запису у явному вигляді, тоді як у послабленій версії задачі класифікації запису присвоюється значення ймовірності тієї чи іншої мітки класу. Інші варіації задачі класифікації дозволяють ранжирувати різні варіанти міток класів для певного запису або дозволяють присвоювати декілька міток [11].

Процес класифікації тестів складається з декількох стадій, що представлені графічно у вигляді схеми на рис. 1.3 [11].

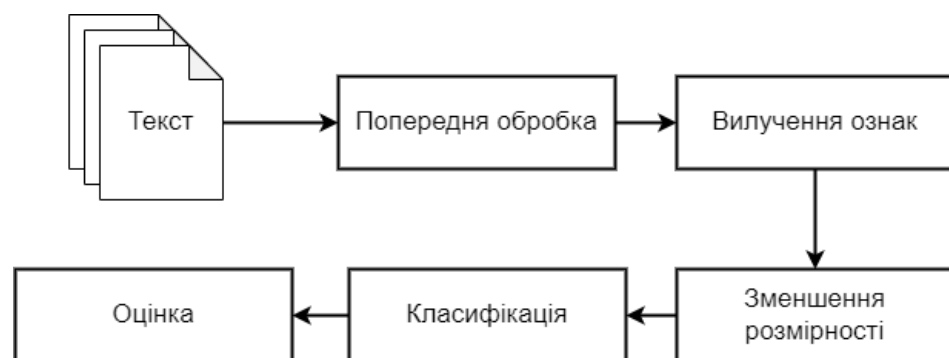


Рисунок 1.3 – Стадії процесу класифікації текстів [11]

Стадія попередньої обробки тексту була розглянута у розділі 1.1. При попередній обробці тексту використовуються вже згадані засоби стеммінгу, лематизації, тощо. Наступним етапом є вилучення ознак, що має на меті вибір тих слів, за якими у майбутньому буде проводитися класифікація. При вилученні ознак словам ставляться у відповідність значення частоти, тощо. Таким чином, до документу, що представлений у математичному вигляді, можуть бути застосовані методи комп'ютерного аналізу. Оскільки процес вилучення ознак може згенерувати великий масив даних, обробка якого може бути трудомісткою, застосовуються так звані методи зменшення розмірності, що залишають тільки необхідні ознаки, зменшуючи таким чином простір ознак та спрощуючи процес обробки класифікатором. Власне процес класифікації ставить кожному тексту у відповідність мітку категорії, до якої він належить. В кінці відбувається оцінка отриманих результатів та роботи класифікатора. Розглянемо детальніше стадії процесу класифікації та можливі варіанти класифікаторів.

1.2.2 Вилучення ознак

Одним з найпоширеніших методів вилучення ознак з тексту і його математичного представлення придатного для подальшої обробки є так званих метод торби слів (bag-of-words). Метод торби слів полягає у створенні вектора-стовпця x , де окрема координата x_j відповідає кількості разів, скільки слово j зустрічається у тексті. Метод торби слів включає тільки інформацію про кількість кожного слова, а не про порядок, в якому слова з'являються. При представленні слів у вигляді торби ігнорується граматики, межі речень, абзаци [13].

Основним недоліком методу торби слів є втрата інформації, оскільки кожне слово аналізується окремо. N -грами допомагають певною мірою вирішити цю проблему. N -грама – це нерозривна послідовність з N елементів тексту. N -грами можуть бути як словами, так і, в залежності від випадку,

буквами, складами, а іноді і фонемами. Біграма – це N -грама, що складається з двох елементів і у випадку роботи зі словами є словосполученням. Один із способів знаходження біграми в тексті є обчислення умовної ймовірності наявності слова, при умові наявності іншого слова. Таким чином знаходяться слова, що найчастіше зустрічаються парами у тексті [14].

У попередніх двох методах словам та словосполученням ставляться у відповідність числа, що позначають кількість разів, яку слово чи словосполучення зустрічаються у тексті. Такий простий метод зважування не дозволяє розрізнити ступінь важливості різних термів. Інтуїтивно зрозуміло, що терм, який частіше з'являється в документі, більшою мірою представляє тему документу. З іншого боку, терм, що зустрічається у меншій кількості документів в колекції, має більшу дискримінаційну силу, тобто за допомогою нього можна точніше віднести тексти до певної категорії. Ґрунтуючись на цьому представленні, більшість моделей векторного простору використовують комбінацію двох величин: частоти терма та інвертованої частоти документа з даним термом. Частота терма (TF) представляє собою кількість входжень терма в певному документі; а інвертована частота документа (IDF) вимірює, наскільки рідко терм з'являється у всіх документах [15].

1.2.3 Зменшення розмірності

Відбір ознак має на меті видалення з набору ознак нерелевантні або невідповідні. Першою причиною для зменшення розмірності при класифікації текстів є попередження перенавчання. Другою і головною причиною є підвищення ефективності обчислень, оскільки багато алгоритмів навчальних моделей не можуть обробляти багатовимірні простори ознак [16].

Найпростішим прикладом методу зменшення розмірності є видалення стоп-слів, тобто таких слів, які не несуть важливої інформації та зустрічаються у всіх документах корпусу. Прикладами таких слів можуть бути сполучники, займенники, артиклі, тощо. Іншим простим методом

зменшення розмірності є видалення слів, значення частот яких не перевищують заданого значення. Таким чином, слова, що рідко зустрічаються у тексті не беруть участі у процесі класифікації.

Одним з найпоширеніших методів зменшення розмірності є використання коефіцієнту (також індексу) Джині. За допомогою коефіцієнту Джині можна обчислити дискримінаційну силу терму. Індекс Джині лежить у інтервалі $[1, 1/k]$, де позначає кількість класів. Чим більше значення індексу Джині, тим більша дискримінаційна сила терму. Так, наприклад, якщо всі документи, що містять певний терм, належать до однієї категорії, то індекс Джині буде дорівнювати одиниці. З іншого боку, якщо документи, що містять даний терм, рівномірно розподілені між всіма класами, то індекс Джині буде дорівнювати $1/k$ [11].

До інших методів зменшення розмірності також відносяться взаємна інформація (mutual information), приріст інформації (information gain) та статистика χ^2 -квадрат.

Слід також відзначити, що окрім методів зменшення розмірності, які видаляють нерелевантні для класифікації слова, є методи, що створюють новий менший простір ознак з початкового. До таких методів відносяться латентно-семантичне індексування, лінійний дискримінантний аналіз та кластерний аналіз.

1.2.4 Методи класифікації

Основними методами класифікації текстів є наївний баєсів класифікатор, метод k -найближчих сусідів, метод опорних векторів, дерево класифікації.

Ідея наївного баєсового класифікатору полягає у використанні ймовірнісної моделі тексту, щоб обчислити вірогідність $P(y|d)$ того, що документ d знаходиться у класі y [16].

Наївний баєсів класифікатор одним з найпростішим і водночас найпоширенішим генеративним класифікатором. Він моделює розподіл документів у кожному класі за допомогою імовірнісної моделі з припущеннями про незалежний розподіл різних термів. Існує два класи моделей наївного баєсового класифікатору: багатовимірний метод Бернуллі (Multivariate Bernoulli Mode) та багатofакторна модель (Multinomial Model). Обидві моделі по суті обчислюють апостеріорну ймовірність класу на основі розподілу слів у документі. Ці моделі ігнорують фактичне розташування слів у документі і працюють з торбою слів. Основна відмінність між цими двома моделями полягає в припущенні щодо врахування (або неврахування) частот слів, а також у відповідному підході до вибірки ймовірнісного простору [11].

Наївний баєсів метод достатньо простий у використанні: ваги можуть бути використані у замкненому вигляді, а ймовірнісна інтерпретація відносно легко може бути розширеною. Однак припущення, що терми є незалежними, може серйозно обмежити точність методу. Представлення тексту у вигляді торби слів порушує припущення про умовну незалежність. Так, наприклад, ймовірність того, що документ міститиме слово «наївний», безумовно, вища, якщо він також містить слово «Баєс». Наївний байєсів класифікатор завжди занижує істинні ймовірності сполучень з позитивно корельованих термів. Щоб використовувати такі терми потрібні алгоритми навчання, які не спираються на припущення про незалежність [13].

Основним принципом роботи SVM є визначення роздільників, або гіперплощин, у просторі пошуку, які можуть найкраще розділити різні класи. На рис. 1.4 [11], зображено два класи, елементи яких позначені "x" та "o" відповідно.

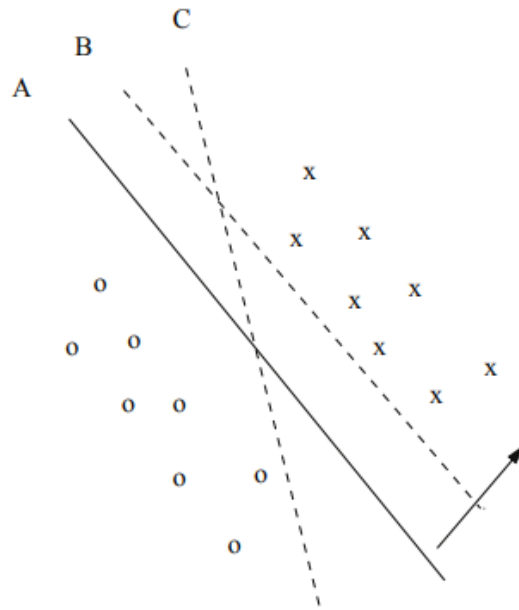


Рисунок 1.4 – Розділення класів гіперплощиною [11]

Три різні гіперплощини, які позначені A , B і C відповідно, розділяють елементи на два класи. Очевидно, що гіперплощина A забезпечує найкраще розділення між різними класами, оскільки відстань будь-якої з точок даних від неї є найбільшою. Таким чином, гіперплощина A забезпечує максимальну межу розподілення. Зауважимо, що вектор нормалі до цієї гіперплощини (зображений стрілкою на рисунку) є напрямком у просторі ознак, вздовж якого ми маємо максимальну дискримінацію. Однією з переваг методу SVM є його стійкість до високої розмірності, оскільки він намагається визначити оптимальний напрямок дискримінації в просторі ознак шляхом дослідження відповідної комбінації ознак. На практиці, метод опорних векторів використовується через його простоту та простоту в інтерпретації [11].

Метод k -найближчих сусідів представляє собою непараметричний метод класифікації. Отримавши тестовий документ, даний метод знаходить k найближчих сусідів даного документу серед усіх документів у навчальній вибірці. Недоліками даного методу є:

– висока складність обчислень: для знаходження k найближчих сусідів необхідно обчислити всі подібності між сусідами. При меншій кількості сусідів метод найближчих сусідів перестає бути оптимальним;

– залежність від навчальної вибірки: класифікатор генерується тільки на основі навчальних даних і не використовує ніяких додаткових даних. Це робить алгоритм залежним від навчальної колекції документів, тому він потребує перерахунку навіть при незначних змінах у навчальній вибірці;

– відсутність різниці між вибірками навчальних даних: всі навчальні вибірки розглядаються однаково, тобто немає різниці між вибірками з невеликою кількістю та великою кількістю даних [17].

Класифікація на основі дерева рішень – це індуктивний підхід до навчання, який виконує розбиття простору документів за ієрархією. У класифікаторі дерева рішень є як внутрішні, так і кінцеві вузли. Внутрішні вузли містять правила, які містять питання стосовно документу і розміщують його в тому чи іншому піддереві, що відповідає результату відповіді на питання. Документ, що проходить по дереву зверху вниз, врешті-решт потрапляє у кінцевий вузол, який представляє той чи інший клас. Таким чином документ класифікується. Рішеннями у внутрішніх вузлах можуть бути наявність або відсутність термів у документі. Для того, щоб уникнути перенавчання, зазвичай виконується відсічення вузлів [18].

1.3 Штучні нейронні мережі

1.3.1 Механізм роботи штучних нейронних мереж

Штучні нейронні мережі (ШНМ) є поширеним методом машинного навчання, що імітує механізм навчання, присутній живим організмам. Нервова система людини містить клітини, які називаються нейронами. Нейрони з'єднуються один з одним за допомогою аксонів і дендритів, а з'єднувальні ділянки між аксонами і дендритами називаються синапсами.

Сила синаптичних зв'язків часто змінюється у відповідь на зовнішні подразники. Ця зміна і є процесом навчання, що відбувається в живих організмах. Даний біологічний механізм навчання симулюється за допомогою ШНМ, що складаються з безлічі обчислювальних елементів, що також називаються нейронами. Обчислювальні елементи зв'язані між собою вагами, що виконують ту ж саму роль, що і сила синаптичних зв'язків у живих організмах. Кожен вхід до нейрону є зваженим, що впливає на функцію, що вираховується у даному обчислювальному елементі [21].

ШНМ обчислює функцію від входів, передаючи розраховані значення від вхідних до вихідних нейронів та використовуючи ваги нейронів у якості проміжних параметрів. Навчання відбувається за допомогою зміни значень ваг зв'язків між нейронами. Подібно до того, як для навчання біологічних організмів необхідні зовнішні стимули, так і в ШНМ зовнішнім стимулом є навчальні дані з навчальної вибірки, що містять приклади пар вхід-вихід функції, яка підлягає навчанню. Наприклад, навчальні дані можуть містити піксели, що представляють зображення (вхід) та їх категорії (наприклад, банан, морква) як виходи. Ці пари навчальних даних подаються до нейронної мережі, і за вхідною картинкою проводиться прогнозування відповідної категорії. Навчальні дані надають зворотній зв'язок про коректність ваг нейронної мережі в залежності від того, яким чином прогнозоване значення виходу (наприклад, вірогідність певної категорії) для певного входу відповідає заявленому значенню виходу у парі вхід-вихід навчальних даних. Можна розглядати помилки, зроблені нейронною мережею при розрахунку функції, як негативний зворотній зв'язок, що спонукає до коригування ваг. Мета зміни ваг полягає у модифікації розрахованої функції для того, щоб отримати більш коректні прогнози у майбутніх ітераціях. Тобто, ваги змінюються у математично-обґрунтованому способі так, щоб зменшити похибку обчислень на даному прикладі. З допомогою послідовної підгонки значень ваг між нейронами на багатьох парах вхід-вихід, функція, що обчислюється нейронною мережею, покращуються з плином часу таким

чином, що може надавати більш коректні передбачення. Така властивість до коректного обчислення функції невідомих входів за допомогою навчання на скінченній вибірці пар вхід-вихід, називається узагальненням моделі. Основна корисність всіх моделей машинного навчання полягає в їх здатності узагальнювати своє навчання від видимих навчальних даних до невидимих прикладів [21].

1.3.2 Загальні поняття та визначення

Розглянемо детальніше термінологію та визначення, які будуть зустрічатися надалі у ході роботи. Категорії, до яких належать основні терміни ШНМ наступні:

- обчислювальний елемент: можна розглядати ШНМ як максимально спрощену модель структури біологічної нейронної мережі. ШНМ складається з зв'язаних між собою обчислювальних елементів. Узагальнена модель обчислювального елемента складається з суматора та з частини обчислення значення виходу. Суматор отримує на вхід N вхідних значень, зважує кожне та обчислює зважену суму. Зважена сума називається значенням активації. Частина обчислення виходу виробляє сигнал зі значення активації. Знак ваги кожного входу визначає, чи є вхід збуджуючим (додатна вага), чи гальмівним (від'ємна вага). Входи і виходи можуть бути як дискретним та і неперервним;

- зв'язки: для того, щоб виконати задачу розпізнавання образу, у ШНМ декілька обчислювальних елементів зв'язані у відповідності до певної топології. Таким чином, значення до входів обчислювального елемента можуть надходити з виходів інших обчислювальних елементів та з зовнішнього середовища. Вихід кожного обчислювального елемента може бути поданий на декілька інших обчислювальних елементів або на вхід того ж обчислювального елемента. Кількість вихідного сигналу одного елемента,

що приймається іншим, залежить від сили зв'язку між елементами, та визначається значенням ваги, асоційованим з цим зв'язком. Якщо у даній ШНМ є N обчислювальних елементів, тоді у будь-який момент часу кожен елемент буде мати унікальне значення активації і унікальне значення виходу. Множина N значень активації визначає так званий стан активації мережі у даний момент часу. Таким же чином, множина N значень виходу мережі визначає вихідний стан мережі у даний момент часу. В залежності від того, дискретними чи неперервними є значення активації та виходу, стан мережі може бути охарактеризований як дискретна або неперервна точка у N -мірному просторі;

– виконання операцій: під час роботи, кожен обчислювальний елемент отримує входи від інших елементів та/або від зовнішнього джерела. У заданий момент часу обчислюється зважена сума входів. Значення активації задає значення виходу блоку вихідної функції, тобто вихідний стан обчислювального елемента. Значення виходу та інших зовнішніх входів, в свою чергу, визначають стани активації виходів інших обчислювальних елементів. Динаміка активації є функцією стану значення від часу, тобто визначає значення активації всіх елементів у певний момент часу. Динаміка активації визначає також динаміку вихідного стану мережі. Динаміка активації визначає траєкторію шляху станів у просторі станів мережі. Для заданої нейронної мережі стани активації визначають функцію короткочасної пам'яті мережі. Зазвичай, за заданим зовнішнім входом, динаміка активації використовується для відтворення патерну, збереженого у мережі. Для того, щоб зберігти патерн у мережі, необхідно налаштувати ваги зв'язків. Сукупність всіх ваг в мережі утворюють вектор ваг. Сукупність всіх можливих векторів ваг визначають простір ваг. Коли ваги змінюються, то синаптична динаміка мережі визначає вектор ваг як функцію від часу. Синаптична динаміка використовується для налаштування вагових коефіцієнтів з метою збереження патернів в мережі. Процес налаштування вагових коефіцієнтів називається навчанням. Після завершення процесу

навчання остаточний набір значень вагових коефіцієнтів відповідає функції довгострокової пам'яті мережі. Процедура поетапного оновлення кожного з вагових коефіцієнтів називається законом навчання або алгоритмом навчання;

– оновлення: існує декілька варіантів як для динаміки активації, так і для синаптичної динаміки. Зокрема, оновлення вихідних станів всіх обчислювальних елементів може виконуватися як синхронно так і асинхронно. У цьому випадку значення активації всіх елементів обчислюються одночасно, коли при асинхронному оновленні кожен елемент оновлюється послідовно, враховуючи кожен раз поточний вихідний стан мережі [19].

1.3.3 Модель штучного нейрона

Структура штучного нейрона, що виконує функції обчислювального елементу у ШНМ, наведена на рис. 1.3. Окрім вже представлених вище входів, ваг, суматора, функції активації та виходу, у штучного нейрона наявний також так званий зсув або також порогове значення. Порогове значення впливає на збільшення або зменшення результуючого входу функції активації в залежності від того, чи є воно додатним або від'ємним відповідно [20].

У математичному вигляді модель штучного нейрону, структура якого відповідає зображеній на рис. 1.5 [20], описується формулами 1.1 та 1.2.

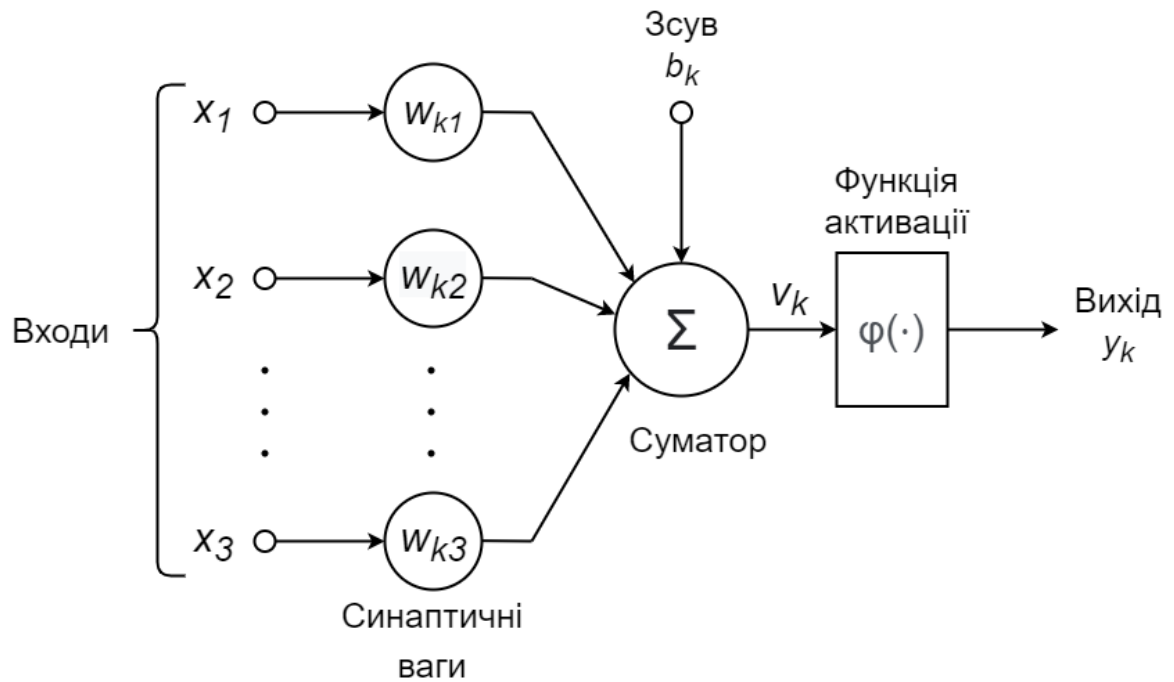


Рисунок 1.5 – Структура штучного нейрона [20]

$$u_k = \sum_{j=1}^m w_{kj} x_j \quad (1.1)$$

$$y_k = \varphi(u_k + b_k), \quad (1.2)$$

де x_j – вхідний сигнал нейрону k ;

w_j – відповідні синаптичні ваги нейрону k ;

u_k – лінійна комбінація входів нейрону k ;

b_k – порогове значення, або зсув нейрону k ;

y_k – вихідний сигнал нейрону k ;

φ – функція активації;

m – кількість синаптичних зв'язків нейрону k [20].

До основних функцій активації належать порогова функція, також відома як функція Хевісайда, та сигмоїдальна, прикладом якої служить логістична функція.

1.4 Нейромережеві методи класифікації текстів

1.4.1 Рекурентні нейронні мережі

Рекурентні нейронні мережі (РНМ), або RNN, є сімейством нейронних мереж, що призначені для обробки послідовних даних. Подібно до того, як ЗНМ є нейронними мережами, що спеціалізується на обробці матриці значень X , таких як зображення, РНМ є нейронними мережами, які спеціалізується на обробці послідовності значень $x(1), \dots, x(n)$. Подібно до того, як згорткові нейронні мережі (ЗНМ) можуть легко масштабуватися відповідно до розмірів зображень, а деякі ЗНМ можуть обробляти зображення змінного розміру, РНМ можуть масштабуватися до для обробки довгих послідовностей, які були б не практичними у обробці у нейронних мережах, що не спеціалізуються на обробці послідовностей значень. Більшість РНМ також можуть обробляти послідовності змінної довжини [22]. РНМ обробляють послідовності даних поелементно, тобто шляхом ітерацій, і підтримують стан, що містить попередню інформацію, що була оброблена мережею. По суті, РНМ – це тип нейронної мережі, що має внутрішній цикл, зображений на рис. 1.6 [23].

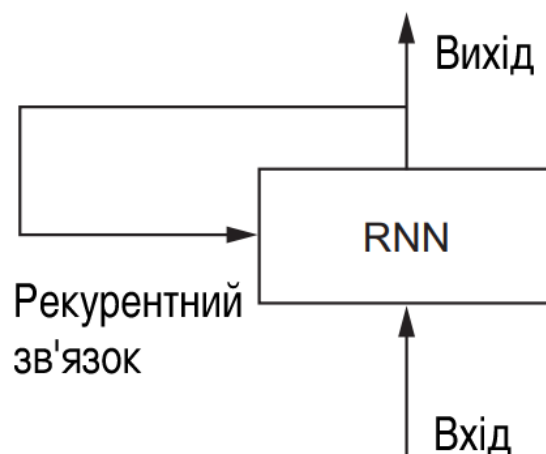


Рисунок 1.6 – Цикл РНМ [23]

РНМ знайшли широке використання у задачах ОПМ, оскільки на відміну від способу представлення та обробки тексту «торба слів», враховують зв'язки між словами за допомогою механізму запам'ятовування стану. Одним з можливих використання РНМ у сфері обробки тексту є аналіз його тональності [24]

Розглянемо приклад того, як РНМ обробляє послідовність слів. За приклад візьмемо фразу «The cat chased the mouse». Структура такої РНМ представлена на рис. 1.7 [21].

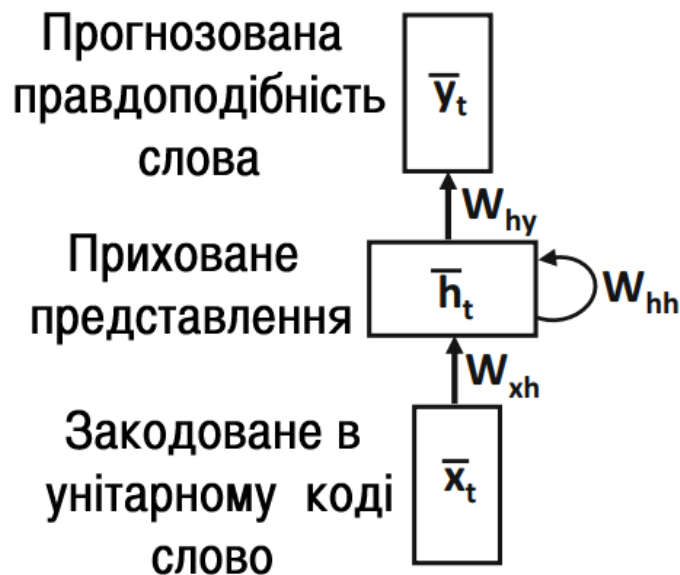


Рисунок 1.7 – Структура РНМ [21]

Внутрішній цикл, що наявний на рис. 1.7, призводить до того, що приховане представлення нейронної мережі буде змінюватися після читання кожного слова з послідовності. Для спрощення розуміння роботи РНМ можна її «розгорнути» в часі (рис. 1.8). Таким чином, її структура буде більше схожа на структуру мережі прямого розповсюдження [21].

Архітектура такої мережі підходить для моделювання мови. Мовна модель є концепцією в сфері ОПМ, за допомогою якої з врахуванням попереднього набору слів прогнозується наступне слово. Послідовність закодованих в унітарному коді слів подається по одному в нейронну мережу,

що представлена на рис. 1.7. У часі цей процес еквівалентний подачі окремих слів на входи у відповідні часові комірки на рис. 1.8. Комірка часу відповідає позиції слова в послідовності, яка починається з нуля (або одиниці), і збільшується на одиницю при переміщенні вперед. Так, наприклад, якщо вхідним словом є «The», то виходом буде вектор ймовірностей всього лексикону, що включає слово «cat». А коли вхідним словом є «cat», то знову виходом буде вектор ймовірностей наступного слова [21].

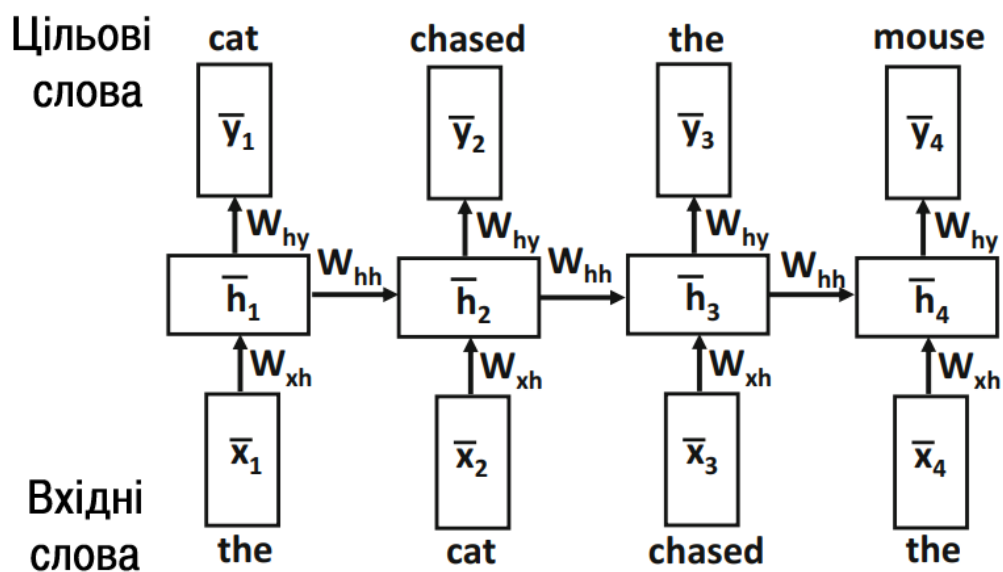


Рисунок 1.8 – Розгорнута в часі РНМ [21]

Перевагами використання РНМ для класифікації тексту є наявність знань про послідовність слів у тексті. Недоліками низька швидкість навчання і складність навчання на довгих послідовностях слів, що виражається невеликою або відсутністю корекції вагів перших шарів нейромережі під час зворотного поширення помилки.

1.4.2 Згорткові нейронні мережі

ЗНМ є спеціалізованим видом нейронних мереж для обробки даних, які мають топологію, подібну до сітки. До прикладів таких даних можна

віднести часові ряди, які можна розглядати як одномірну сітку, що відбирає дані через однакові інтервали часу, і зображення, які можна розглядати як двомірну сітку з пікселів. Назва "ЗНМ" вказує на те, що мережа використовує математичну операцію, яка називається згортка. Згортка – це спеціалізований вид лінійної операції. ЗНМ – це просто нейронні мережі, які використовують операцію згортки замість звичайного множення матриць принаймні в одному зі своїх шарів [22].

Операція згортки – це операція скалярного добутку між структурованим у матрицю набором вагових коефіцієнтів та аналогічними структурованими у матрицю вхідними даними, що отримані з різних просторових локалізацій у вхідному просторі. Цей тип операції корисний для даних з високим рівнем просторової або іншої локальності, таких як дані зображень. Тому ЗНМ визначаються як мережі, які використовують операцію згортання принаймні в одному шарі, хоча більшість ЗНМ використовують цю операцію в декількох шарах [22].

На рис. 1.9 наведено приклад архітектури простої загорткової нейронної мережі, що складається з п'яти шарів [25]. В основному функціоналі наведеного прикладу загорткової нейронної мережі можна виділити чотири елементи:

- вхідний шар містить значення вхідних даних, таких як пікселі зображення;
- згортковий шар визначає вихід нейронів, які пов'язані з локальними областями вхідних даних, шляхом виконання операції згортки, тобто обчислення скалярного добутку між їх вагами та локальною областю вхідних даних, що пов'язана з вхідним масивом даних. До отриманих значень застосовуються функція активації;
- операція пулінгу, або субдискретизація виконує зменшення розмірності заданих вхідних даних, тобто зменшення кількості параметрів;

– Повнозв'язні шари виконують ті ж самі функції, що і в звичайних нейронних мережах, тобто намагатимуться отримати оцінки класів зі значень активації [25].

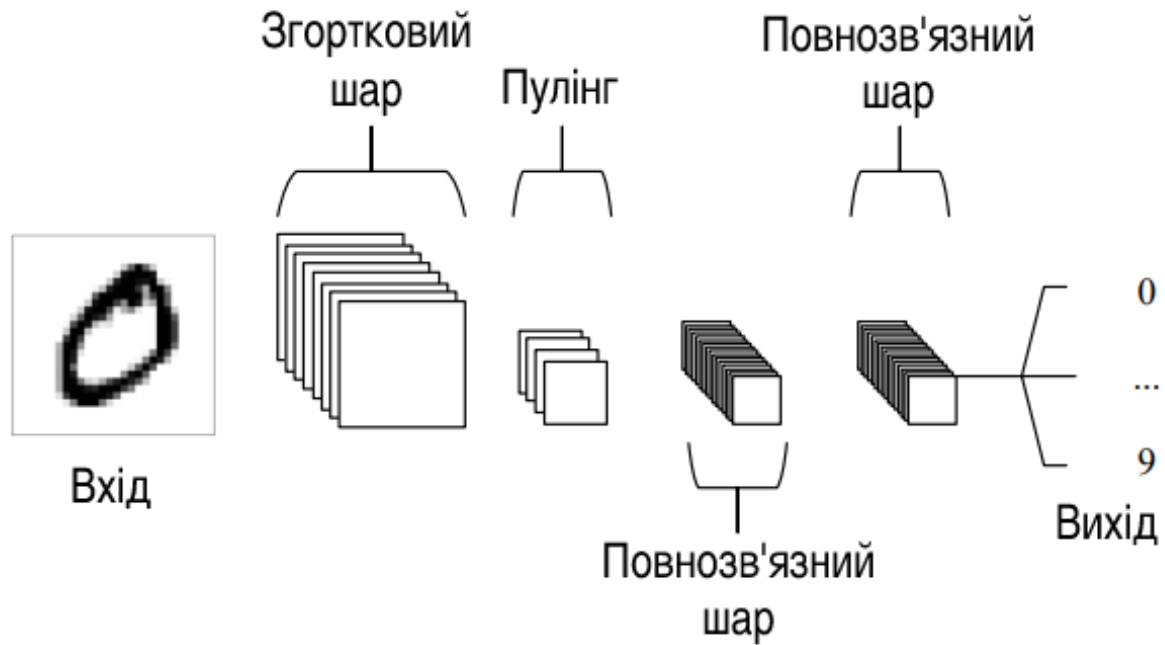


Рисунок 1.9 – Архітектура загорткової нейронної мережі [25]

Архітектура загорткової мережі для класифікації текстів дуже схожа на архітектуру загорткової мережі для обробки зображень, з тією відмінністю, що для класифікації текстів використовуються одновимірні згортка та одновимірний пулінг над векторами-представленнями вхідних слів. Такий підхід дозволяє зменшити розмірність представлення вхідного слова.

Перевагами такого методу для класифікації текстів є відносно менша кількість параметрів моделі і швидкість навчання. До недоліків можна віднести брак знань про послідовність тексту та взаємозв'язки між словами у тексті.

1.4.3 Порівняння нейромережових методів класифікації текстів

Проаналізувавши методи класифікації текстів, можна порівняти їх за спільними характеристиками. Крім того, з описаними методами також можна порівняти нейромережу, що використовує лише повнозв'язні шари.

У таблиці 1.1 наведено порівняння нейромережових методів класифікації текстів.

Таблиця 1.1 – Порівняння нейромережових методів класифікації текстів

Критерій	Нейромережа прямого поширення	ЗНМ	РНМ
Складність реалізації	Невелика	Середня	Середня
Складність навчання	Невелика	Середня	Достатньо висока
Швидкість навчання	Середня	Середня	Низька
Залежність від розмірності даних	Так	Ні	Ні
Якість класифікації текстів	Низька	Середня	Висока

1.5 Використання FPGA у задачах апаратного прискорення

1.5.1 Внутрішня структура FPGA

FPGA – різновид інтегральних схем, внутрішня структура яких може бути запрограмована після виготовлення. Таким чином, користувач може реалізовувати власні цифрові пристрої «у польових умовах» (in-field), тобто власноруч, без необхідності залучення промислових процесів у реалізації інтегральної схеми.

На рис. 1.10 зображено типову внутрішню структуру FPGA [26]. Основними структурними блоками мікросхеми FPGA є блоки логіки, пам'яті, помножувачі та порти введення/виведення. Між ними лінії зв'язку, які, окрім передачі інформації між блоками, можуть також бути запрограмованими, що надає гнучкості до процесу розробки.



Рисунок 1.10 – Внутрішня структура FPGA [26]

Розглянемо внутрішню будову мікросхеми FPGA фірми Altera сімейства Cyclone IV. На рис. 1.11 наведено будову так званого логічного елемента [27]. На прикладі логічного елемента FPGA Cyclone IV можна розглянути основні складові будь-якого логічного елемента. Основними

структурними елементами є look-up таблиця (look-up table), схема переносу (carry chain) та регістр (programmable register).

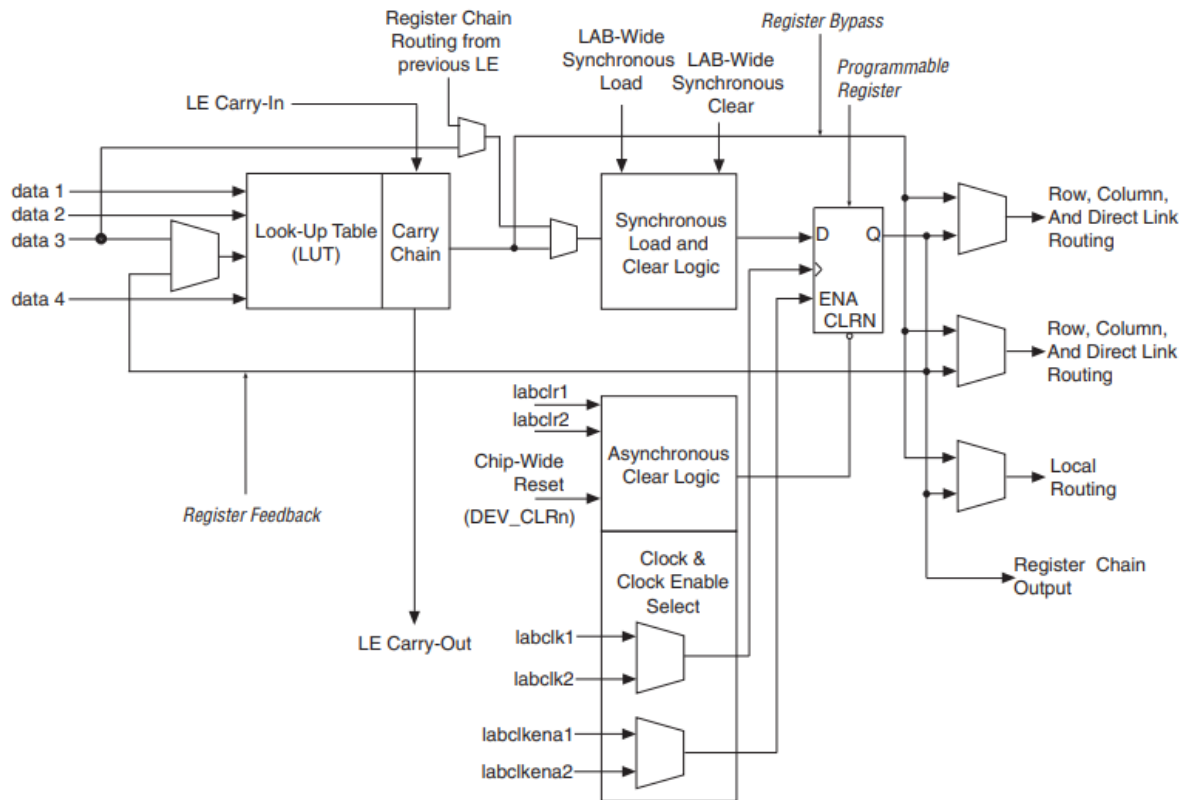


Рисунок 1.11 – Будова логічного елементу мікросхеми FPGA Cyclone IV [27]

Look-up таблиці використовується для формування будь-яких булевих функції. Вони діють за принципом звичайних таблиць істинності і можуть відтворювати функції різної кількості аргументів. На рис. 1.12 наведено внутрішню будову такої таблиці [26]. Оскільки ця таблиця може відтворити булеву функцію чотирьох аргументів, вона містить 16 комірок пам'яті, що зберігають кожний рядок таблиці істинності. Вибір необхідного рядка відбувається за допомогою мультиплексорів, з'єднаних у формі деревовидної структури.

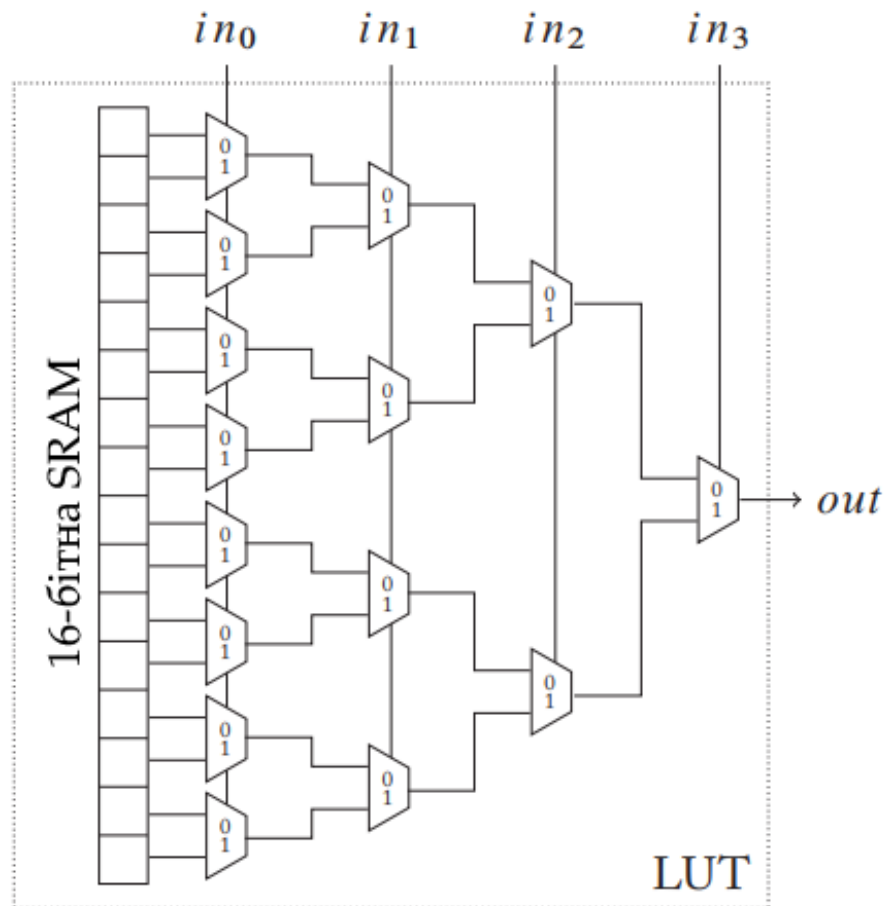


Рисунок 1.12 – Внутрішня будова Look-up таблиці [26]

Схема переносу, що присутня у структурі на рис. 1.10, необхідна для побудови арифметичних схем, а саме для реалізації суматора з послідовним перенесенням. Запам'ятовувальний елемент, що представлений у вигляді D-тригера, дозволяє будувати послідовні схеми.

На рис. 1.13 зображено, як з'єднуються логічні елементи FPGA Cyclone IV у так звані блоки логічних масивів (LABs). Кожен LAB містить 16 логічних елементів, керуючі сигнали, ланцюги переносу, ланцюги регістрів, місцеві з'єднання. Місцеві з'єднання направляють сигнали між логічними елементами одного масиву. Ланцюги регістрів направляють сигнал від одного логічного елемента певного масиву, до іншого логічного елемента сусіднього масиву. Спеціалізоване середовище розробки Quartus II розміщує відповідну логіку в LAB або суміжних LAB, дозволяючи використовувати

місцеві та регістрові ланцюги для підвищення продуктивності та ефективного використання площі мікросхеми [27].

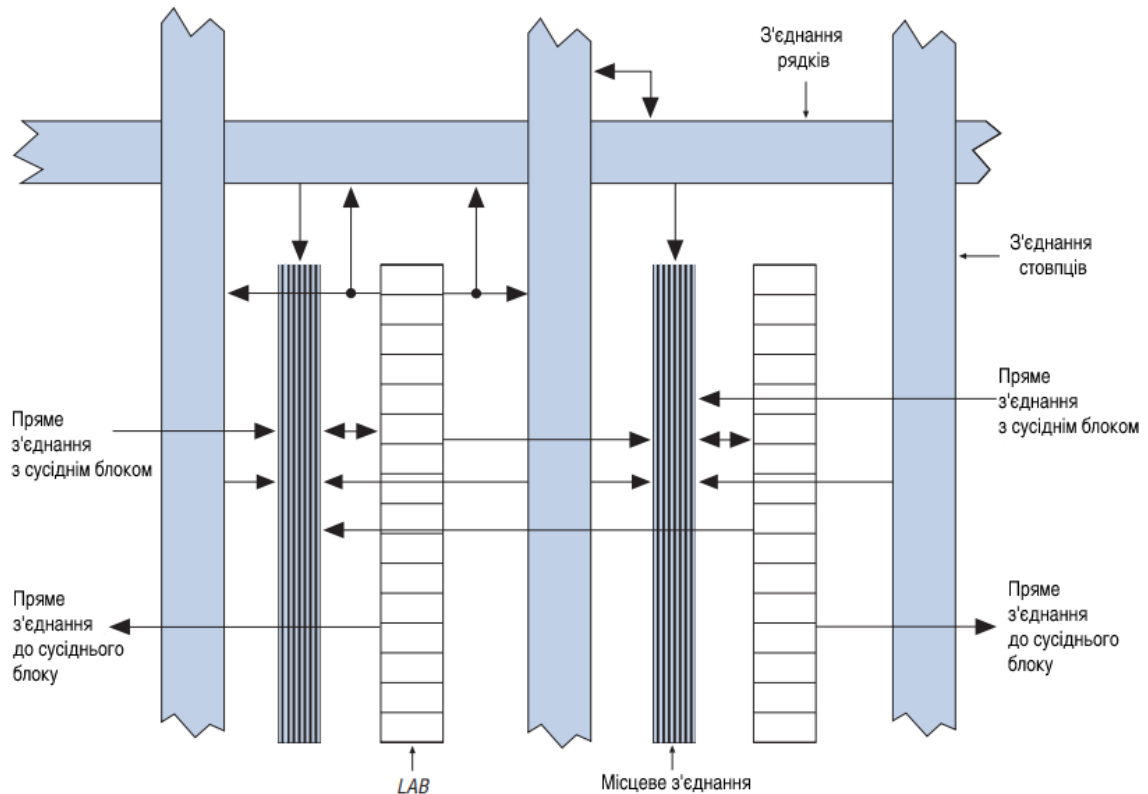


Рисунок 1.13 – Блоки логічних масивів мікросхеми FPGA Cyclone IV [27]

На рис. 1.14 зображені різні типи маршрутизацій, тобто з'єднань між блоками всередині FPGA. Маршрутизація класифікується відповідно до того, як з'єднані логічні блоки між собою, або з іншими блоками, такими як блоки пам'яті, помножувачі чи блоки введення/виведення. Маршрутизації в FPGA можна приблизно класифікувати на чотири типи:

- матричний (або перехресний) тип, що дозволяє поєднувати всі логічні блоки між собою одночасно. Така структура маршрутизації зазвичай зустрічається в пристроях програмованої матричної логіки;
- одновимірний тип, у якому блоки поєднанні провідниками тільки в одному напрямку;

– двовимірний (або острівний) тип, який є найпоширенішим, і з'єднувальні канали такого роду поєднують логічні блоки як у горизонтальному, так і у вертикальному напрямках;

– ієрархічний тип, у якому з'єднання розподіляються на декілька рівнів, на кожному з яких з'єднуються між собою блоки [29].

Маршрути складаються з монтажних доріжок і перемикачів, що можуть програмуватися. Окремий шлях маршрутизації визначається відповідно до значень бітів пам'яті конфігурації [29].

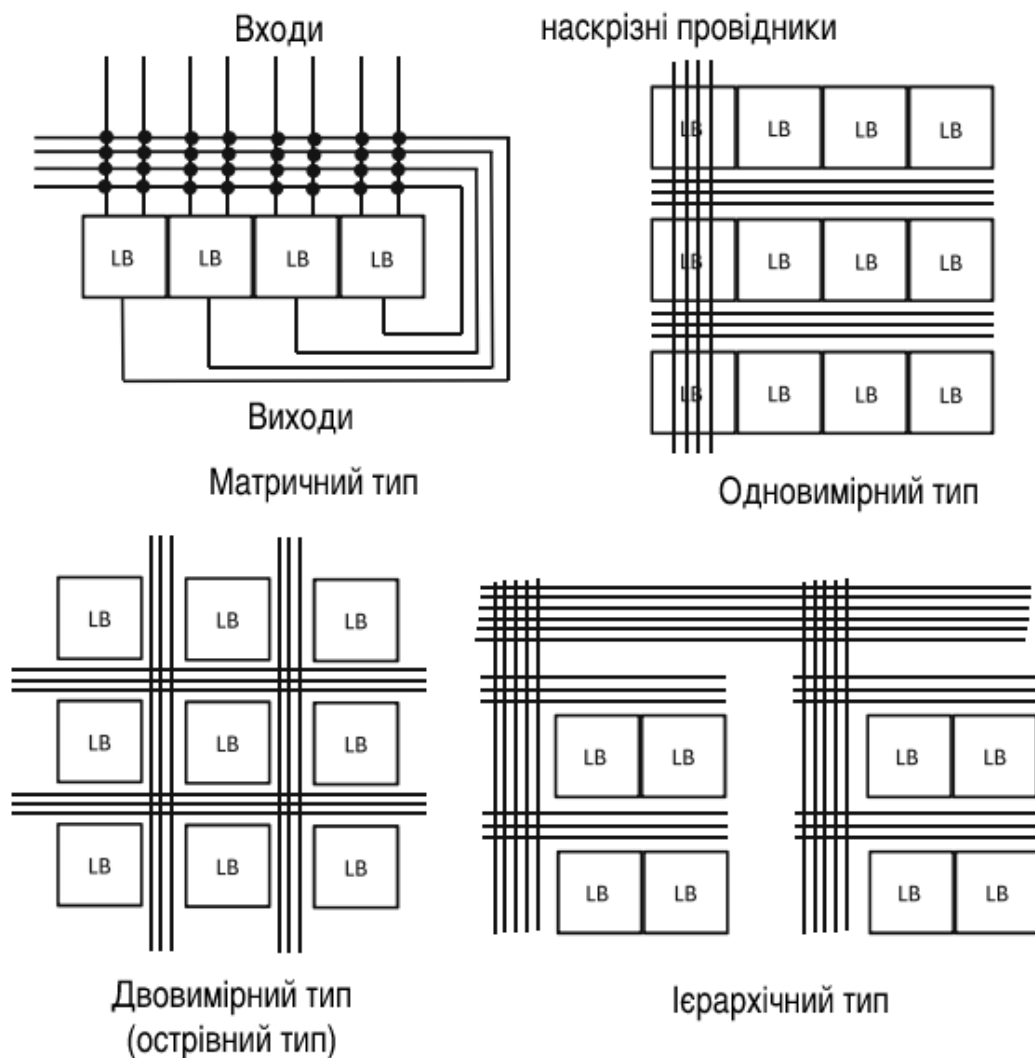


Рисунок 1.14 – Типи маршрутизації в FPGA [29]

Окрім показаний на рис. 1.10 основний структурних блоків, в FPGA присутні також маршрути розповсюдження тактових сигналів, блоки PLL та DLL, інтерфейс для зв'язку з FLASH-пам'яттю, JTAG, а в деяких сімействах блоки DSP.

Таким чином FPGA містить багату елементну базу для реалізації різноманітних цифрових пристроїв, а можливості конфігурації кожного структурного елементу надають гнучкості при розробці.

1.5.2 Принципи апаратного прискорення

Апаратне прискорення полягає у розробці апаратного забезпечення, що, на відміну від процесорів загального призначення, виконує специфічний алгоритм з метою прискорення швидкості обчислень та/або зменшення кількості використовуваних ресурсів.

У спеціалізованих апаратних прискорювачів використовується чотири методи для підвищення продуктивності та ефективності:

- спеціалізація даних: операції над специфічними для задачі типами даних можуть виконуватись за один цикл, що на процесорі загального призначення може зайняти десятки циклів. Спеціалізоване апаратне забезпечення для виконання часто використовуваних функцій підвищує як продуктивність, так і ефективність. Прикладом такої обробки є виконання арифметичних операцій над числами з рухомою комою, як описано у роботі [30];

- паралелізм: високий ступінь паралелізму, який часто використовується на декількох рівнях, забезпечує підвищення продуктивності. Для того, щоб обчислення були ефективними, блоки паралельної обробки даних повинні слідувати принципу локальності і робити дуже мало звернень до глобальної пам'яті, інакше їх продуктивність буде обмежена пам'яттю;

– локальна та оптимізована пам'ять: зберігаючи необхідні структури даних у багатьох невеликих локальних комірках пам'яті, можна досягти дуже високої пропускної здатності пам'яті при низьких витратах енергії. Схеми доступу до глобальної пам'яті оптимізуються для досягнення максимально можливої пропускної здатності. Необхідні структури даних можуть бути стиснуті для збільшення пропускної здатності. Доступ до пам'яті збалансований за каналами пам'яті та ретельно спланований для максимально ефективного використання пам'яті;

– зниження накладних витрат: спеціалізоване апаратне забезпечення усуває або зменшує накладні витрати на виконання програм [31].

Як правило для задач апаратного прискорення використовуються FPGA з наступних причин:

- можливість перепрограмування, на відміну від ASIC;
- можливість реалізації апаратного паралелізму;
- простота зміни розміру даних (наприклад, операндів в арифметичних операціях) [32].

На сьогоднішній день ПЛІС можуть використовуватися як у вигляді автономних пристроїв, так і в складі більш складних систем, що включають різні типи процесорів, пам'яті, інтерфейсів тощо [32].

1.6 Постановка задачі розробки

У даній роботі необхідно дослідити та розробити засоби програмної та апаратної класифікації текстів з використанням нейронних мереж, проаналізувати отримані результати.

Програмна складова системи класифікації текстів представляє собою наступне:

- програму тренування нейронної мережі;

- хост-програма, що керує процесом обчислення на апаратному прискорювачі.

Апаратна складова системи класифікації текстів містить наступне:

- апаратний прискорювач, що представляє собою платформу з мікросхемою FPGA;
- програми ядер, що виконують код нейронної мережі на апаратному прискорювачі.

1.7 Висновки за розділом 1

У даному розділі було проаналізовано предметну область, розглянуто внутрішню структуру FPGA та принципи апаратного прискорення, а також розглянуто та порівняно нейромережеві методи класифікації текстів.

2 ОГЛЯД І АНАЛІЗ МЕТОДІВ ТА ЗАСОБІВ КЛАСИФІКАЦІЇ ТЕКСТІВ

2.1 Вибір методу розробки на FPGA

Підвищення рівня абстракції при апаратному проектуванні має важливе значення для оцінки архітектурних рішень, синтезу, верифікації, організації пам'яті і управління енергоспоживанням. HLS (High-level synthesis – високорівневий синтез) дозволяє повторно використовувати одну і ту ж високорівневу специфікацію, призначену для широкого спектру проектних вимог і технологій [33].

Як правило, при HLS розробник створює специфікацію застосунку, що повинен бути реалізований у вигляді процесора спеціального призначення, виділеного співпроцесора або будь-якого іншого спеціального апаратного блоку, такого як контролер переривань, міст, арбітр, інтерфейсний блок або спеціальний функціональний блок, за допомогою високорівневого опису бажаної функціональності, використовуючи HLL (High level language – високорівнева мова програмування). Таким чином, цей перший крок включає в себе написання функціональної специфікації, в якій функція приймає на вхід дані одночасно, виконує всі обчислення без будь-якої затримки і надає всі свої вихідні дані також одночасно. На цьому рівні абстракції змінні (структури і масиви) і типи даних (як цілі так і з рухомою комою) не пов'язані ні з областю апаратного проектування (біти, бітові вектори), ні з вбудованим програмним забезпеченням. Таким чином, апаратна реалізація вимагає перетворення типів даних з рухомою комою та цілих чисел у бітові типи даних певної довжини з прийнятною точністю обчислень, а також створення оптимізованої апаратної архітектури, яка б виходила з цієї бітової специфікації [33].

RTL являє собою метод розробки при якому робота розроблюваного пристрою представляється у вигляді потоку даних між регістрами. При переході від одного регістру до іншого дані оброблюються певним чином на

операційних блоках. Розробка при такому методі зазвичай виконується за допомогою таких HDL, як VHDL або Verilog. Хоча код, написаний на HDL може запускатися на різних типах програмованих мікросхем і може бути краще оптимізованим за код, згенерований HLS або OpenCL, написання його є процесом трудомістким.

У роботах [34], [35] та [36] проводиться порівняння методів розробки HLS та RTL. Зокрема, досліджуються такі критерії як простота розробки, кількість рядків коду, час розробки, рівень оптимізації та використання апаратних ресурсів. Результати порівняння підсумовано у таблиці 2.1.

Враховуючи переваги HLS за такими критеріями, як час розробки, рівень абстракції, кількість рядків коду та наявність вбудованих методів, методом розробки обрано HLS.

Таблиця 2.1 – Порівняння методів розробки на FPGA

Критерій	HLS	RTL
Рівень абстракції	+++	+
Кількість рядків коду	+++	+
Часові витрати на розробку	+++	+
Рівень оптимізації	++	+++
Наявність вбудованих методів	++	+
Використання апаратних ресурсів	++	+++

2.2 Вибір програмного і апаратного середовища

Для реалізації апаратної частини системи класифікації текстів необхідна наявність платформи з мікросхемою FPGA. Окрім мікросхеми, має

бути також присутній інтерфейс PCI для зв'язку з комп'ютером та оперативна пам'ять для зберігання проміжних результатів. Два найбільших виробника чипів FPGA, Intel та Xilinx, пропонують свої плати прискорювачів на базі FPGA. Сервіс AWS пропонує так звані інстанси EC2 F1, представляють собою обчислювальні модулі з FPGA, архітектура яких зображена на рис. 2.1 [37].

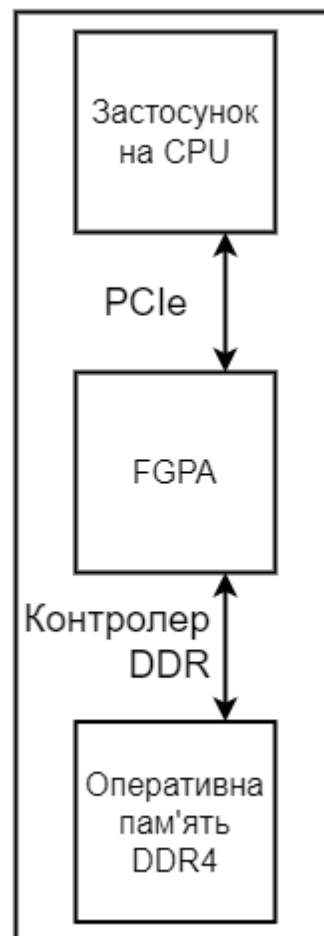


Рисунок 2.1 – Архітектура інстансу EC2 F1 [37]

Враховуючи високу вартість плат прискорювачів, необхідність у самостійному налаштуванні середовища розробки та відсутність потреби у високій обчислювальній здатності для даної задачі, перевагу було надано сервісу AWS.

Середовище розробки сервісу AWS надає інструменти для написання коду мовами C++, C та OpenCL.

OpenCL – це індустріальний стандарт для програмування комп'ютерів, що складаються з комбінації CPU, GPU та інших процесорів. Одна з причин, чому OpenCL є настільки важливим і має потенціал для трансформації індустрії програмного забезпечення полягає в тому, що за допомогою OpenCL можна написати одну програму, яка може працювати на широкому спектрі систем, від мобільних телефонів, ноутбуків, до вузлів у масивних суперкомп'ютерах. Програміст на OpenCL повинен явно визначити платформу, її контекст і те, як планується робота на різних пристроях, оскільки OpenCL не працює з програмними абстракціями. Таким чином, забезпечується високий рівень переносимості і використання апаратного забезпечення [38].

OpenCL API – це мова C з C++ Wrapper API, що визначена в термінах C API. Існують сторонні прив'язки для багатьох мов, включаючи Java, Python та .NET. Код, який виконується на пристрої OpenCL, що в загальному випадку не є тим же пристроєм, що і головний центральний процесор, написаний на мові OpenCL C. OpenCL C є обмеженою версією мови C99 з розширеннями, придатними для виконання паралельного з даними коду на різноманітних гетерогенних пристроях [39].

На рис. 2.2 зображено процес розробки застосунку у середовищі SDAccel, що використовується у Amazon EC2 F1 [40].

Першим кроком у процесі розробки є отримання написання коду OpenCL та перевірка його функціональної коректності шляхом виконання програмної симуляції, що називається процесорною емуляцією. Це найшвидша частина процесу проєктування, яка дозволяє швидко перевірити функціональність проєкту. Цей крок, як правило, вимагає виконання тестів основного коду, які містять вхідні дані та перевіряють вихідні дані алгоритму. Після перевірки функціональності оцінюється продуктивність кожного окремого ядра, тобто продуктивність окремої обчислювальної одиниці та використання нею ресурсів. Це дає попереднє уявлення про кінцевий приріст продуктивності, беручи до уваги цільовий апаратний

пристрій, а також згенеровані обчислювальні блоки для виконання програми. SDAccel генерує апаратне забезпечення з ядра OpenCL на рівні робочої групи. Для підвищення продуктивності можуть бути створені декілька обчислювальних блоків, які керуються головним кодом. Кожна робоча група може виконувати свої робочі елементи або конвеєрним способом, або розгортаючи їх, або обома способами. Як правило, конвеєризація забезпечує найкращий компроміс між апаратними затратами та продуктивністю [40].

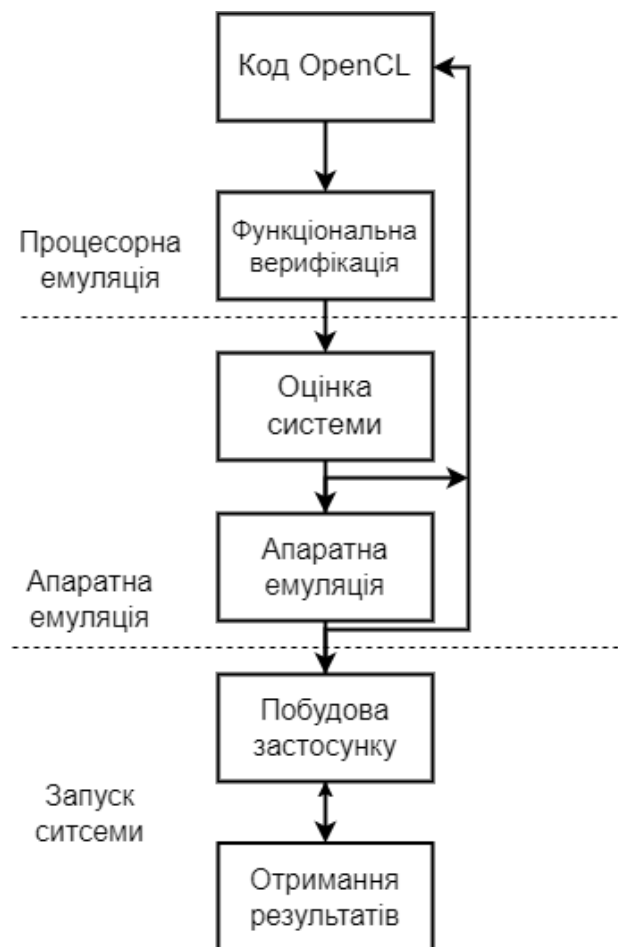


Рисунок 2.2 – Процес розробки застосунку у середовищі SDAccel [40]

Згодом, RTL-симуляція, яку в контексті процесу розробки також називають апаратною емуляцією, використовується для перевірки того, що обчислювальні блоки, створені для всіх ядер, функціонують правильно, і для аналізу їх загальної продуктивності. У той час як процесорна емуляція перевіряє функціональну коректність програми, її продуктивність

перевіряється за допомогою апаратної емуляції. Заключним етапом є пакування згенерованої системи для розгортання на платах на основі FPGA [40].

Для написання програми для інстанса EC2 F1 обрано OpenCL, оскільки даний фреймворк містить безліч вбудованих функцій для створення паралельних додатків.

2.3 Вибір мови програмування для створення нейромережі

Проведемо порівняння таких поширених у сфері машинного навчання мов програмування, як C++, Java та Python.

Java – алгоритмічна, паралельна, об'єктно-орієнтована мова програмування. Як і у випадку з синтаксисом, семантично подібна до всіх представників сімейства C, особливо C#. Java є суворо типізованою мовою, що має набір правил і певні операції, які можуть бути виконані тільки з певними типами даних. Також Java підтримує явне приведення подібних типів, наприклад, double до типу float, але при цьому можлива втрата даних. Java-додатки зазвичай компілюються в байт-код (файл класу), який може виконуватися на будь-якій віртуальній машині Java (JVM) незалежно від архітектури комп'ютера. Процес трансляції JVM коду Java в байт-код прихований від користувача. Це означає, що спосіб, яким буде трансльовано код, залежить від алгоритмів трансляції JVM. Цей дозволяє уникнути таких звичних для програмістів на мові C проблем, як переповнення буферів і витоку пам'яті. Всі об'єкти, що визначаються користувачем в Java виділяються динамічно, а за їх звільнення відповідає збирач сміття. Хоча Java не є компільованою мовою і парадигма компільованих мов наполягає на перевагах компільованих мов у швидкості обробки даних, це твердження є досить спірним. Вся логіка коду на Java повинна бути реалізована в класах. Об'єктно-орієнтований підхід до програмування вносить в написання коду певний рівень абстракції і об'єкти, що створюються програмістом, можуть

асоціюватися з об'єктами з реального життя, що спрощує розуміння структури програми і повторне використання коду. З цієї точки зору Java є більш ефективною, ніж, наприклад, C++, яка здатна поєднувати в собі обидва підходи – об'єктний та процедурний. Таке поєднання підходів може плутати програміста, який використовує застарілий код [41].

Python – універсальна мова програмування високого рівня, яка широко використовується останнім часом. Її філософія підкреслює важливість читабельності коду, а синтаксис дозволяє програмістам виражати концепції в меншій кількості рядків коду, ніж це було б можливо в таких мовах, як C. Конструкції мови дозволяють користувачеві писати зрозумілі програми як малого, так і великого розміру. Найважливішою особливістю мови Python є те, що вона підтримує декілька парадигм програмування, включаючи об'єктно-орієнтоване, імперативне та функціональне або процедурні програмування. Python підтримує динамічну систему типів та автоматичне управління пам'яттю, а також має велику стандартну бібліотеку. Інтерпретатори Python доступні для багатьох операційних систем. Python здобув популярність завдяки тому, що є мовою, дружньою до початківців, і замінив Java як найпопулярніша вступна мова для початківців. Python є дуже гнучкою мовою програмування, завдяки можливості використовувати модульні компоненти, які були розроблені на інших мовах програмування. Наприклад, можна написати програму на C++ та імпортувати її в програму на Python як модуль. Статус Python як мови програмування, що найшвидше розвивається, завдячую різкому зростанню її використання в області науки про дані. Незважаючи на те, що всі права на цю програму зарезервовані за інститутом Python, вона має відкритий вихідний код і не має обмежень у використанні, зміні та розповсюдженні. Можна вільно використовувати та розповсюджувати Python, навіть у комерційних цілях. На відміну від C/C++, немає необхідності у контролі над такими складними задачами, як управління пам'яттю, збір сміття тощо [42].

C++ – мова програмування загального призначення з ухилом у бік системного програмування, що підтримує ефективні низькорівневі обчислення, абстрагування даних, об'єктно-орієнтоване програмування та узагальнене програмування. За винятком незначних деталей, C++ є підмножиною мови C. C++ була розроблена для того, щоб бути кращою за C, підтримувати абстракцію даних, підтримувати об'єктно-орієнтоване програмування. Перша мета C++ - бути "кращою за C" шляхом забезпечення кращої підтримки стилів програмування, для яких C найчастіше використовується. Це, в першу чергу, передбачає надання функцій, які роблять найпоширеніші помилки малоймовірними. C++ забезпечує підтримку абстракції даних: програміст може визначати типи, які можна використовувати так само зручно, як і вбудовані типи, і подібним чином. Традиційно програма на C або C++ складається з ряду вихідних файлів, які окремо компілюються в об'єктні файли. Ці об'єктні файли потім зв'язуються разом для отримання виконуваної форми програми [43].

У таблиці 2.2 наведено порівняння мов програмування Python, C++ та Java.

Таблиця 2.2 – Порівняння мов програмування

Критерій	Python	C++	Java
Типізація	Динамічна	Статична	Статична
Простота	+++	+	++
Швидкість	++	+++	++
Кількість модулів і бібліотек	+++	++	+++
Підтримка методів машинного навчання	+++	+	++
Кількість учбових та довідкових ресурсів	+++	+	++

Враховуючи переваги та недоліки мов у таблиці 2.2, перевагу у розробці нейронних мереж надано мові Python.

2.4 Вибір фреймворку глибинного навчання

TensorFlow – популярна та гнучка бібліотека машинного навчання. Вона має як низькорівневі, так і високорівневі API, такі як Keras та Estimators, та підтримує декілька мовних інтерфейсів, окрім Python. Ця бібліотека та пов'язані з нею інструменти дозволяють користувачам ефективно програмувати та навчати нейронні мережі та інші моделі машинного навчання, а також розгорнути їх у виробництво. Основні алгоритми TensorFlow написані на високооптимізованих мовах C++ і CUDA, паралельних обчислювальних платформах та API для паралельних обчислень, створених компанією NVIDIA. Вона має API, доступні на декількох мовах. Найбільш повним і стабільним є Python API. Інші офіційно підтримувані мови включають JavaScript, C++, Java, Go та Swift. Пакети сторонніх розробників доступні для більшої кількості мов, таких як C# та Ruby. TensorFlow надає такі основні будівельні блоки для створення нейронних мереж, як повністю з'єднаний шар, згортковий шар, рекурентний модуль нейронної мережі та нелінійні функції активації. TensorFlow також пропонує різні функції втрат, такі як перехресна ентропія та середньоквадратична помилка (MSE) [45].

PyTorch – це бібліотека для програм на мові Python, яка полегшує створення проєктів глибокого навчання. Вона забезпечує гнучкість і дозволяє виражати моделі глибокого навчання ідіоматичною мовою Python. Ця доступність і простота використання знайшла перших прихильників у дослідницькому співтоваристві, і за роки, що минули з моменту її першого випуску, бібліотека PyTorch перетворилася на один з найвідоміших інструментів глибокого навчання в різних сферах застосувань. PyTorch забезпечує таку основну структуру даних, як тензор, який є багатовимірним

масивом, що має багато спільного з масивами NumPy. PyTorch постачається з функціями для виконання прискорених математичних операцій на спеціалізованому обладнанні, що робить зручним проєктування архітектур нейронних мереж та їх навчання на окремих машинах або паралельних обчислювальних пристроях. Багато дослідників і практиків вважають, що його легко вивчати, використовувати, розширювати і налагоджувати. PyTorch пропонує дві речі, які роблять його особливо актуальним для глибокого навчання: по-перше, він забезпечує виконання прискорених обчислень з використанням графічних процесорів, часто даючи прискорення в діапазоні 50х в порівнянні з виконанням тих же обчислень на центральному процесорі. По-друге, PyTorch надає засоби, які підтримують чисельну оптимізацію загальних математичних виразів, які використовуються в глибокому навчанні. Зауважимо, що обидві можливості корисні для наукових обчислень в цілому, а не виключно для глибокого навчання. Фактично, можна сміливо характеризувати PyTorch як високопродуктивну бібліотеку з підтримкою оптимізації для наукових обчислень на мові Python [46].

Keras – це фреймворк глибокого навчання для Python, який забезпечує зручний спосіб визначення та навчання практично будь-якої моделі глибокого навчання. Keras був розроблений для дослідників з метою забезпечення швидкого проведення експериментів. Keras дозволяє безперешкодно виконувати один і той самий код на CPU або GPU. Він має зручний API, який дозволяє швидко створювати прототипи моделей глибокого навчання. Keras Має вбудовану підтримку ЗНМ (для комп'ютерного зору), РНМ (для обробки послідовностей) та будь-якої їх комбінації. Він підтримує довільні мережеві архітектури: моделі з декількома входами або декількома виходами, спільне використання шарів, спільне використання моделей і так далі. Це означає, що Keras підходить для побудови практично будь-якої моделі глибокого навчання, від генеративної змагальної мережі до нейронної машини Тьюринга. Keras використовується в

Google, Netflix, Uber, CERN, Yelp, Square та сотнях стартапів, що працюють над широким спектром проблем. Keras – це бібліотека рівня моделей, що надає високорівневі будівельні блоки для розробки моделей глибокого навчання. Вона не обробляє низькорівневі операції, такі як маніпуляції з тензорами та диференціювання [23].

ONNX – це відкрита екосистема, яка дає можливість розробникам штучного інтелекту обирати правильні інструменти в міру розвитку проєкту. ONNX надає формат з відкритим вихідним кодом для моделей штучного інтелекту, як глибинного навчання, так і традиційного ML. ONNX визначає розширювану модель обчислювального графа, а також вбудовані оператори і стандартні типи даних. Наразі розробники ONNX зосереджуються на можливостях, необхідних для інференції (скорингу). ONNX широко підтримується і може бути знайдений у багатьох фреймворках, інструментах та апаратному забезпеченні. Забезпечення сумісності між різними фреймворками та спрощення шляху від дослідження до виробництва, що надає ONNX, допомагає збільшити швидкість інновацій у спільноті штучного інтелекту [47].

Враховуючи приведені вище особливості кожного з фреймворків та спираючись на дослідження у роботі [48], проведемо порівняння фреймворків PyTorch, Tensorflow, Keras та ONNX (табл. 2.3). Враховуючи переваги фреймворку PyTorch по кожному з критеріїв, перевагу у розробці було надано саме йому.

Таблиця 2.3 – Порівняння фреймворків глибинного навчання

Критерій	PyTorch	Tensorflow	Keras	ONNX
Простота	+++	++	+++	++
Гнучкість	+++	+++	++	+
Підтримка	+++	++	++	+
Поширеність	+++	++	++	+

2.5 Висновки за розділом 2

У даному розділі було обрано та обґрунтовано вибір програмних і апаратних засобів розробки, серед яких метод розробки на FPGA, програмне і апаратне середовище, мова програмування для створення нейромережі і фреймворк глибинного навчання.

3 РОЗРОБКА ПРОГРАМНИХ ЗАСОБІВ КЛАСИФІКАЦІЇ ТЕКСТІВ

3.1 Розробка нейромережі

3.1.1 Архітектура нейромережі

За результатами порівняння, наведеного у табл. 1.1, у якості основної архітектури для нейромережі було обрано РНМ.

Звичайні рекурентні нейронні мережі прості у реалізації, але мають суттєвий недолік: при довгій входній послідовності виникає проблема затухаючого градієнта. Проблема затухаючого градієнта – це ситуація, при якій глибока нейронна мережа не може поширювати корисну інформацію від вихідного шару до перших шарів під час зворотного поширення помилки [49].

Для вирішення цієї проблеми було запропоновано покращену архітектуру РНМ: LSTM (Long short-term memory – довга короткочасна пам'ять) [50].

Основою LSTM є декілька так званих вентилів, які відповідають за певну частину обробки інформації:

- «забувальний вентиль» відповідає за те, яка входна інформація залишається в блоці LSTM і передається до потоку стану блоку (клітини) LSTM;

- «вхідний вентиль» відповідає за те, яка входна інформація буде оновлена в потоці стану блоку LSTM;

- «вихідний вентиль» відповідає за те, яка оброблена інформація із потоку стану клітини буде передана на вхід наступному блоку LSTM і на вихід із поточного [51].

На рис. 3.1 зображено архітектуру звичайного блоку РНМ, а на рис. 3.2 зображено архітектуру блоку LSTM [51].

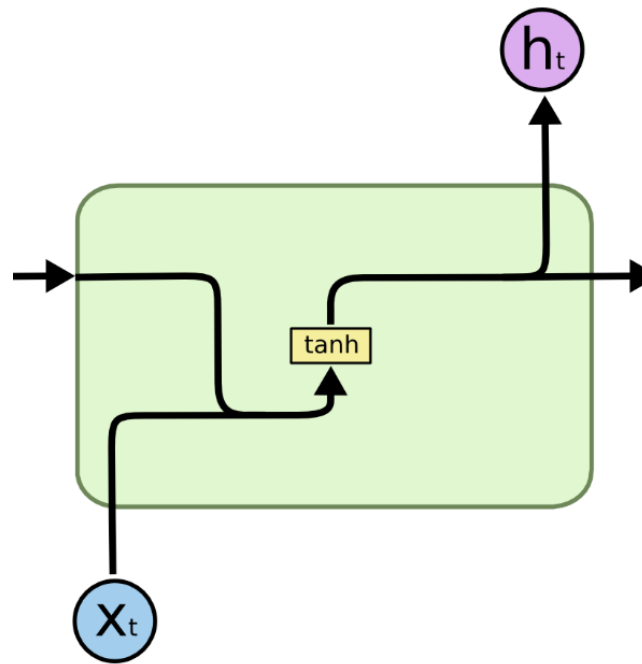


Рисунок 3.1 – Звичайний блок РНМ [51]

Умовні позначки:

x_t – вхід на кроці t ;

h_t – прихований стан на кроці t .

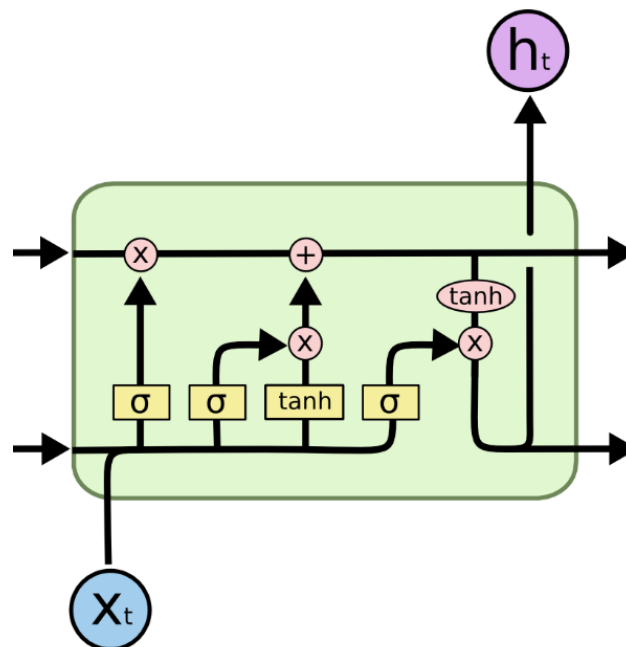


Рисунок 3.2 – Архітектура блоку LSTM [51]

Умовні позначки:

x_t – вхід на кроці t ;

h_t – прихований стан на кроці t ;

$\tanh$ – гіперболічний тангенс;

σ – сигмоїд.

Для експериментів було створено декілька архітектур нейронних мереж (рис. 3.3) з різними параметрами, а саме: кількість нейронів, шарів, тип рекурентного шару.

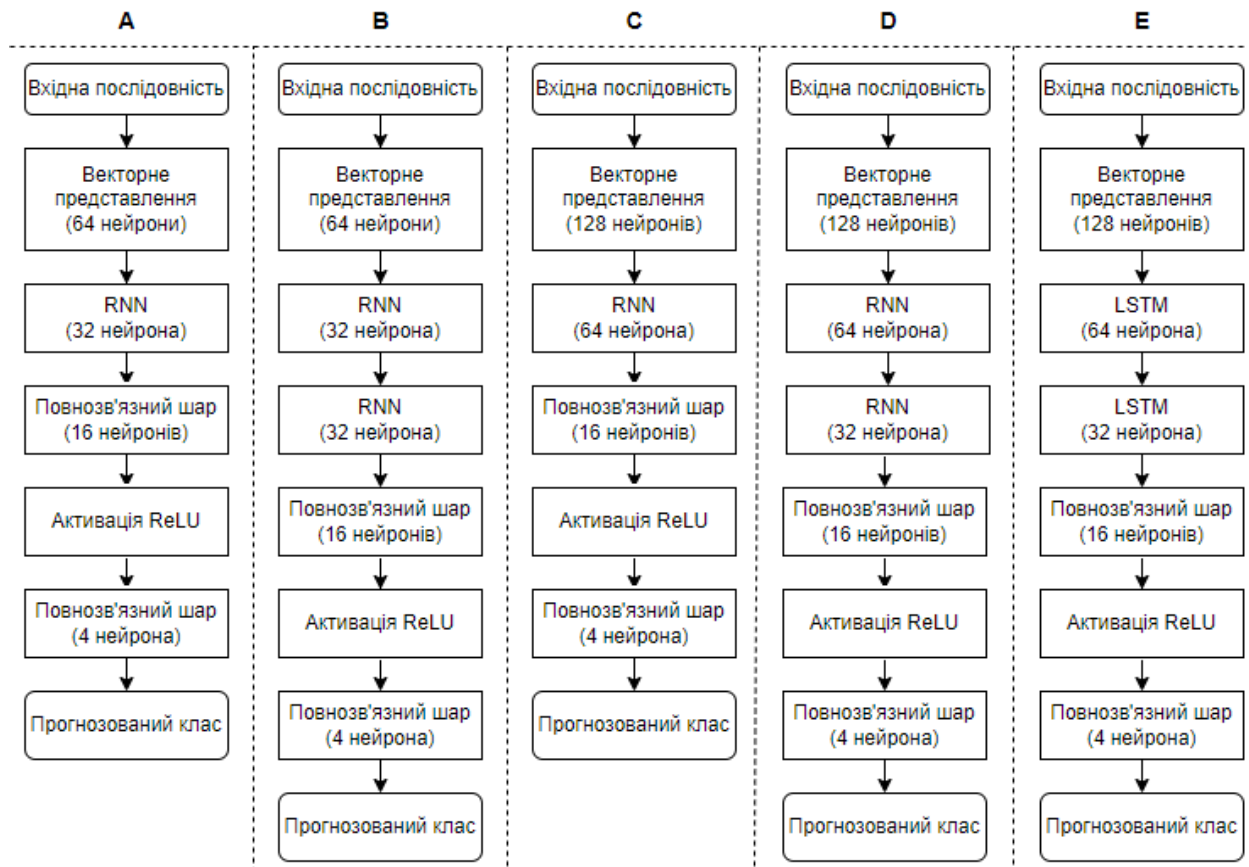


Рисунок 3.3 – Архітектури нейромереж для експериментів

3.1.2 Вибір датасету

На сьогоднішній день існує велика кількість датасетів для класифікації текстів. Зазвичай, основними відмінностями таких датасетів є розмір навчальної та тестової вибірок, тематика та кількість вихідних класів.

Серед популярних датасетів можна виділити наступні:

- IMDb Movie Reviews: датасет, що містить відгуки на фільми, які поділені на позитивні та негативні [52].

– Hate Speech: датасет, що складається з коментарів з інтернет-форума. Коментарі поділені на такі, що містять і такі, що не містять мову ворожнечі [53].

– AG News: датасет, що містить заголовки і описи до новин чотирьох великих категорій: «Світ», «Спорт», «Бізнес» і «Наука і техніка» [54].

Порівняння датасетів наведено у табл. 3.1.

Таблиця 3.1 – Порівняння датасетів

Критерій	IMDb	Hate Speech	AG News
Тематика	Кіно	Мова ворожнечі	Новини
Тип текстів	Відгуки на фільми	Коментарі	Заголовки і описи
Джерело	IMDb	Інтернет-форуми	Корпус текстів AG
Ліцензія	Не комерційний	Не комерційний	Не комерційний
Розмір	25 тисяч (навчальна вибірка) 25 тисяч (тестова вибірка)	10,5 тисяч	120 тисяч (навчальна вибірка) 7600 (тестова вибірка)
Класи	Позитивний, негативний	Позитивний, негативний	«Світ», «Спорт», «Бізнес» і «Наука і техніка»

Для вирішення задачі класифікації текстів було обрано датасет AG News, оскільки він містить більше класів і має більший розмір.

3.1.3 Обробка даних

Оскільки нейронні мережі вміють працювати тільки з числовими даними, необхідно оброблювати вхідні текстові дані.

Для цього спершу створюється словник на основі датасету. Такий словник містить в собі пари слово-індекс. Створивши такий словник, далі зручно замінити слова вхідної послідовності на відповідні їм чисельні індекси. Також, зазвичай, нульовий індекс відводиться під слово, яке не зустрічається в словнику.

Після цього, створюється унітарний код для кожного індексу вхідного слова, розмір якого дорівнює розміру словника, і значення якого дорівнюють нулю, окрім значення по індексу слова, яке дорівнює одиниці.

Описаний процес схематично зображено на рис. 3.4.

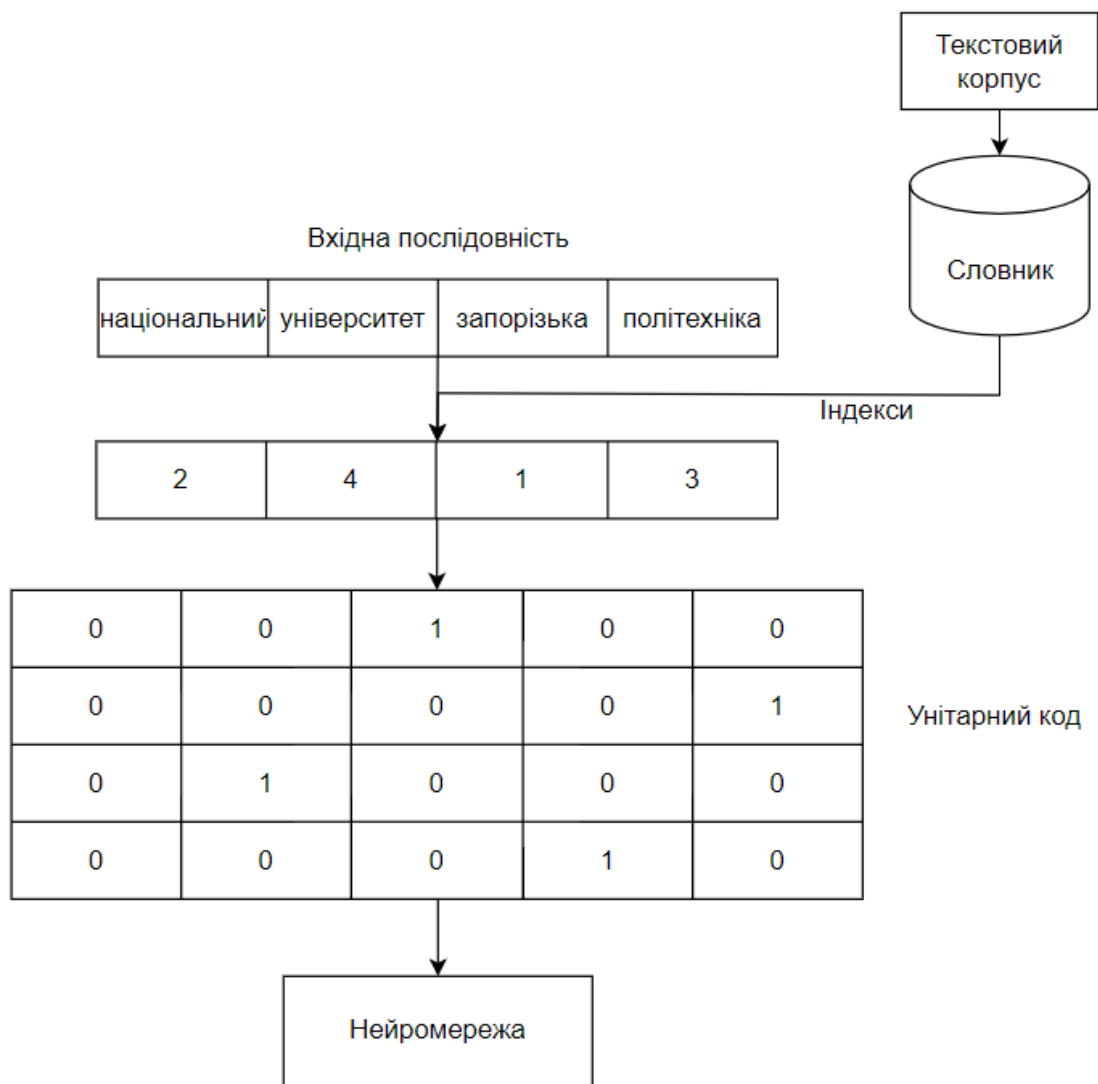


Рисунок 3.4 – Процес обробки вхідних даних

3.2 Навчання нейромережі

Усі нейромережі навчалися 100 епох, розмір пакету даних навчальної вибірки – 1024, алгоритм оптимізації – Adam, крок навчання – 0.0002.

Результати навчання мереж (рис. 3.3) наведено у табл. 3.2.

Таблиця 3.2 – Результати навчання моделей

Метрика/Модель	A	B	C	D	E
Влучність	0.82	0.81	0.83	0.87	0.88
Повнота	0.81	0.80	0.83	0.86	0.87
F1	0.82	0.81	0.83	0.86	0.88

За результатами навчання, найкращою моделлю виявилась нейромережа з двома шарами LSTM, структура якої наведена на рис. 3.5.

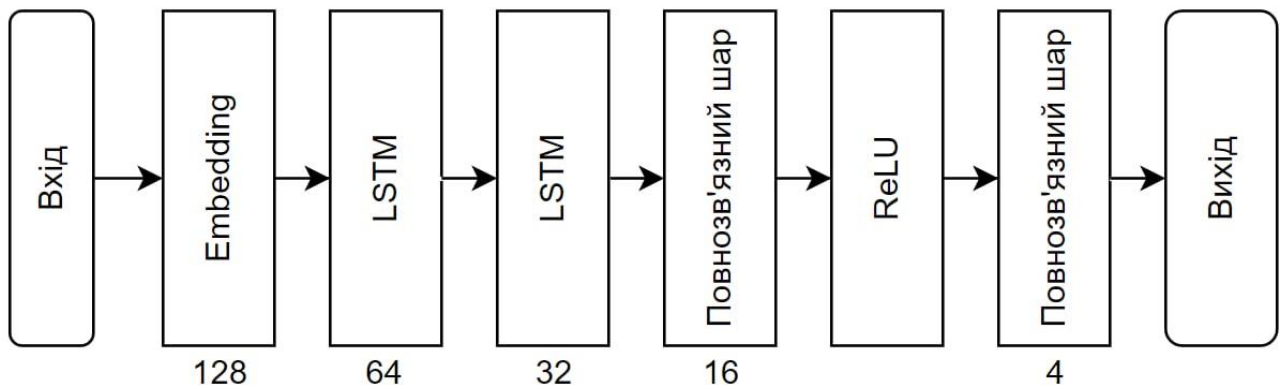


Рисунок 3.5 – Структура обраної моделі нейромережі

На рис. 3.6 зображено матрицю невідповідностей найкращої нейромережі на тестовому наборі даних.

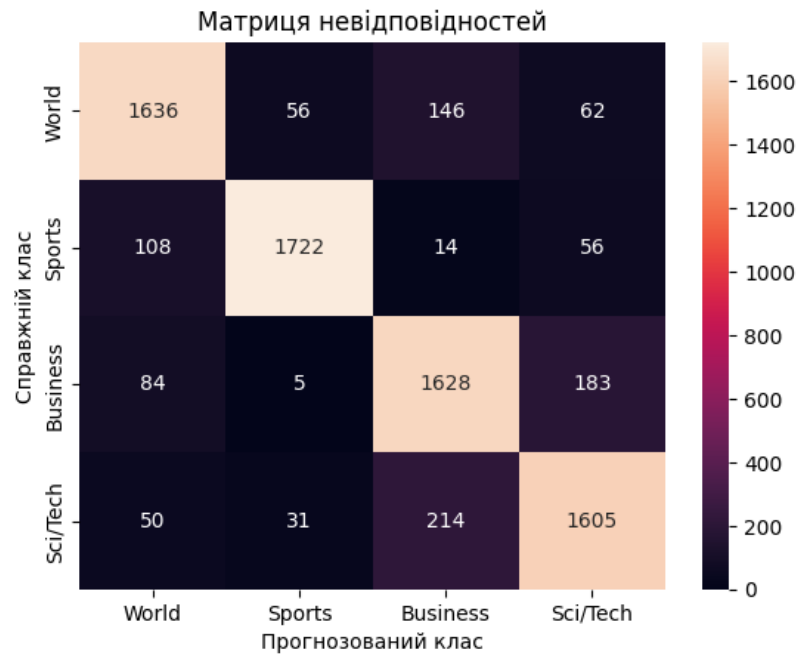


Рисунок 3.6 – Матриця невідповідностей моделі E

3.3 Висновки за розділом 3

У даному розділі було обрано датасет для навчання нейронної мережі та обрано найбільш ефективну за результатами навчання архітектуру нейронної мережі.

4 РОЗРОБКА АПАРАТНИХ ЗАСОБІВ ПРИСКОРЕННЯ КЛАСИФІКАЦІЇ ТЕКСТІВ

4.1 Розробка ядер для FPGA

Після вибору нейромережі для класифікації тексту можна починати створювати програму для запуску нейромережі на пристрої FPGA.

Умовно написаний код можна поділити на дві частини:

- код хост-програми, яка виконується як звичайна програма на процесорі і керує основним потоком операцій;
- код ядер.

Для запуску процесу виконання математичних операцій з матрицями на FPGA було розроблено декілька ядер за допомогою фреймворка OpenCL. Ядро – це невелика функція чи програма, що розроблена для виконання на певному пристрої. У табл. 4.1 наведено перелік розроблених ядер.

Таблиця 4.1 – Розроблені ядра

Ядро	Опис
MATMUL_KERNEL	Множення матриць
MATELPROD_KERNEL	Поелементне множення матриць
MATSUM_KERNEL	Сума матриць
BIAS_KERNEL	Додавання зміщення
COPY_KERNEL	Копіювання матриці
RELU_KERNEL	Функція активації ReLU
SIGMOID_KERNEL	Функція активації Sigmoid
TANH_KERNEL	Функція активації Tanh
SOFTMAX_KERNEL	Функція активації Softmax

Програма розроблювалась на хмарному сервісі AWS в наступні кроки.

Крок 1. Вибір серверу для проведення програмної (процесорної) та апаратної емуляції. Для розробки програми необхідно обрати сервер, який буде оптимальним по ціні і обчислювальною потужністю, тому було обрано інстанс AWS m5.xlarge;

Крок 2. Встановлення необхідного програмного середовища: для розробки і тестування програми на сервер було встановлено програмне забезпечення від Xilinx. Для полегшення цього процесу, AWS пропонує готовий пакет програмного забезпечення під FPGA разом з операційною системою, що називається «FPGA Developer AMI»;

Крок 3. Написання коду програми: код програми та ядер був написаний мовою програмування C++ з використанням фреймворку OpenCL;

Крок 4. Процесорна емуляція: швидка перевірка працездатності розробленої програми;

Крок 5. Апаратна емуляція: збірка ядер під пристрій FPGA;

Крок 6. Перенос створених ядер на сервер з пристроєм FPGA. Цей крок включає в себе збір розроблених ядер та частин програми, вибір сервера з пристроєм FPGA;

Крок 7. Тестування розробленої програми використовуючи пристрій FPGA: перевірка працездатності розробленої програми на пристрої FPGA на тестових даних.

4.2 Розробка хост-програми

За основу хост-програми було взято бібліотеку nn-on-fpgas [55], що містить базові класи та компоненти для запуску нейромереж на FPGA.

Хост-програма розроблювалась на основі принципів об'єктно-орієнтованого програмування, та складається з наступних класів (табл. 4.2).

Таблиця 4.2 – Класи розробленої хост-програми

Клас	Опис
TextClassifier	Основний клас, що описує структуру нейромережі
DenseLayer	Клас, що містить ваги повнозв'язного шару та логіку їхнього застосування
EmbeddingLayer	Клас, що містить ваги для представлень слів та логіку їхнього застосування
LSTMLayer	Клас, що містить ваги для шару LSTM та логіку їхнього застосування
Matrix	Клас, що описує основні алгоритми обробки матриць та логіку збереження даних у пам'яті

UML-діаграма класів хост-програми зображена на рис. 4.1.

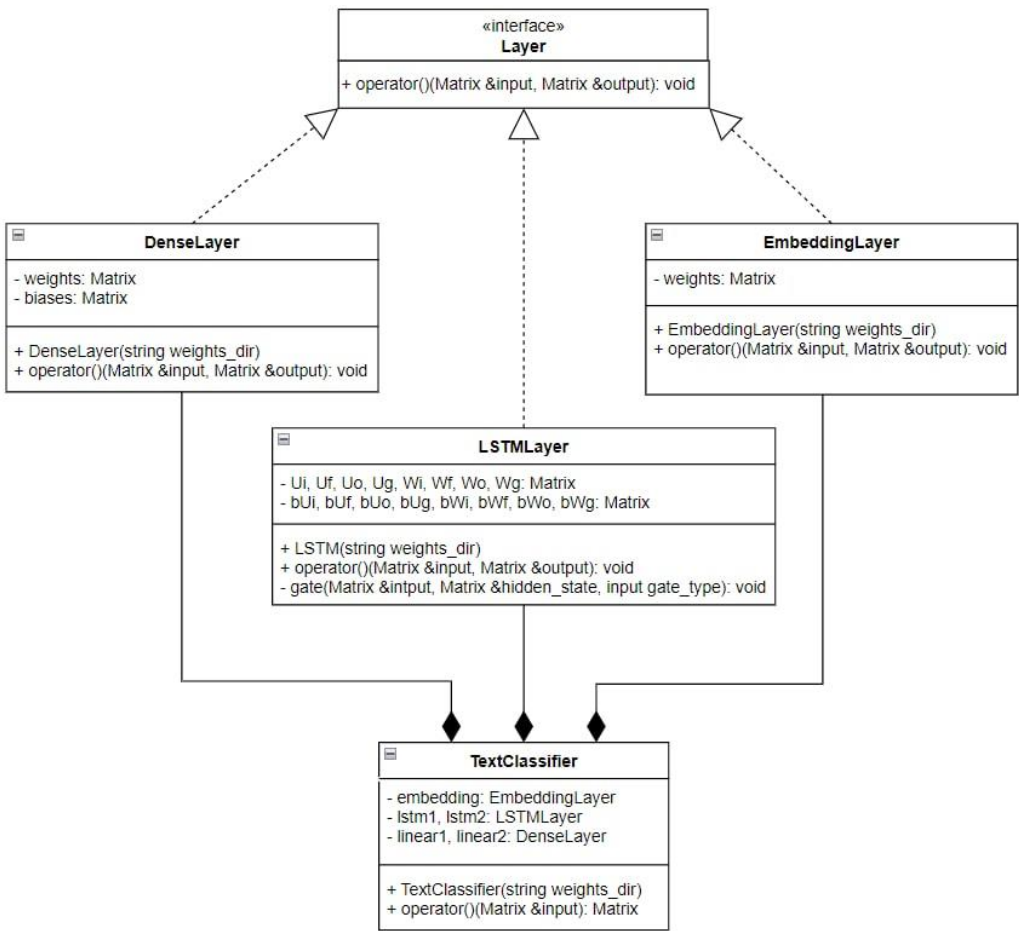


Рисунок 4.1 – UML-діаграма класів хост-програми

Схема розробленої системи виглядає наступним чином (рис. 4.2).

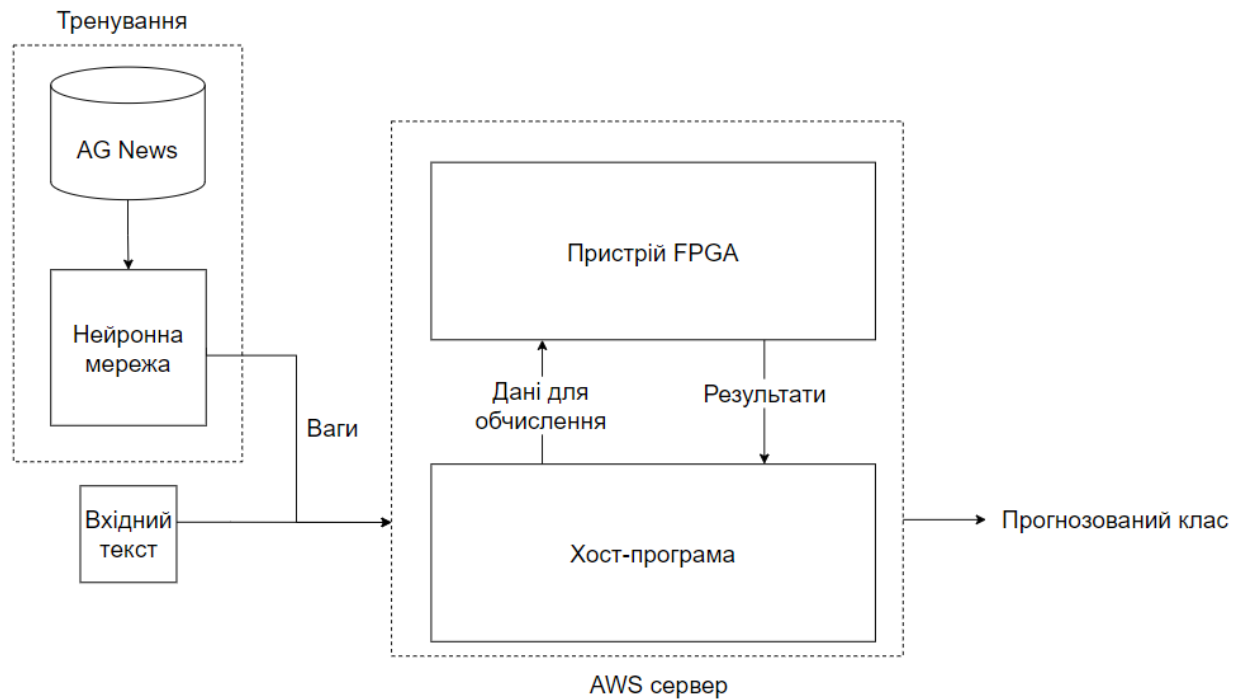


Рисунок 4.2 – Схема розробленої системи

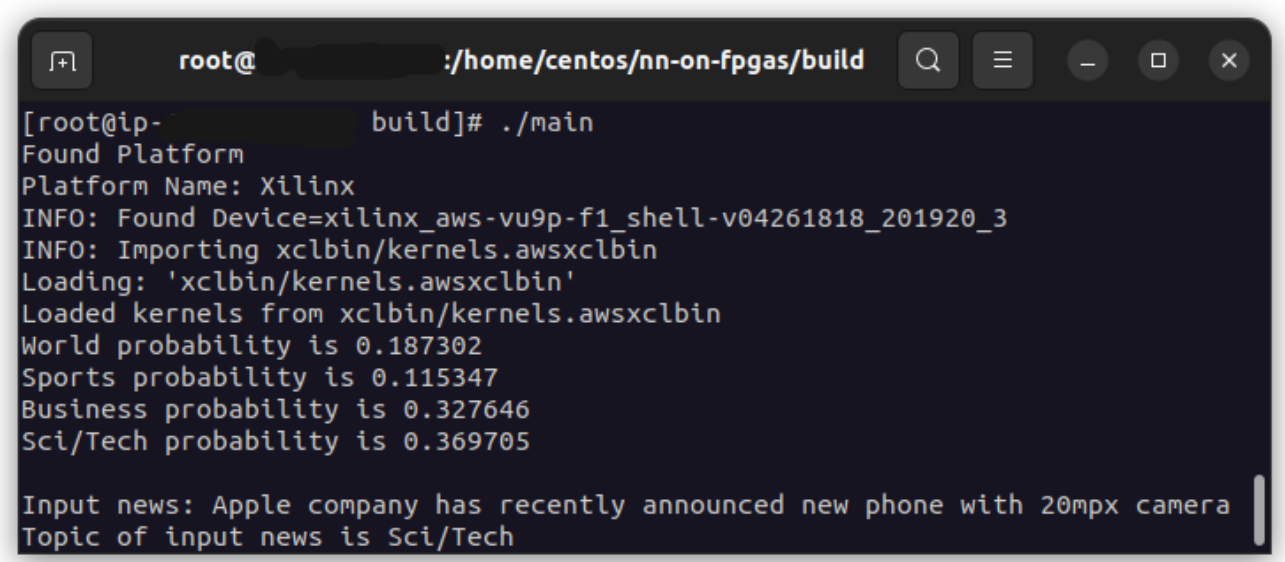
4.3 Висновки за розділом 4

В даному розділі було розроблено ядра, що виконують обчислення на FPGA, описано процес розробки на хмарному сервісі AWS, розроблено хост-програму, що керує обчисленнями на ядрах та розроблено структуру системи.

5 ТЕСТУВАННЯ РОЗРОБЛЕНОЇ СИСТЕМИ

5.1 Тестування та перевірка результатів

Для тестування розробленої системи було обраний наступний текст для класифікації: «Apple company has recently announced new phone with 20mpx camera». На рис. 5.1 наведено приклад класифікації даного тексту. Результат класифікації представлений у вигляді ймовірностей відповідних назв категорій обраного тексту. З результатів видно, що текст класифікований як «Sci/Tech».



```
root@ :/home/centos/nn-on-fpgas/build
[root@ip- build]# ./main
Found Platform
Platform Name: Xilinx
INFO: Found Device=xilinx_aws-vu9p-f1_shell-v04261818_201920_3
INFO: Importing xclbin/kernels.awsxclbin
Loading: 'xclbin/kernels.awsxclbin'
Loaded kernels from xclbin/kernels.awsxclbin
World probability is 0.187302
Sports probability is 0.115347
Business probability is 0.327646
Sci/Tech probability is 0.369705

Input news: Apple company has recently announced new phone with 20mpx camera
Topic of input news is Sci/Tech
```

Рисунок 5.1 – Приклад класифікації тексту

На рис. 5.2 наведена діаграма, на якій представлено час роботи класифікатора текстів на основі нейронної мережі на різних обчислювальних пристроях, а саме на CPU Intel Core i7, GPU Nvidia A10G та FPGA Xilinx UltraScale+ FPGA. З діаграми видно, що класифікація текстів на GPU займає менше всього часу, коли як на CPU – більше. Таким чином, FPGA стає компромісом між CPU та GPU.

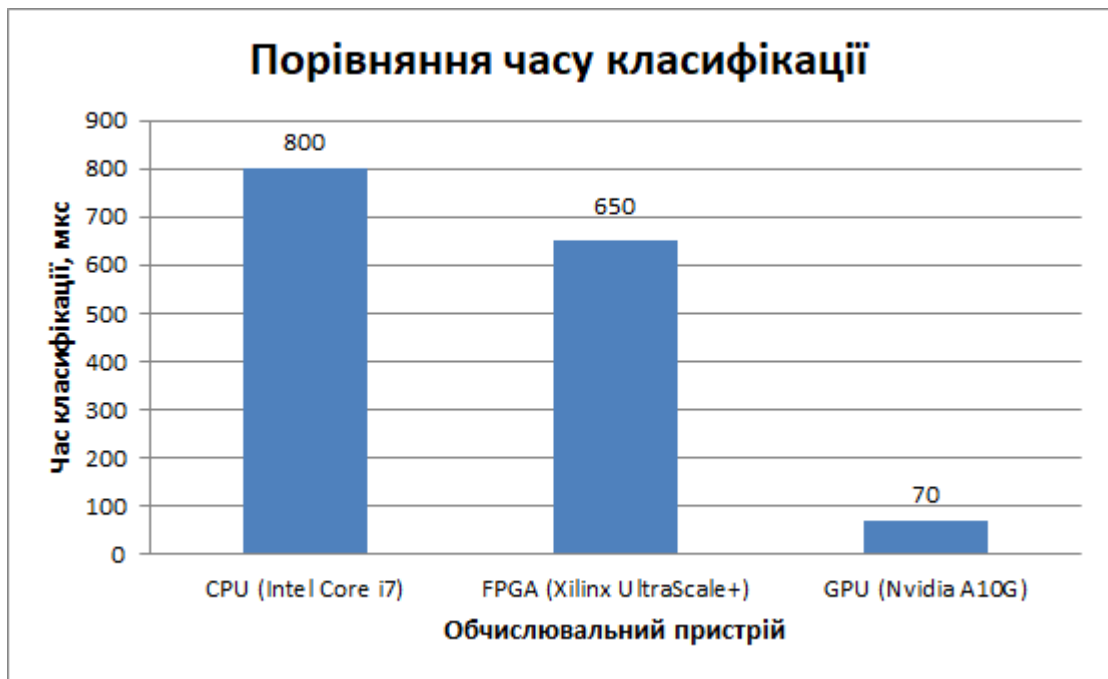


Рисунок 5.2 – Порівняння часу роботи класифікатора

5.2 Висновки за розділом 5

У даному розділі було проведено тестування класифікатора текстів на основі нейронної мережі. Результати роботи класифікатора текстів з прискорювачем на FPGA були порівняні з результатами роботи класифікаторів на CPU та GPU. Виявлено, що розроблений класифікатор поступається у швидкодії класифікатору на графічному процесорі, але працює швидше, порівняно з класифікатором на процесорі загального призначення.

6 КЕРІВНИЦТВО ПРОГРАМІСТА

6.1 Вимоги до технічних та програмних засобів

Для коректної роботи системи програмно-апаратної класифікації текстів на основі нейронних мереж на ПК має бути встановлене наступне програмне забезпечення:

- операційна система Linux;
- Python версії не нижче 3.9;
- бібліотека PyTorch;
- драйвера CUDA.

ПК, на якому виконується навчання нейронної мережі мусить мати наступні характеристики:

- 8 ГБ оперативної пам'яті;
- відеокарта з 16 ГБ пам'яті;
- 1 ГБ вільного місця на диску;
- процесор з тактовою частотою 4 ГГц.

На віртуальній машині віддаленого сервісу AWS, на якій відбувається апаратне прискорення класифікатора текстів, має бути встановлене наступне програмне забезпечення:

- операційна система Linux;
- фреймворк OpenCL;
- Xilinx Runtime Library;
- система керування версіями git;
- Xilinx Vivado.

6.2 Налаштування робочого середовища

Перед початком роботи з FPGA, яку надає сервіс AWS, необхідно провести налаштування робочого середовища, а саме віртуальної машини. Налаштування відбувається у наступній послідовності кроків.

Крок 1. Збірка хост-програми та бінарних файлів, що будуть виконуватися на FPGA;

Крок 2. Компіляція та валідація розроблених ядер за допомогою програмної та апаратної емуляції;

Крок 3. Створення образу AFI.

Після виконання даних кроків можна переходити до запуску інстансу FPGA та запуску хост-програми.

6.3 Висновки за розділом 6

У даному розділі було визначено вимоги до технічних та програмних засобів, що необхідні для запуску та розробки програмної та апаратної складових класифікатора та описано процес налаштування робочого середовища.

ВИСНОВКИ

У результаті виконання роботи проведено аналіз предметної області, розглянуто методи апаратного прискорення нейронних мереж, а також проаналізовано і порівняно нейромережеві методи класифікації текстів.

Розроблено декілька різних архітектур нейронних мереж для класифікації тексту, та за результатами тестування навчених мереж обрано найкращу модель.

За допомогою фреймворка OpenCL створено ядра основних матричних операцій. Створені ядра синтезовано під виконання на FPGA. На основі архітектури найкращої моделі розроблено програму, що виконує класифікацію текстів на FPGA.

Наукова новизна роботи полягає в тому, що розроблено новий засіб програмно-апаратного прискорення процесу класифікації текстів із використанням нейронних мереж, який дозволяє зменшити час класифікації порівняно з процесором загального призначення.

Практична цінність роботи полягає у тому, що розроблена система може бути адаптована до використання з різними типами нейронних мереж.

Також визначено наступні кроки для поліпшення якості класифікації текстів і прискорення нейронної мережі на FPGA:

- проведення експериментів з іншими архітектурами нейронних мереж;
- оптимізація програмного коду під нові архітектури нейронних мереж;
- оптимізація алгоритмів ядер;
- покращення паралельного виконання декількох матричних операцій, де це можливо.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Lane, H. Natural Language Processing in Action [Text] / H. Lane, C. Howard, H. Hapke. – Shelter Island : Manning, 2019. – 544 p.
2. Liddy, E. D. Natural Language Processing Elizabeth / E. D. Liddy. // Encyclopedia of Library and Information Science, Second Edition / M. A. Drake. – New York : Marcel Dekker, Inc, 2003. – Vol. 3. P. 2126-2136.
3. Allen, J. Natural Language Understanding-Pearson 2nd Edition [Text] / J. Allen – London : Pearson, 1994. – 672 p.
4. Khurana, D. Natural Language Processing: State of The Art, Current Trends and Challenges [Electronic recourse] / [D. Khurana, A. Koli, K. Khatter et al.] – Access mode : <https://arxiv.org/ftp/arxiv/papers/1708/1708.05148.pdf>.
5. Reiter, E. Building Natural Language Generation Systems [Text] / E. Reiter, R. Dale. – Cambridge : Cambridge University Press, 2000. – 248 p.
6. Jackson, P. Natural Language Processing for Online Applications Peter Jackson [Text] / P. Jackson, I. Moulinier. – Amsterdam : John Benjamins Publishing Company, 2007. – 232 p.
7. T. Golub, I. Zeleneva, and S. Hrushko, “Software method of preparing text data for their hardware processing using FPGA,” in Science, engineering and technology: global trends, problems and solutions, 2020, pp. 16-20.
8. Indurkha, N. Handbook of Natural Language Processing 2nd Edition [Text] / N. Indurkha, F. J. Damerau. – Boca Raton : CRC Press, 2010. – 702 p.
9. Halteren, H. Syntactic Wordclass Tagging [Text] / H. Halteren. – Dordrecht : Springer Dordrecht, 2010. – 351 p.
10. Paice, C. D. Another Stemmer [Electronic recourse] / C. D. Paice. – Access mode : <https://dl.acm.org/doi/pdf/10.1145/101306.101310>.
11. Aggarwal, C. C. A Survey of Text Classification Algorithms / C. C. Aggarwal, C. X. Zhai // Mining Text Data / C. C. Aggarwal, C. X. Zhai. – New York : Springer, 2012. – Chapter 6. P. 163-222.

12. Kumbhar, P. Survey on Feature Selection Techniques and Classification Algorithms for Efficient Text Classification [Text] / P. Kumbhar, M. Mali // International Journal of Science and Research (IJSR). – 2016. – Vol. 5, Issue 5. – P. 1267-1275.
13. Eisenstein, J. Introduction to Natural Language Processing [Text] / J. Eisenstein. – Cambridge (Massachusetts): MIT Press, 2019. – 536 p.
14. Srinivasa-Desikan, B. Natural Language Processing and Computational Linguistics [Text] / B. Srinivasa-Desikan. – Birmingham: Packt, 2018. – 306 p.
15. Mitkov, R. The Oxford Handbook of Computational Linguistics [Text] / R. Mitkov. – Oxford : Oxford University Press, 2005. 806 p.
16. Joachims, T. Learning to Classify Text Using Support Vector Machines [Text] / T. Joachims. – New York : Springer, 2002. – 222p.
17. Kalaivani, P. An Improved k-Nearest Neighbor Classification Using Genetic Algorithm [Text] / P. Kalaivani, K. L. Shunmuganathan // Circuit, Power and Computing Technologies : 2014 International Conference, 20-21 March 2014 : proceedings. – Nagercoil : IEEE, 2014. – P. 1647-1651.
18. Vijayan, V. K. A Comprehensive Study of Text Classification Algorithms [Text] / V. K. Vijayan, K. R. Bindu, L. Parameswaran // Advances in Computing, Communications and Informatics (ICACCI) : 2017 International Conference, 13-16 September 2017 : proceedings. – Udupi : IEEE, 2017. – P. 1109-1113.
19. Yegnanarayana, B. Artificial Neural Networks [Text] / B. Yegnanarayana. – New Delhi : Prentice-Hall of India Pvt.Ltd, 2004. – P. 476.
20. Haykin, S. Neural Networks and Learning Machines 3rd Edition [Text] / S Haykin. – Upper Saddle River : Prentice Hall, 2008. – P. 936.
21. Aggarwal, C. C. Neural Networks and Deep Learning : A Textbook [Text] / C. C. Aggarwal. – Cham : Springer Cham, 2018. – P. 520.
22. Goodfellow, I. Deep Learning [Text] / I. Goodfellow, Y. Bengio, A. Courville. – Cambridge, Massachusetts : MIT Press, 2016 – 800 p.

23. Chollet, F. Deep learning with Python [Text] / F. Chollet. – Shelter Island : Manning Publications Co., 2018. – P. 504.
24. Y. Shendrikov, D. Tabunshchik, S. Subbotin, P. Arras, and G. Tabunshchik, “Implementation of Language Processing Tools for the University Quality System,” in IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), 2021, P. 705-708.
25. O'Shea, K. An introduction to convolutional neural networks [Electronic resource] / K. O'Shea, R. Nash. – Access mode : <https://arxiv.org/pdf/1511.08458.pdf>.
26. Kuon, I. FPGA Architecture: Survey and [Text] / I. Kuon, R. Tessier, J. Rose // Foundations and Trends® in Electronic Design Automation. – 2008. – Vol. 2, No. 2. – P. 135-253.
27. Cyclone IV Device Handbook Volume 1. San Jose : Altera, 2016. – P. 490.
28. Teubner, J. Data Processing on FPGAs [Text] / J. Teubner, L. Woods. – San Rafael : Morgan & Claypool Publisher, 2013. – P. 118.
29. Amano, H. Principles and Structures of FPGAs [Text] / H. Amano. – Singapore : Springer Singapore, 2018. – P. 231.
30. I. Zeleneva, T. Golub, T. Diachuk, and A. Didenko, “Pipeline model of real number adder on FPGA” Electrical and computer systems, vol. 32(108), 2020, P. 21-31.
31. Dally, W. J. Domain-Specific Hardware Accelerators [Text] / W. J. Dally, Y. Turakhia, S. Han // Communications of the ACM. – 2020. – Vol. 63, No. 7. – P. 48-57.
32. Skliarova, I. FPGA-BASED Hardware Accelerators [Text] / I. Skliarova, V. Sklyarov. – Cham : Springer Cham, 2019. – P. 245.
33. Coussy, P. An Introduction to High-Level Synthesis / P. Coussy, D. D. Gajski, M. Meredith, A. Takach [Text] // IEEE Design & Test of Computers. – 2009. – Vol. 26, Issue 4. – P. 8-17.

34. Meeus, W. An overview of today's high-level synthesis tools [Text] / W. Meeus, K. Beeck, T. Goedeme, J. Meel, D. Stroobandt // Design Automation for Embedded Systems. – 2012. – Vol. 16, Issue 3. – P. 31-51.
35. Lahti, S. Are We There Yet? A Study on the State of High-level Synthesis [Text] / S. Lahti, J. Vanne, P. Sjovall, T. D. Hamalainen // IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. – 2019. – Vol. 38, Issue 5. – P. 898-911.
36. A Survey and Evaluation of FPGA High-Level Synthesis Tools [Text] / [R. Nane, V. M. Sima, C. Pilato et al.] // IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. – 2016. – Vol. 35, Issue 10. – P. 1591-1604.
37. Accelerate applications using Amazon EC2 F1 FPGA instances [Electronic resource]. – Access mode : https://d1.awsstatic.com/events/reinvent/2019/Accelerate_applications_using_Ama_zon_EC2_F1_FPGA_instances_CMP314.pdf.
38. Munshi, A. OpenCL Programming Guide [Text] / A. Munshi, B. Gaster, T. G. Mattson, D. Ginsburg. – Boston: Pearson Education, 2011. – 648 p.
39. Gaster, B. Heterogeneous computing with openCL: revised openCL 1.2 [Text] / B. Gaster, L. Howes, D. R. Kaeli, P. Mistry, D. Schaa. – Waltham MA: Morgan Kaufmann, 2013. – 308 p.
40. Muslim, F. B. Efficient FPGA Implementation of OpenCL High-Performance Computing Applications via High-Level Synthesis [Text] / F. B. Muslim, L. Ma, M. Roozmeh, L. Lavagno // IEEE Access. – 2017. – Vol. 5. – P. 2747-2762.
41. Dmitrijev, O. Essential characteristics of Java [Electronic resource] / O. Dmitrijev. – Access mode : http://www.tud.ttu.ee/im/Vladimir.Viies/materials/IAS0450_Prog.lang.anal/progLangAnal20/algkeeled/Essay_by121998_IASM.pdf.

42. Srinath, K. R. Python – the fastest growing programming language [Text] / K. R. Srinath // International Research Journal of Engineering and Technology. – 2017. – Vol 4, Issue 12. – P. 354-357.
43. Stroustrup, B. An overview of C++ [Text] / B. Stroustrup // SIGPLAN Notices. – 1986. – Vol. 21, Issue 10. – P. 7-18.
44. Sarkar, D. The Python Machine Learning Ecosystem [Text] / D. Sarkar, R. Bali, T. Sharma // Practical Machine Learning with Python.– Berkeley, CA: Apress, 2018. – P. 67-118.
45. Pang, B. Deep learning with tensorflow: A review [Text] / B. Pang, E. Nijkamp, Y. N. Wu // Journal of Educational and Behavioral Statistics. – 2020. – Vol. 45, Issue 2. P. 227-248.
46. Stevens, E. Deep Learning with PyTorch [Text] / E. Stevens, L. Antiga, T. Viehmann. – Shelter Island, NY: Manning Publications Co., 2020. – 520 p.
47. ONNX [Electronic resource]. – Access mode : <https://github.com/onnx/onnx>.
48. Stancin, I. An overview and comparison of free Python libraries for data mining and big data analysis [Text] / I. Stančin, A. Jović // Information and Communication Technology, Electronics and Microelectronics (MIPRO) : 42nd International Convention, Opatija, 20-24 May : proceedings. – Los Alamitos: IEEE, 2019. – P. 977-982.
49. Vanishing gradient problem [Electronic resource]. – Access mode : <https://www.engati.com/glossary/vanishing-gradient-problem>
50. Hochreiter, S. Long Short-Term Memory [Electronic resource] / S. Hochreiter, J. Schmidhuber. – Access mode: <http://www.bioinf.jku.at/publications/older/2604.pdf>.
51. Understanding LSTM Networks [Electronic resource]. – Access mode : <https://colah.github.io/posts/2015-08-Understanding-LSTMs>.

52. Maas, A. L. Learning Word Vectors for Sentiment Analysis [Electronic resource] / [A. L. Maas, R. E. Daly, P. T. Pham et al.] –Access mode : <https://aclanthology.org/P11-1015>.

53. Gilbert, O. Hate Speech Dataset from a White Supremacy Forum [Electronic resource]/ [O. Gilbert, N. Perez, A. Garcia-Pablos et al.] – Access mode : <https://arxiv.org/pdf/1809.04444v1.pdf>.

54. AG's corpus of news articles [Electornic resource]. – Access mode : http://groups.di.unipi.it/~gulli/AG_corpus_of_news_articles.html.

55. Examples for the "From Scratch: Neural Network inference on FPGAs" series [Electronic resource]. – Access mode : <https://github.com/dsuess/nn-on-fpgas/tree/part-1-v1>.

ДОДАТОК А
Технічне завдання

A.1 Вступ

У даному розділі описано область застосування, призначення і вимоги до програми, що розроблюється.

A.1.1 Назва системи

Система називається «Text Classifier».

A 1.2 Призначення та область застосування

Система призначена для класифікації текстів за допомогою нейронних мереж.

Система складається з програмної та апаратної складових. Апаратна складова прискорює роботу нейронної мережі засобами мікросхеми FPGA.

A.2 Підстави для розробки

Підставою для розробки є завдання на диплому роботу на тему «Дослідження та реалізація програмної та апаратної складових класифікатора текста на основі нейронної мережі».

Актуальність роботи полягає в потребі швидких методах класифікації текстів на основі нейронних мереж.

A.3 Призначення розробки

Система надає можливості класифікації текстів шляхом використання штучних нейронних мереж та прискорення обчислень на FPGA.

A.4 Вимоги до системи

В даному розділі описані вимоги до системи.

A.4.1 Вимоги до функціональних характеристик

Система повинна забезпечувати виконання наступних функцій:

- можливість вибору тексту для класифікації;
- можливість виводу результатів класифікації;
- програма апаратного прискорювача повинна виконуватись на віддаленому сервісі AWS.

A.4.2 Вимоги до надійності

Апаратний прискорювач системи призначений для виконання на віддаленому сервісі і має наступні вимоги до надійності:

- забезпечення безперебійної роботи програми;
- захист від помилкових дій користувача.

A.4.3 Вимоги до технічних та програмних засобів

До комп'ютера, на якому буде виконуватись програма, висуваються наступні вимоги:

- 8 ГБ оперативної пам'яті;
- відеокарта з 16 ГБ пам'яті;
- 1 ГБ вільного місця на диску;
- процесор з тактовою частотою 4 ГГц;
- операційна система Linux;
- встановлена бібліотека PyTorch;

- встановлені драйвера CUDA;
- наявність файлів створених нейронних мереж на комп'ютері.

До плати FPGA, на якій працює прискорювач, висуваються наступні вимоги:

- можливість запуску синтезованого коду з OpenCL;
- наявність DSP блоків.

A.5 Вимоги до програмної документації

Склад програмної документації до системи:

- технічне завдання;
- опис розробки програми;
- опис експлуатації програми.

A.6 Стадії і етапи розробки

Розробка системи має бути проведена в наступні стадії:

- створення та узгодження технічного завдання;
- аналіз методів збільшення роздільної здатності термальних і звичайних зображень;
- пошук і обробка датасету для навчання моделей;
- створення архітектури і навчання моделей штучних нейронних мереж для збільшення роздільної здатності термальних зображень;
- аналіз результатів роботи навчених моделей штучних нейронних мереж;
- створення програмного за стосунку для взаємодії користувача зі створеними моделями;
- створення супровідної документації.

A.7 Порядок контролю та приймання

Програмного продукту та документації необхідно пройти наступний порядок контролю та приймання:

- аналіз роботи створених та навчених моделей штучних нейронних мереж;
- тестування програми;
- затвердження працездатності програмного продукту та супровідної документації науковим керівником;
- нормоконтроль документації до програмного продукту.

ДОДАТОК Б

Текст програми

Б.1 Код аппаратного прискорювача

Код файлу main.cpp

```
#include <string.h>
#include <sys/time.h>
#include <CL/cl2.hpp>
#include <algorithm>
#include <iostream>
#include <tuple>
#include "utils.hpp"
#include <filesystem>
#include "net.hpp"
#include <fstream>

#include "matrix.hpp"

int main(int argc, const char *argv[]) {
    init_kernels();

    std::vector<std::string> topics{ "World", "Sports", "Business", "Sci/Tech"
};

    auto model = TextClassifier("../weights/");
    auto input = Matrix::from_npy("../weights/text_inputs.npy");
    input.to_device();
    auto result = model(input);
    result.to_cpu();

    int maxIdx = -1;
    int currMax = -1;
    for (int i = 0; i < result.rows; i++)
    {
        for (int j = 0; j < result.cols; j++)
        {
            std::cout << topics[j] << " probability is " << result(i, j) <<
std::endl;
            if (currMax < result(i, j))
            {
                maxIdx = j;
                currMax = result(i, j);
            }
        }
        std::cout << std::endl;
    }
    std::fstream file;
    file.open("../weights/input_text.txt", std::ios::in);
    if (file.is_open())
    {
        std::string text;
        getline(file, text);
        file.close();
        std::cout << "Input news: " << text << std::endl;
    }
    std::cout << "Topic of input news is " << topics[maxIdx] << std::endl;
    return 0;
}
```

Код файла net.hpp

```
#include <CL/cl2.hpp>
#include <vector>
#include "matrix.hpp"
#include "utils.hpp"
#include "xcl2.hpp"
#include <iostream>
#include <string.h>
#include "layers.hpp"

class TextClassifier
{
private:
    EmbeddingLayer embedding;
    DenseLayer linear1, linear2;
    LSTMLayer lstm1, lstm2;

public:
    TextClassifier(std::string weights_dir)
    {
        embedding = EmbeddingLayer(weights_dir + "/embeddings");
        linear1 = DenseLayer(weights_dir + "/linear1");
        linear2 = DenseLayer(weights_dir + "/linear2");
        lstm1 = LSTMLayer(weights_dir + "/lstm1");
        lstm2 = LSTMLayer(weights_dir + "/lstm2");
    }

    Matrix operator()(Matrix &input)
    {
        Matrix embedded_input;
        embedding(input, embedded_input);

        std::vector<Matrix> lstm1_hidden_outputs(1);
        lstm1(embedded_input, lstm1_hidden_outputs);

        std::vector<Matrix> lstm2_hidden_outputs(1);
        lstm2(lstm1_hidden_outputs, lstm2_hidden_outputs);

        Matrix l1out;
        linear1(lstm2_hidden_outputs.back(), l1out);
        Matrix l1act = activation(l1out, RELU_KERNEL);

        Matrix l2out;
        linear2(l1act, l2out);

        Matrix l2act = activation(l2out, SOFTMAX_KERNEL);
        l2act.to_cpu();
        return l2act;
    }
};
```

Код файла layers.hpp

```
#include <iostream>
#include <vector>
#include "xcl2.hpp"
#include "matrix.hpp"
#include <CL/cl2.hpp>
#include <string.h>
```

```

class Layer:
{
public:
    virtual void operator()(Matrix &input, Matrix &output) = 0;
}

class DenseLayer : public Layer
{
private:
    Matrix weights, biases;

public:
    DenseLayer()
    {

    }

    DenseLayer(std::string layer_weights_dir)
    {
        weights = Matrix::from_npy(layer_weights_dir + "/weights.npy");
        weights.to_device();

        biases = Matrix::from_npy(layer_weights_dir + "/biases.npy");
        biases.to_device();
    }

    void operator()(Matrix &input, Matrix &output)
    {
        output = matmul(input, weights, MATMUL_KERNEL);
        output = add_bias(output, biases, BIAS_KERNEL);
    }
};

class EmbeddingLayer : public Layer
{
private:
    Matrix weights;

public:
    EmbeddingLayer()
    {

    }

    EmbeddingLayer(std::string layer_weights_dir)
    {
        weights = Matrix::from_npy(layer_weights_dir + "/embeddings.npy");
        weights.to_device();
    }

    void operator()(Matrix &input, Matrix &output)
    {
        output = Matrix::constant(input.rows, weights.cols, 0.0);
        for (int i = 0; i < output.rows; ++i)
        {
            for (int j = 0; j < output.cols; ++j)
            {
                output[input[i]][j] = weights[input[i]][j];
            }
        }
    }
};

```

```

        }
        output.to_device();
    }
};

class LSTMLayer : public Layer
{
private:
    Matrix Ui, Uf, Uo, Ug, Wi, Wf, Wo, Wg;
    Matrix bUi, bUf, bUo, bUg, bWi, bWf, bWo, bWg;

    Matrix gate(Matrix &input, Matrix &hidden_state, int gate_type)
    {
        Matrix U, W, bU, bW;
        switch (gate_type) {
            case 0:
                U = Ui;
                W = Wi;
                bU = bUi;
                bW = bWi;
                break;
            case 1:
                U = Uf;
                W = Wf;
                bU = bUf;
                bW = bWf;
                break;
            case 2:
                U = Uo;
                W = Wo;
                bU = bUo;
                bW = bWo;
                break;
        }
        Matrix weihted_inputs = matmul(input, U, MATMUL_KERNEL);
        weihted_inputs = add_bias(weihted_inputs, bU, BIAS_KERNEL);

        Matrix weihted_hidden = matmul(hidden_state, W, MATMUL_KERNEL);
        weihted_hidden = add_bias(weihted_hidden, bW, BIAS_KERNEL);

        Matrix gate_output = matop(weihted_inputs, weihted_hidden,
MATSUM_KERNEL);
        gate_output = activation(gate_output, SIGMOID_KERNEL);
        return std::move(gate_output);
    }

public:
    LSTMLayer()
    {

    }

    LSTMLayer(std::string layer_weights_dir)
    {
        Ui = Matrix::from_npy(layer_weights_dir + "/Ui.npy");
        Ui.to_device();

        Uf = Matrix::from_npy(layer_weights_dir + "/Uf.npy");
        Uf.to_device();

        Uo = Matrix::from_npy(layer_weights_dir + "/Uo.npy");
        Uo.to_device();
    }
};

```

```

    Ug = Matrix::from_npy(layer_weights_dir + "/Ug.npy");
    Ug.to_device();

    Wi = Matrix::from_npy(layer_weights_dir + "/Wi.npy");
    Wi.to_device();

    Wf = Matrix::from_npy(layer_weights_dir + "/Wf.npy");
    Wf.to_device();

    Wo = Matrix::from_npy(layer_weights_dir + "/Wo.npy");
    Wo.to_device();

    Wg = Matrix::from_npy(layer_weights_dir + "/Wg.npy");
    Wg.to_device();

    bUi = Matrix::from_npy(layer_weights_dir + "/bUi.npy");
    bUi.to_device();

    bUf = Matrix::from_npy(layer_weights_dir + "/bUf.npy");
    bUf.to_device();

    bUo = Matrix::from_npy(layer_weights_dir + "/bUo.npy");
    bUo.to_device();

    bUg = Matrix::from_npy(layer_weights_dir + "/bUg.npy");
    bUg.to_device();

    bWi = Matrix::from_npy(layer_weights_dir + "/bWi.npy");
    bWi.to_device();

    bWf = Matrix::from_npy(layer_weights_dir + "/bWf.npy");
    bWf.to_device();

    bWo = Matrix::from_npy(layer_weights_dir + "/bWo.npy");
    bWo.to_device();

    bWg = Matrix::from_npy(layer_weights_dir + "/bWg.npy");
    bWg.to_device();
}

void operator()(Matrix &input, std::vector<Matrix> &hidden_states)
{
    std::vector<Matrix> inputs(input.rows);

    for (int i = 0 ; i < input.rows; ++i)
    {
        inputs[i] = copy_row(input, i, COPY_KERNEL);
    }
    this->operator()(inputs, hidden_states);
}

void operator()(std::vector<Matrix> &input, std::vector<Matrix>
&hidden_states)
{
    std::vector<Matrix> cell_states(1);

    hidden_states[0] = Matrix::constant(1, Wi.rows, 0.0);
    hidden_states[0].to_device();

    cell_states[0] = Matrix::constant(1, Wi.rows, 0.0);

```

```

cell_states[0].to_device();

for (int t = 0; t < input.size(); ++t)
{
    Matrix i = gate(input[t], hidden_states[t], 0);
    Matrix f = gate(input[t], hidden_states[t], 1);
    Matrix o = gate(input[t], hidden_states[t], 2);

    Matrix iUg = matmul(input[t], Ug, MATMUL_KERNEL);
    Matrix hWg = matmul(hidden_states[t], Wg, MATMUL_KERNEL);
    Matrix cell_state_hat = matop(iUg, hWg, MATSUM_KERNEL);
    cell_state_hat = activation(cell_state_hat, TANH_KERNEL);

    Matrix fC = matop(f, cell_states[t], MATELPROD_KERNEL);
    Matrix iCh = matop(i, cell_state_hat, MATELPROD_KERNEL);
    Matrix cell_state = matop(fC, iCh, MATSUM_KERNEL);

    Matrix hidden_state = activation(cell_state, TANH_KERNEL);
    hidden_state = matop(o, hidden_state, MATELPROD_KERNEL);

    hidden_state.to_cpu();
    cell_state.to_cpu();
    hidden_states.push_back(Matrix(hidden_state));
    cell_states.push_back(Matrix(cell_state));
    hidden_states.back().to_device();
    cell_states.back().to_device();
}
hidden_states.erase(hidden_states.begin());
return;
}
};

```

Код файла copy_kernel.cpp

```

#include <iostream>
typedef unsigned int uint;

extern "C" void copy_kernel(float *const inputs, const uint row, const uint
cols, float *const outputs)
{
    for (uint c = 0; c < cols; c++)
    {
        const uint idx = cols * row + c;
        outputs[c] = inputs[idx];
    }
}

```

Код файла relu_kernel.cpp

```

#include <iostream>
typedef unsigned int uint;

inline float relu(const float x)
{
    if (x < 0.f)
        return 0.f;
    return x;
}

```

```
extern "C" void relu_kernel(float *const inputs, const uint rows, const uint
cols, float *const outputs)
{
    for (uint r = 0; r < rows; r++)
    {
        for (uint c = 0; c < cols; c++)
        {
            const uint idx = cols * r + c;
            outputs[idx] = relu(inputs[idx]);
        }
    }
}
```

Код файла tanh_kernel.cpp

```
#include <iostream>
typedef unsigned int uint;
#include "hls_math.h"

extern "C" void tanh_kernel(float *const inputs, const uint rows, const uint
cols, float *const outputs)
{
    for (uint r = 0; r < rows; r++)
    {
        for (uint c = 0; c < cols; c++)
        {
            const uint idx = cols * r + c;
            outputs[idx] = tanh(inputs[idx]);
        }
    }
}
```

Код файла sigmoid_kernel.cpp

```
#include <iostream>
typedef unsigned int uint;
#include "hls_math.h"

float sigmoid(float x)
{
    return 1 / (1 + exp(-x));
}

extern "C" void sigmoid_kernel(float *const inputs, const uint rows, const
uint cols, float *const outputs)
{
    for (uint r = 0; r < rows; r++)
    {
        for (uint c = 0; c < cols; c++)
        {
            const uint idx = cols * r + c;
            outputs[idx] = sigmoid(-inputs[idx]);
        }
    }
}
```

Код файла matsum_kernel.cpp

```
#include <iostream>

extern "C" void matsum_kernel(const float *const matrixA, const float *const
matrixB, const uint rows, const uint cols, float *const out)
{
    for (uint r = 0; r < rows; ++r)
    {
        for (uint c = 0; c < cols; ++c)
        {
            const uint idx = cols * r + c;
            out[idx] = matrixA[idx] + matrixB[idx];
        }
    }
}
```

Код файла matelemprod_kernel.cpp

```
#include <iostream>

extern "C" void matelemprod_kernel(const float *const matrixA, const float
*const matrixB, const uint rows, const uint cols, float *const out)
{
    for (uint r = 0; r < rows; ++r)
    {
        for (uint c = 0; c < cols; ++c)
        {
            const uint idx = cols * r + c;
            out[idx] = matrixA[idx] * matrixB[idx];
        }
    }
}
```

Код файла bias_kernel.cpp

```
#include <iostream>
typedef unsigned int uint;

extern "C" void bias_kernel(float *const inputs, const float *const bias,
const uint rows, const uint cols, float *const outputs)
{
    for (uint r = 0; r < rows; r++)
    {
        for (uint c = 0; c < cols; c++)
        {
            const uint idx = cols * r + c;
            outputs[idx] = inputs[idx] + bias[c];
        }
    }
}
```

ДОДАТОК В

Слайди презентації

Національний університет "Запорізька політехніка"
Кафедра програмних засобів

Дослідження та реалізація програмної та апаратної складових класифікатора текста на основі нейронної мережі

Виконав
ст. гр. КНТ-111м

Ар. Є. Діденко

Керівник:
Професор, к.т.н

Г. В. Табунщик

2022

Рисунок В.1 – Слайд №1

Мета, об'єкт, предмет дослідження

Мета роботи — прискорення процесу класифікації текстів з використанням нейронних мереж.

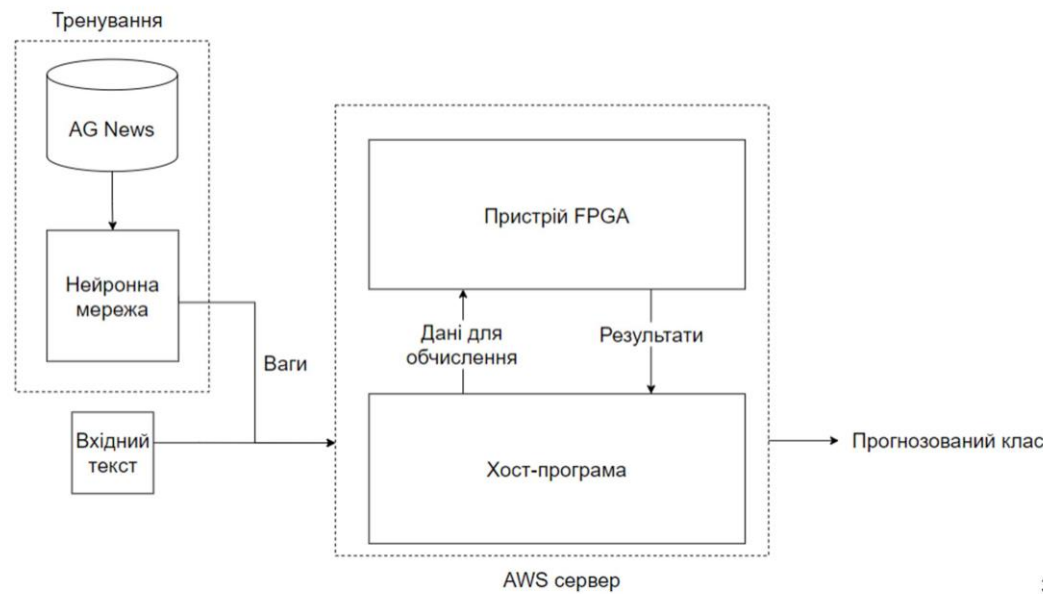
Об'єкт дослідження — програмні та апаратні засоби класифікації текстів.

Предмет дослідження — методи програмно-апаратної класифікації текстів з використанням нейронних мереж.

2

Рисунок В.2 – Слайд №2

Структура розробленої системи



3

Рисунок В.3 – Слайд №3

Вибір нейромережевого методу класифікації

Критерій	Нейромережа прямого поширення	Згоркова нейронна мережа	Рекурентна нейронна мережа
Складність реалізації	Невелика	Середня	Середня
Складність навчання	Невелика	Середня	Достатньо висока
Швидкість навчання	Середня	Середня	Низька
Залежність від розмірності даних	Так	Ні	Ні
Якість класифікації текстів	Низька	Середня	Висока

Відповідно до результатів порівняння обрано **рекурентну нейронну мережу**

4

Рисунок В.4 – Слайд №4

Порівняння платформ для апаратного прискорення

Критерій	CPU	FPGA	GPU
Гнучкість	+	+++	++
Простота	+++	+	++
Тактова частота	+++	+	++
Величина паралелізму	+	+++	++
Вартість	++	+++	+

5

Рисунок В.5 – Слайд №5

Вибір методу розробки на FPGA

Критерій	HLS (OpenCL)	RTL
Рівень абстракції	+++	+
Кількість рядків коду	+++	+
Часові витрати на розробку	+++	+
Рівень оптимізації	++	+++
Наявність вбудованих методів	++	+
Використання апаратних ресурсів	++	+++

Відповідно до результатів порівняння обрано **метод HLS**

6

Рисунок В.6 – Слайд №6

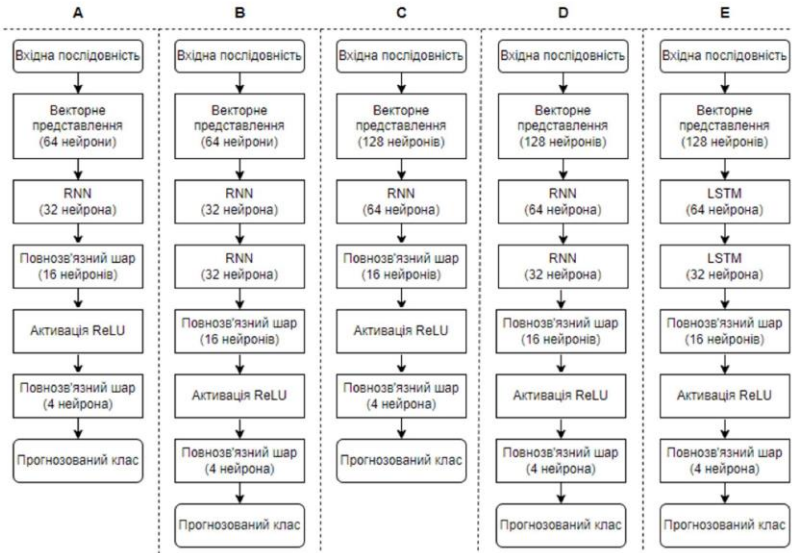
Вибір мови програмування і фреймворку глибинного навчання

Мови програмування				Фреймворки глибинного навчання				
Критерій	Python	C++	Java	Критерій	PyTorch	Tensorflow	Keras	ONNX
Типізація	Динамічна	Статична	Статична	Простота	+++	++	+++	++
Простота	+++	+	++	Гнучкість	+++	+++	++	+
Швидкість	++	+++	++	Підтримка	+++	++	++	+
Кількість модулів і бібліотек	+++	++	+++	Поширеність	+++	++	++	+
Підтримка методів машинного навчання	+++	+	++					
Кількість учбових та довідкових ресурсів	+++	+	++					

Відповідно до результатів порівняння обрано мову програмування Python і фреймворк PyTorch 7

Рисунок В.7 – Слайд №7

Вибір архітектури нейронної мережі



Архітектури різних моделей нейромереж для експериментів

Рисунок В.8 – Слайд №8

Вибір архітектури нейронної мережі

Результати навчання різних моделей нейронних мереж

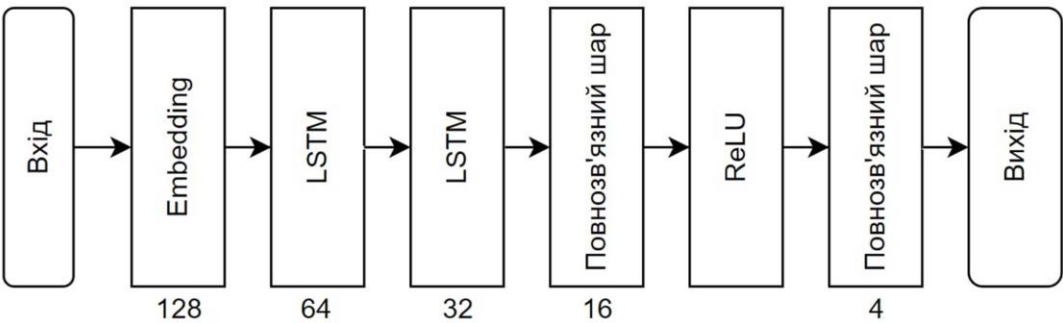
Метрика/Модель	A	B	C	D	E
Влучність	0.82	0.81	0.83	0.87	0.88
Повнота	0.81	0.80	0.83	0.86	0.87
F1	0.82	0.81	0.83	0.86	0.88

За результатами навчання обрано **нейромережу E (LSTM)**

9

Рисунок В.9 – Слайд №9

Архітектура розробленої нейронної мережі



10

Рисунок В.10 – Слайд №10

Розроблені ядра апаратного прискорювача

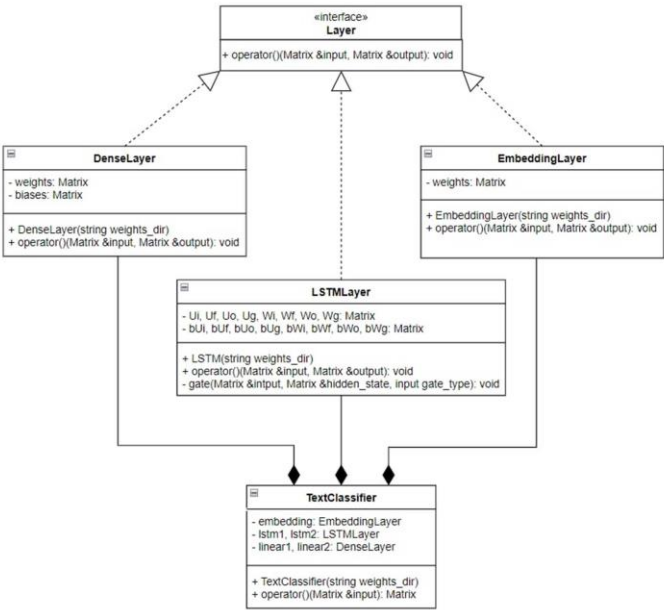
Ядро	Опис
MATMUL_KERNEL	Множення матриць
MATELPROD_KERNEL	Поелементне множення матриць
MATSUM_KERNEL	Сума матриць
BIAS_KERNEL	Додавання зміщення
COPY_KERNEL	Копіювання матриці
RELU_KERNEL	Функція активації ReLU
SIGMOID_KERNEL	Функція активації Sigmoid
TANH_KERNEL	Функція активації Tanh
SOFTMAX_KERNEL	Функція активації Softmax



11

Рисунок В.11 – Слайд №11

Структура розробленої хост-програми



12

Рисунок В.12 – Слайд №12

Діаграма розгортання розробленої системи

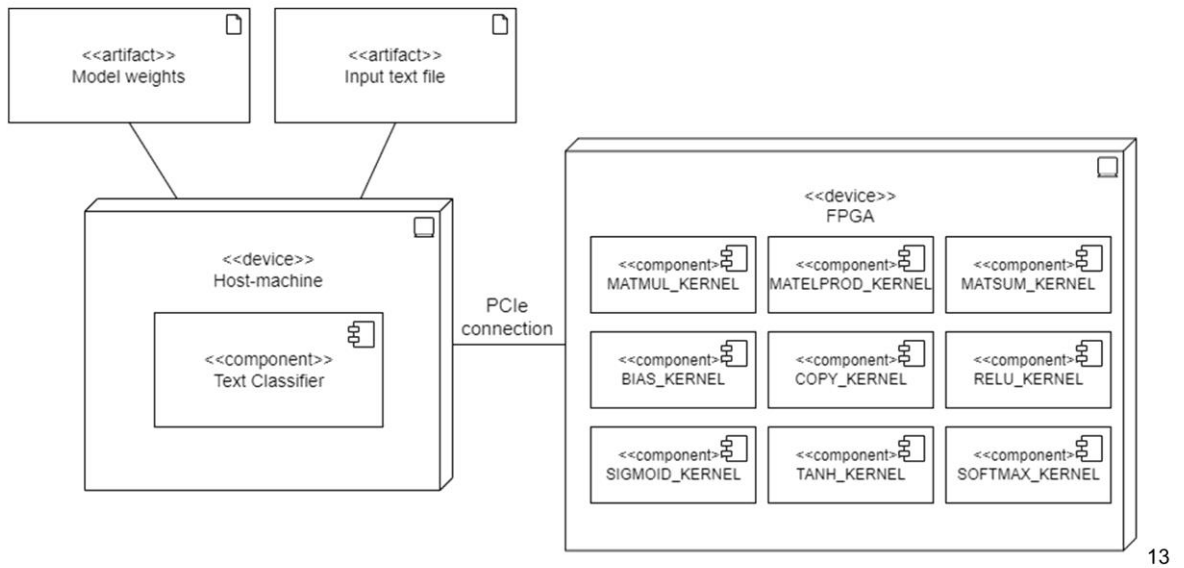


Рисунок В.13 – Слайд №13

Результати запуску нейромережі на FPGA

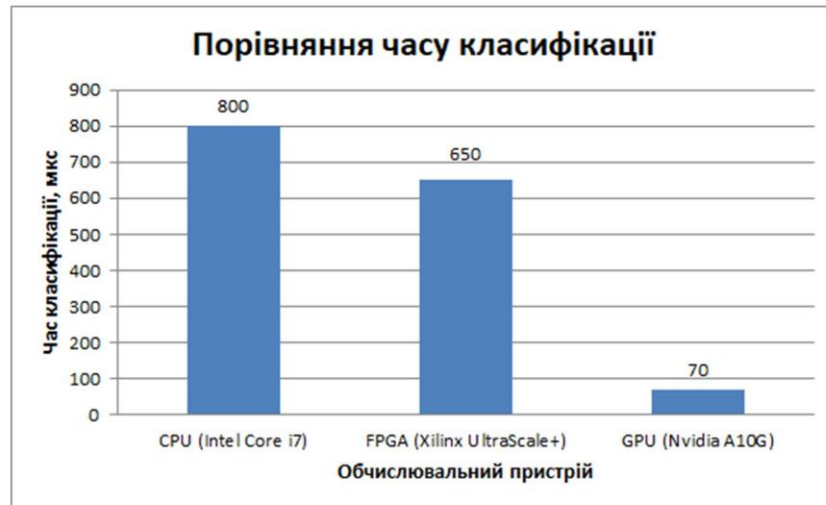
```

root@ip-          :/home/centos/nn-on-fpgas/build
[root@ip-          build]# ./main
Found Platform
Platform Name: Xilinx
INFO: Found Device=xilinx_aws-vu9p-f1_shell-v04261818_201920_3
INFO: Importing xclbin/kernels.awsxclbin
Loading: 'xclbin/kernels.awsxclbin'
Loaded kernels from xclbin/kernels.awsxclbin
World probability is 0.187302
Sports probability is 0.115347
Business probability is 0.327646
Sci/Tech probability is 0.369705

Input news: Apple company has recently announced new phone with 20mpx camera
Topic of input news is Sci/Tech
  
```

Рисунок В.11 – Слайд №14

Тестування розробленої системи



15

Рисунок В.11 – Слайд №15

Наукова новизна і практична цінність

Наукова новизна: розроблено новий засіб програмно-апаратного прискорення процесу класифікації текстів із використанням нейронних мереж, який дозволяє зменшити час класифікації порівняно з процесором загального призначення.

Практична цінність: розроблена система може бути адаптована до використання з різними типами нейронних мереж.

16

Рисунок В.11 – Слайд №16

Висновки

- Розглянуто методи апаратного прискорення нейронних мереж, проаналізовано методи класифікації текстів на основі нейронних мереж.
- Розроблено архітектуру нейронної мережі для класифікації тексту.
- За допомогою фреймворка OpenCL створено ядра основних матричних операцій. Створені ядра синтезовано під виконання на FPGA.
- На основі архітектури найкращої моделі розроблено програму, що виконує класифікацію текстів на FPGA.
- Досягнуто прискорення процесу класифікації тексту.

17

Рисунок В.11 – Слайд №17

Подальші вдосконалення

- проведення експериментів з іншими архітектурами нейронних мереж;
- оптимізація програмного коду під нові архітектури нейронних мереж;
- оптимізація алгоритмів ядер;
- дослідження і використання більш ефективних методів виконання матричних операцій

18

Рисунок В.11 – Слайд №18