Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(назва освітнього ступеня)

на тему: Дослідження застосування методів штучного інтелекту для обробки природної мови засобами глибинного навчання

Виконав: студент VI курсу, групи СНм-61
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)

_____ Климко І.М.
(підпис) (прізвище та ініціали)

Керівник _____ Млинко Б.Б.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Никитюк В.В.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

Рецензент _____ Тиш Є.В.
(підпис) (прізвище та ініціали)

Тернопіль
2024

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
« 24 » грудня 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Магістр
(назва освітнього ступеня)

за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)

Студенту Климко Ігор Михайлович
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження застосування методів штучного інтелекту для обробки природної мови засобами глибинного навчання

Керівник роботи Млинко Богдана Богданівна, к.т.н., доцент кафедри КН
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « 29 » листопада 2024 року № 4/7-1138

2. Термін подання студентом завершеної роботи 24 грудня 2024р.

3. Вихідні дані до роботи Наукові публікації про обробку природньої мови методами глибинного навчання

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1 Аналітична частина. 1.1 Поточний стан в галузі обробки природньої мови.
1.2 Порівняння різних архітектур нейронних мереж для обробки природньої мови.
1.3 Порівняння мовних моделей для генерації і обробки тексту.
1.4 Функціональність розробленої мережі. 2 Теоретична частина.
2.1 Загальний опис напряму роботи. 2.2 Навчання моделей. 2.3 Рекурентні нейронні мережі.
2.4 Архітектура LSTM. 2.5 Обробка природньої мови. 2.6 Роль статистики у задачах NLP.
3. Практична частина. 3.1 Огляд інструментів, використаних в роботі.
3.2 Фреймворк глибинного навчання. 3.3 Реалізація мережі.
4. Охорона праці та безпека в надзвичайних ситуаціях. Висновки. Додатки.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	Сенчишин В.С., доцент		
Безпека в надзвичайних ситуаціях	Клепчик В.М., ст. викладач		

7. Дата видачі завдання 29 листопада 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	29.11.2024	*Виконано*
2.	Підбір наукових джерел про глибинне навчання	29.11.2024-04.12.2024	*Виконано*
3.	Опрацювання наукових публікацій та збір даних по	29.11.2024-04.12.2024	*Виконано*
	темі роботи		
4.	Виконання дослідження згідно мети кваліфікаційної	05.12.2024-09.12.2024	*Виконано*
	роботи		
5.	Оформлення розділу «Аналітична частина»	05.12.2024-09.12.2024	*Виконано*
6.	Оформлення розділу «Теоретична частина»	05.12.2024-09.12.2024	*Виконано*
7.	Оформлення розділу «Практична частина»	05.12.2024-09.12.2024	*Виконано*
8.	Виконання завдання до підрозділу «Охорона праці»	02.12.2024-09.12.2024	*Виконано*
9.	Виконання завдання до підрозділу «Безпека в	02.12.2024-09.12.2024	*Виконано*
	надзвичайних ситуаціях»		
10.	Оформлення кваліфікаційної роботи	10.12.2024-11.12.2024	*Виконано*
11.	Нормоконтроль	12.12.2024-13.12.2024	*Виконано*
12.	Перевірка на плагіат	16.12.2024	*Виконано*
13.	Попередній захист кваліфікаційної роботи	17.12.2024	*Виконано*
14.	Захист кваліфікаційної роботи	26.12.2024	

Студент _____ Климко І.М.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Млинко Б.Б.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дослідження застосування методів штучного інтелекту для обробки природної мови засобами глибинного навчання // Кваліфікаційна робота освітнього рівня «Магістр» // Климко Ігор Михайлович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СНм-61 // Тернопіль, 2024 // С. – 95, рис. – 6, додат. – 3.

Ключові слова: штучний інтелект, глибинне навчання, обробка природньої мови, мовні моделі.

У даній кваліфікаційній роботі досліджується область застосування методів глибинного навчання в обробці природньої мови. Основний акцент роботи спрямований на розробку моделі глибинного навчання, яка буде генерувати текстові дані на основі навчальних. У роботі аналізуються сучасні тенденції в галузі обробки природньої мови та проводиться детальне вивчення методів, які застосовуються в даній області.

Реалізована модель виконує поставлене завдання і здатна генерувати текст, схожий на людський.

ANNOTATION

Research on the Use of Artificial Intelligence Methods for Natural Language Processing Using Deep Learning // Klymko Ihor Mykhailovych // Ternopil Ivan Pulyuy National Technical University, Faculty of Computer Information Systems and Software Engineering, Computer Science Department, SNm-61 group // Ternopil, 2024 // P. – 95, fig. – 6, table – 2, apendix – 3.

Key words: artificial intelligence, deep learning, natural language processing, language models.

This qualification work explores the application of deep learning methods in natural language processing. The main focus of the work is on developing a deep learning model that will generate text data based on training. The work analyzes current trends in the field of vernacular language processing and a detailed study of the methods used in this area.

The implemented model fulfills the task and is able to generate human-like text.

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ І ТЕРМІНІВ

NLP	–	Natural language processing
GRU	–	Gated Recurrent Unit
LSTM	–	Long short-term memory
RNN	–	Recurrent neural networks
BERT	–	Bidirectional Encoder Representations from Transformers
GPT	–	Generative Pre-trained Transformer
T5	–	Text-to-Text Transfer Transformer
BPTT	–	Backpropagation through time
SGD	–	Stochastic gradient descent

ЗМІСТ

ВСТУП

Актуальність теми. Методи обробки природньої мови є тією технологією, яка забезпечує взаємодію між користувачем і комп'ютером з допомогою простого тексту. З кожним роком об'єми текстової інформації тільки зростають, тому способи ефективної її обробки стають надзвичайно важливими, включаючи такі галузі як пошукові системи, машинний переклад, електронну комерцію, аналіз соціальних мереж і інших [1]. Методи NLP покращують процеси автоматизації текстових даних.

Останнім часом було досягнуто значних успіхів в області глибинного навчання, які суттєво вплинули, зокрема, й на сферу обробки природньої мови. Завдяки використанню таких моделей глибинного навчання, як рекурентні нейронні мережі, LSTM і транформери, стало можливим підвищити точність автоматизованої обробки тексту в рази. Саме ці методи дозволяють нам досягати кращих результатів у прикладних задачах, таких як розпізнавання голосу, класифікація тексту, автоматизована генерація тексту, аналіз настрою автора і т.д.

Актуальність даної роботи полягає в тому, що хоч значний успіх в розробці методів NLP і був досягнутий, але в них досі залишається ряд проблем, які заважають їх повноцінному застосуванню. Незважаючи на прогрес у розробці нових методів, багато завдань обробки мови досі залишаються нерозв'язаними, тому ця область потребує подальших досліджень. Ці дослідження включають розробку нових моделей, а також покращення якості підготовки навчальних даних і навчання моделей на їх основі. Крім того, технології обробки мови з кожним днем розвиваються, а областей їх застосування стає тільки більше [2].

Таким чином, дане дослідження застосування методів глибинного навчання для обробки природної мови є актуальним, оскільки сприяє розвитку нових технологій, а також має перспективи в практичному застосуванні.

Мета і задачі дослідження. Метою магістерської кваліфікаційної роботи є розробка моделі глибокого навчання для генерації тексту

Для того, щоб досягти вище вказану мету вирішити такі завдання:

– дослідити предметну область;

– побудувати модель глибинного навчання;

– реалізувати програмну реалізацію моделі для генерації тексту;

– провести тестування розробленої моделі.

Об’єкт дослідження: процес обробки природньої мови.

Предмет дослідження: методи глибинного навчання для процесу обробки природньої мови на прикладі генерації текстів.

Наукова новизна одержаних результатів кваліфікаційної роботи полягає у тому, що отримали подальший розвиток методи обробки природньої мови на основі глибинного навчання.

Практичне значення одержаних результатів. Практичне значення одержаних результатів полягає у розробці моделі глибинного навчання, яка при подальших удосконаленнях може бути використана в різноманітних галузях, таких як чат-боти чи засоби машинного перекладу.

Апробація результатів магістерської роботи. Основні результати проведених досліджень обговорювались на XII науково-технічну конференцію «Інформаційні моделі, системи та технології» та XIII міжнародну науково-технічну конференцію молодих учених та студентів «Актуальні задачі сучасних технологій», які висвітлюють основні результати дослідження.

Публікації. Основні результати кваліфікаційної роботи опубліковано у двох працях конференції (Див. додатки В).

Структура й обсяг кваліфікаційної роботи. Кваліфікаційна робота складається з вступу, аналітичного, чотирьох розділів, висновку, списку джерел та 3 додатків. Обсяг роботи становить 95 сторінок, з яких 25 сторінок займають

додатки. Загальний обсяг кваліфікаційної роботи складає 70 сторінки, з них 57 сторінок основного тексту, який містить 6 рисунки та 2 таблиці.

1 АНАЛІТИЧНА ЧАСТИНА

1.1 Поточний стан в галузі обробки природньої мови

Щодня в інтернеті створюється величезна кількість текстових матеріалів, обробка яких вручну зайняла б надто багато часу. Для аналізу та інтерпретації таких даних було розроблено новітні алгоритми глибинного навчання, які сьогодні є одними з найперспективніших технологій у сфері штучного інтелекту. Сучасні методи обробки природної мови, що базуються на глибоких нейронних мережах, суттєво підвищують ефективність взаємодії людини з комп'ютером, забезпечуючи високу швидкість і точність виконання різних завдань.

Векторизація слів стала одним із ключових напрямків розвитку NLP, оскільки вона дозволяє перетворювати текстові дані у зручний формат для навчання нейронних мереж. Однією з найвідоміших моделей у цій сфері є Word2Vec, яка створює векторні представлення слів, враховуючи їхній контекст у навчальному корпусі. Ця модель стала основою для подальших розробок, які безпосередньо працюють із текстом, використовуючи векторні представлення [3].

Серед таких моделей особливо виділяються рекурентні нейронні мережі та їхні модифікації, зокрема LSTM. Ці моделі ефективно враховують порядок і контекст слів у тексті, що дозволяє застосовувати їх для автоматизованого перекладу, аналізу та генерації текстів, розпізнавання мовлення тощо.

Однак справжнім проривом у цій галузі стало впровадження моделей-трансформерів, таких як BERT та GPT. Завдяки механізму уваги трансформери здатні аналізувати весь контекст тексту, а не лише його останні частини, як це було в попередніх підходах. Наприклад, модель BERT, яка працює в двонаправленому режимі, забезпечує високу точність у багатьох NLP-завданнях.

Модель GPT-3, одна з найпотужніших генеративних моделей, демонструє виняткові можливості. Вона може писати тексти на різні теми, генерувати код, відповідати на запитання, створювати поезію та навіть брати участь у творчих процесах. Її успіх базується на використанні мільярдів параметрів, що дозволяють враховувати найменші нюанси семантики й структури тексту. Однак масштабування таких моделей викликає певні труднощі, наприклад, високі обчислювальні витрати та необхідність у значних обсягах даних для навчання [4].

Застосування генеративних моделей тексту охоплює різні сфери. У журналістиці вони можуть створювати автоматичні звіти та аналіз подій, у сфері освіти — генерувати навчальні матеріали та відповіді на студентські запитання. У бізнесі ці моделі використовуються для автоматизації контенту, написання електронних листів, створення сценаріїв для чат-ботів тощо. У художній літературі вони допомагають придумувати нові сюжети, персонажів і навіть експериментальні форми мистецтва.

Важливим фактором є якість даних, що використовуються для навчання моделей. Незбалансовані або упереджені дані можуть призводити до створення некоректного чи стереотипного контенту. Тому дослідники активно працюють над покращенням методів попередньої обробки даних і впровадженням механізмів контролю якості текстів.

Ще однією важливою проблемою є етичний аспект. Генерація текстів викликає занепокоєння через можливість зловживання технологією, наприклад, для створення фейкових новин чи маніпулятивного контенту. Для запобігання таким ризикам розробляються інструменти, які дозволяють виявляти машинно-генеровані тексти й забезпечувати прозорість використання цих технологій [5].

Варто також згадати про обмеження генеративних моделей. Вони можуть створювати тексти, які здаються логічними, але містять фактичні помилки або вигадки. Це пояснюється тим, що моделі не розуміють реальний світ, а лише

аналізують статистичні закономірності в даних. Тому їх інтеграція в реальні додатки вимагає ретельного тестування й налаштування [6].

У підсумку, генеративні моделі тексту на основі глибинного навчання є потужним інструментом, що трансформує наше життя. Завдяки розвитку архітектур нейронних мереж і вдосконаленню обчислювальних систем, ця технологія має великий потенціал для подальшого вдосконалення. Однак важливо враховувати технічні, етичні й соціальні виклики, що виникають під час її впровадження.

1.2 Порівняння різних архітектур нейронних мереж для обробки природньої мови

Для генерації природньої мови зазвичай використовують такі моделі глибинного навчання, як RNN, LSTM і трансформери. Вони займають ключові позиції в цій галузі. Кожна з цих архітектур має свої унікальні особливості, переваги та недоліки, які визначають їх ефективність у певних завданнях. У цьому підрозділі розглянуто принципи роботи, особливості та основні відмінності між цими моделями.

1.2.1 Recurrent Neural Networks

RNN є базовою архітектурою для роботи з послідовними даними. Основна ідея полягає у використанні зворотного зв'язку: вихід попереднього кроку подається на вхід наступного. Це дозволяє моделі запам'ятовувати інформацію з попередніх кроків і використовувати її для обробки поточних даних [7].

Основні переваги RNN:

– Простота та ефективність у задачах, де послідовність даних є критичною, таких як обробка тексту, звуку чи часових рядів.

– Можливість обробки послідовностей змінної довжини.

Недоліки:

– Проблема зникнення градієнта. при навчанні на довгих послідовностях градієнт може ставати надто малим, що унеможливлює ефективне оновлення ваг.
– Складність у запам'ятовуванні довготривалої залежності через обмежену здатність до збереження контексту.

1.2.2 Long Short-Term Memory

LSTM була запропонована як розширення RNN для вирішення проблеми зникнення градієнта. Архітектура LSTM включає спеціальні структури, відомі як "комірки пам'яті", які дозволяють вибірково зберігати та видаляти інформацію [8].

Основні переваги:

– Ефективність у роботі з довготривалими залежностями.
– Стійкість до проблеми зникнення градієнта завдяки механізму керування пам'яттю.
– Застосування в широкому спектрі задач, таких як машинний переклад, розпізнавання мови, прогнозування часових рядів.

Недоліки:

– Складність у навчанні через велику кількість гіперпараметрів.

1.2.3 Трансформери

Трансформатори представляють сучасний підхід до обробки послідовних даних і базуються на механізмі самоприділення (self-attention). Дана модель стала основою для багатьох сучасних моделей, таких як BERT, GPT тощо [9].

Основний принцип роботи трансформерів полягає у визначенні важливості кожного елемента вхідної послідовності відносно інших.

Основні переваги:

- Паралельна обробка послідовностей, що значно пришвидшує навчання.
- Висока точність у задачах обробки природної мови та комп'ютерного зору.
- Гнучкість у роботі з довгими послідовностями.

Недоліки:

- Висока обчислювальна складність, особливо при роботі з дуже довгими послідовностями через квадратичну складність обчислення уваги.
- Залежність від великих обсягів даних для ефективного навчання.

У таблиці 1.1 зображено коротке порівняння характеристик даних архітектур.

Таблиця 1.1 – Порівняння моделей мереж

Характеристика	**RNN**	**LSTM**	**Трансформери**
Запам'ятовування	Короткотривале	Довготривале	Контекст усієї послідовності
Проблема зникнення градієнта	Так	Ні	Ні
Складність обчислення	Низька	Середня	Висока
Паралельна обробка	Ні	Ні	Так
Робота з довгими послідовностями	Обмежена	Добра	Відмінна

1.3 Порівняння мовних моделей для генерації і обробки тексту

Мовні є ключовим інструментом сучасного штучного інтелекту, який здатен виконувати широкий спектр різноманітних завдань, таких як переклад, генерація тексту, відповіді на запитання і інше. Розглянемо кілька прикладів комерційних мереж а також порівняємо з ними нашу створену мережу.

1.3.1 GPT

GPT, розроблена компанією OpenAI, є однією з найвідоміших моделей у сфері генерації тексту. Головною характеристикою цієї моделі є її автогресивний підхід: вона передбачає наступне слово в тексті на основі попередніх слів. Архітектура GPT базується виключно на механізмі трансформерів, які використовують механізм уваги для обробки тексту [10].

Основні переваги RNN:

– Завдяки автогресивній природі GPT здатна створювати текст, що виглядає природно.
– GPT використовує уніфіковану архітектуру, що дозволяє її навчати на великих обсягах даних без потреби в спеціальній розмітці.
– Модель підходить для завдань, таких як написання статей, відповіді на запитання, переклад та створення коду.

Недоліки:

– Модель має обмеження на розмір контекстного вікна, що може впливати на довгі послідовності.
– GPT обробляє текст тільки зліва направо, що може впливати на точність у завданнях, де важливий контекст з обох боків.

1.3.2 BERT

BERT, розроблена компанією Google, є революційною моделлю, яка використовує бідирекційний підхід до обробки тексту. Завдяки цьому модель враховує контекст слів як зліва, так і справа, що робить її надзвичайно ефективною для задач аналізу тексту [11].

Основні переваги:

– Ця властивість дозволяє BERT досягати високої точності в завданнях класифікації тексту, вилучення сутностей та аналізу тональності.
– Під час навчання модель навчається прогнозувати приховані слова, що робить її більш універсальною.

Недоліки:

– Через архітектуру, орієнтовану на розуміння тексту, BERT не є оптимальним вибором для задач генерації тексту.
– Навчання та використання BERT вимагає значних ресурсів.

1.3.3 T5 (Text-to-Text Transfer Transformer)

T5, розроблена компанією Google, є універсальною моделлю, яка перетворює будь-яке завдання обробки тексту у формат "текст на вхід - текст на вихід" [12]. Ця модель поєднує переваги генеративного підходу GPT та аналітичного підходу BERT.

Основні переваги:

– Завдяки текстовій формулюванню задач, T5 може виконувати переклад, класифікацію, узагальнення тексту та багато іншого.
– T5 має кілька варіантів архітектури, що дозволяє адаптувати її до різних завдань і ресурсів.

Недоліки:

– Модель потребує ретельного підбору параметрів для досягнення високої продуктивності.
– Як і інші великі моделі, T5 вимагає значних ресурсів для навчання та використання.

У таблиці 1.2 порівняно деякі характеристики цих моделей з створеною в рамках цієї дипломної роботи.

Таблиця 1.2 – Порівняння характеристик даних моделей мереж, а також моделі, представленої в цій роботі.

Модель	**Основна функція**	**Архітектура**	**К-сть навчальних даних**	**К-сть гіперпараметрів**
GPT-3	Генерація тексту	Трансформери (автогресивні)	~800 ГБ тексту	До 175 мільярдів
BERT	Аналіз тексту	Трансформери (бідирекційні)	~16 ГБ тексту	До 340 мільйонів
T5	Генерація та аналіз тексту	Трансформери (текст-текст)	~800 ГБ тексту	До 11 мільярдів
Модель	Генерація тексту	LSTM	~100 КБ тексту	До 2 мільйонів

1.4 Функціональність розробленої мережі

Розроблена нейронна мережа є системою, створеною з нуля на мові програмування Python. Вона базується на фреймворку, розробленому спеціально для цієї мережі. Основною метою даної розробки є створення моделі, здатної генерувати текст, який за своєю структурою та стилем схожий на текст, створений людиною.

Мережа побудована на LSTM-архітектурі, що дозволяє ефективно працювати з послідовними даними та враховувати довготривалі залежності. Вибір LSTM як основи архітектури обумовлений її здатністю вирішувати проблеми, характерні для традиційних рекурентних нейронних мереж (RNN), такі як зникнення або вибух градієнтів під час навчання на довгих послідовностях. Завдяки коміркам пам’яті, LSTM може зберігати важливу інформацію протягом тривалого часу, що є критично важливим для завдань генерації тексту [13].

Фреймворк, що слугує основою для розробленої мережі, реалізує такі ключові компоненти:

- Фреймворк надає низькорівневий контроль над процесом обчислення та оновлення градієнтів, що дозволяє гнучко налаштовувати алгоритми навчання.
- Реалізовано різні методи оптимізації, включаючи стандартні алгоритми, такі як SGD, що забезпечує адаптивний підхід до навчання залежно від специфіки задачі.
- Фреймворк забезпечує можливість розширення за рахунок простого додавання нових компонентів, таких як додаткові шари або активаційні функції.

Основна функціональність мережі зводиться до генерації тексту на основі заданого початкового символу. Кожен подальший символ генерується на основі попередніх.

Було проведено навчання мережі на наборі даних з текстами Шекспіра. Результатом роботи моделі став згенерований текст, який нагадував людську мову, що є непоганим показником для моделей з малим корпусом навчальних даних.

1.5 Висновок до першого розділу

В першому розділі кваліфікаційної роботи:

– Розглянуто предметну область обробки природньої мови;
– Проаналізовано основні алгоритми в даній області;
– Досліджено основні моделі конкурентів;
– Обґрунтовано вибір архітектури для нейронної мережі;
– Сформовано вимоги до розроблюваної моделі.

2 ТЕОРЕТИЧНА ЧАСТИНА

2.1 Загальний опис напряму роботи

Нейронні мережі – це комп’ютерне програмне забезпечення, яке імітує роботу людського мозку. Вони здатні навчатися на величезних масивах даних і виявляти закономірності, які людям важко помітити. Основними складовими нейронних мереж є вузли, які називають штучними нейронами, і з’єднання між ними, що мають певні ваги. Нейронні мережі можна застосовувати для виконання широкого спектра завдань, таких як класифікація, регресія, генерація, розпізнавання об’єктів, обробка природної мови, прогнозування часових рядів тощо [14]. Приклад нейронної мережі зображений на рисунку 2.1.

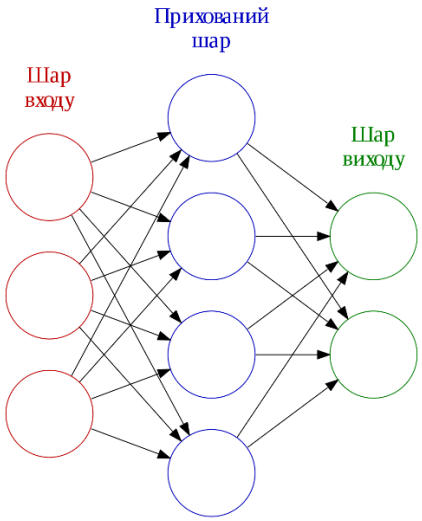

Рисунок 2.1 – Приклад тришарової нейронної мережі

Нейронна мережа навчається виконувати завдання шляхом тренування на прикладах. Ці приклади складаються з вхідних даних та відповідних вихідних даних, які має передбачити мережа. Під час тренування коригуються ваги з’єднань між нейронами для мінімізації похибки між фактичними та очікуваними

результатами. Для цього використовуються алгоритми оптимізації, такі як градієнтний спуск, стохастичний градієнтний спуск, адаптивні алгоритми тощо. Одним з найпоширеніших методів навчання нейронних мереж є зворотне поширення помилки (“backpropagation”), яке включає обчислення градієнтів похибки по відношенню до ваг з’єднань і їх подальше оновлення від останнього шару до першого. Цей процес повторюється до досягнення бажаної точності [15].

Нейронні мережі можуть мати різні архітектури, залежно від типу і складності завдання. Одношарова нейронна мережа має лише один шар нейронів, який безпосередньо з’єднує вхідні дані з вихідними. Багатошарові нейронні мережі включають один або більше прихованих шарів, які виконують додаткову обробку даних, дозволяючи моделювати складні та нелінійні залежності.

Згорткові нейронні мережі використовують спеціальні згорткові шари для аналізу зображень, відео або аудіо. Вони ефективно витягують просторові особливості, наприклад, краєчки, текстури, ієрархії об’єктів. Рекурентні нейронні мережі призначені для роботи з послідовними даними, такими як текст, мова або часові ряди. Особливість RNN полягає у використанні зворотних зв’язків, що дозволяють зберігати інформацію про попередні стани під час обробки послідовності.

Однак стандартні RNN мають обмеження, пов’язані з короткочасною пам’яттю, що ускладнює роботу з довгими послідовностями. Для розв’язання цієї проблеми було розроблено архітектури з покращеною пам’яттю, такі як довга короткочасна пам’ять та GRU. LSTM має спеціальні блоки пам’яті, які дозволяють зберігати та використовувати важливу інформацію протягом довгого часу, що робить її ідеальною для задач обробки природної мови та генерації тексту.

Обробка природної мови є одним з основних напрямків застосування нейронних мереж. Це поле включає такі задачі, як розпізнавання мовлення,

переклад тексту, аналіз тональності, вилучення інформації, генерація тексту тощо. Нейронні мережі дозволили досягти значного прогресу в цих завданнях завдяки своїй здатності моделювати складні залежності між словами, враховувати контекст і працювати з великою кількістю даних. В NLP сигнали мають періодичні зміни, тому в цій галузі є важливими попередні дослідження, спрямовані на аналіз властивостей умовних лінійних процесів в циклічних стаціонарних сигналах [16].

Окремий інтерес становлять архітектури, що використовуються для NLP. Наприклад, RNN і LSTM широко застосовуються для задач, де важливий порядок слів і довготривалий контекст. Водночас, сучасні підходи, такі як трансформери, ще більше розширили можливості нейронних мереж у цій галузі. Трансформери використовують механізм самоуваги, який дозволяє моделі зосереджуватися на найважливіших частинах вхідного тексту, незалежно від їхньої позиції. Це дає змогу ефективно працювати з довгими послідовностями.

Генерація тексту є однією з найбільш вражаючих можливостей нейронних мереж. Наприклад, автогресивні моделі, такі як GPT, здатні створювати текст, який виглядає природно і відповідає заданому контексту. Ці моделі навчаються на величезних масивах текстових даних, щоб передбачати наступне слово в реченні, базуючись на попередніх словах. Завдяки цьому GPT успішно застосовується для написання статей, створення відповідей на запитання, перекладу текстів та навіть написання коду.

Інший підхід до генерації тексту демонструють моделі, такі як T5, які формулюють будь-яку задачу обробки тексту у вигляді “текст на вхід – текст на вихід”. Цей підхід дозволяє використовувати одну архітектуру для різних завдань, таких як узагальнення тексту, переклад або відповіді на запитання.

Мовні моделі – це класи моделей, які прогнозують імовірність появи наступного слова або послідовності слів. Вони є основою багатьох задач NLP. Сучасні мовні моделі, такі як GPT, BERT, T5 та інші, демонструють вражаючі

результати завдяки використанню трансформерів. Моделі типу BERT застосовують бідирекційний підхід, аналізуючи контекст як зліва, так і справа від слова, що забезпечує високу точність у завданнях класифікації та аналізу тексту [17]. Приклад використання мовної моделі наведено на рисунку 2.2.

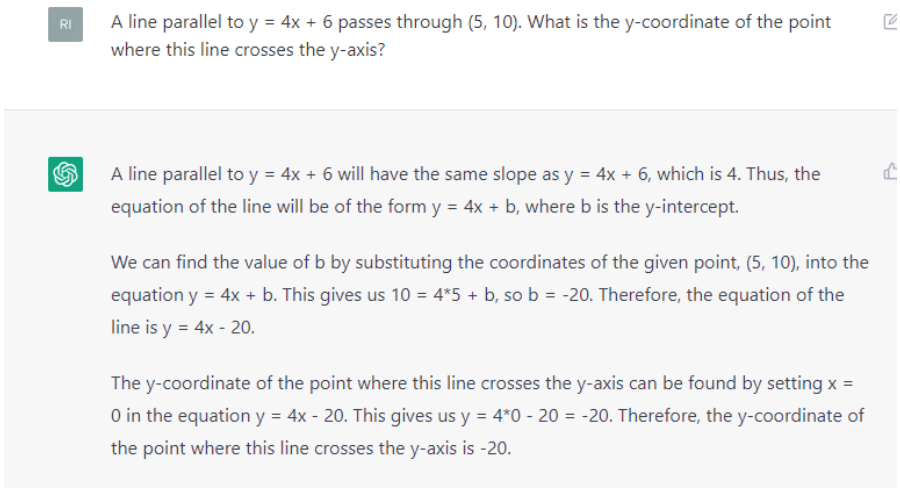

Рисунок 2.2 – Приклад застосування LM

Успіх мовних моделей значною мірою залежить від обсягів навчальних даних та кількості параметрів. GPT-3, наприклад, має 175 мільярдів параметрів і навчалася на текстах, що охоплюють сотні гігабайтів. Це дозволяє їй відповідати на складні запитання, створювати тексти в різних стилях і навіть вирішувати математичні задачі.

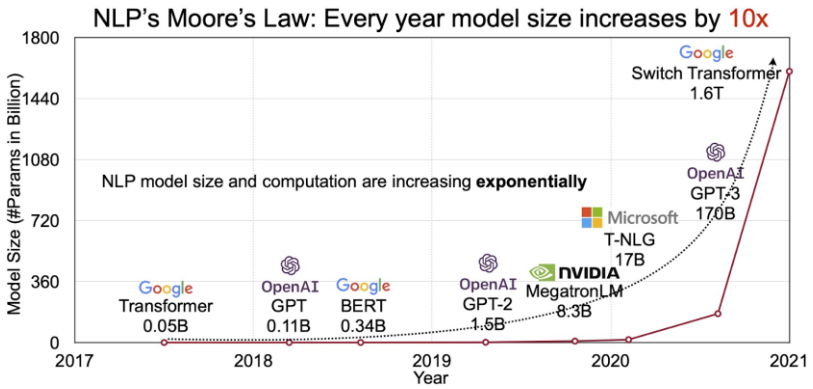

Рисунок 2.3 – Кількість параметрів у популярних LM

На рисунку 2.3 показано діаграму росту кількості параметрів в популярних LM з плином часу.

2.2 Навчання моделей

Навчання — це багатогранний процес, що охоплює безліч аспектів і дозволяє людині адаптуватися до оточуючого середовища. Якщо коротко, навчання можна визначити як формування внутрішньої моделі зовнішнього світу. Ці моделі дозволяють людині розуміти, інтерпретувати та взаємодіяти з навколишнім світом [18].

В нашому мозку зберігається велика кількість таких моделей, які відтворюють певні аспекти зовнішнього світу. До таких моделей відносяться наприклад ментальна модель дому, яка дозволяє безпроблемно орієнтуватися у власній кімнаті, моделі різних частин тіла, які відстежують положення кінцівок і дозволяють контролювати їх. Також є моделі певнх набутих навичок, які, наприклад, дозволяють користуватися ручкою для письма.

Також в нас є велика мовна модель, яка допомагає зрозуміти що слово *тужливий* - українське, *make* – іноземне, а *олпртр* – просто набір незв’язаних символів. Якби на ці три слова поглянув хтось, ненаділений мовною моделлю, то для нього всі б три були простим набором незрозумілих символів.

Всі ці моделі були набуті людьми в процесі навчання, що дозволило краще пристосуватися до життя в умовах навколишнього середовища.

Ми можемо уточнити минуле визначення, сказавши що навчання – це уточнення параметрів ментальної моделі.

Кожна ментальна модель має набір параметрів, які налаштовуються в процесі навчання. Наприклад, модель, що відповідає за координацію рухів, має параметри, які визначають силу, швидкість і точність руху. Ці параметри підлаштовуються залежно від досвіду [19].

Ще у XVII столітті Рене Декарт вказував на необхідність існування в нервовій системі механізмів, які можуть обробляти сенсорну інформацію та трансформувати її у відповідні дії. Сучасні дослідження підтверджують, що такі механізми реалізовані через мережу нейронів, здатних до навчання та адаптації.

Розглянемо простий експеримент. Поставте перед собою предмет і спробуйте взяти його руками. Тепер одягніть короткозорі окуляри й повторіть спробу. Скоріш за все, ваша перша спроба буде невдалою, адже спотворення зору порушує налаштування моделі, яка відповідає за просторову координацію. Проте після кількох спроб мозок адаптується, і рухи стануть точнішими. Зніміть окуляри й повторіть спробу — знову можуть виникнути труднощі, адже система координат змінилася.

Цей експеримент ілюструє, як швидко наш мозок адаптує ментальні моделі через короткий цикл навчання. При цьому ваги нейронних зв'язків, що відповідають за орієнтацію в просторі, були змінені для адаптації до нових умов.

Нейронні мережі, які відповідають за обробку вхідної інформації в нашому мозку, діють за принципом “від простого, до складного”, тобто спочатку нейрони шукають елементарні закономірності, а вже наступні шари нейронів на цих простих закономірностей шукають складніші. Чудовим прикладом цього є обробка мозком зорової інформації: нейрони первинної зорової кори обробляють інформацію з дрібної частини сітківки. Кожен з цих нейронів виявляє маленькі закономірності (наприклад, нахилена лінія), а мільйони інших нейронів роблять те саме для інших областей. Після цього інформація від цих нейронів переходить на інший шар, який виводить закономірності з цих закономірностей. І так далі, з шару на шар. Ця ієрархія дозволяє сканувати все складніші об'єкти: лінія, палець, долоня, рука, людина і т.д [20].

На основі структури кори мозку були створені штучні нейронні мережі - комп'ютерні алгоритми, що складаються з ієрархії перцептронів. На кожному рівні ці нейронні мережі досліджують все складніші закономірності. Глибокі

мережі - це мережі з великою кількістю таких рівнів. Це дозволяє досліджувати найскладніші закономірності, наприклад, розрізнення обличчя конкретної людини серед великої кількості людей на певній вулиці.

Такі мережі навчаються за допомогою навчання з учителем - нейронна мережа, спочатку неточна, видає певний результат на основі вхідних даних. Якщо результат неправильний, відбувається зворотне поширення помилки, і параметри нейронів змінюються. Після багатьох повторень цього процесу ми отримуємо мережу, яка з високою точністю виконує задане завдання.

У штучних нейронних мережах весь процес навчання базується на принципі градієнтного спуску. Градієнтний спуск - це алгоритм оптимізації, який шукає локальний мінімум функції, роблячи малі кроки в напрямку, протилежному градієнту (або нахилу) функції в поточній точці. Цей алгоритм має один недолік - іноді мережа потрапляє в пастку локального мінімуму, знаходячи якийсь мінімальний показник помилки, який збільшується після зміни параметрів. Такий мінімум не завжди є найкращим, тому вчені ввели стохастичну оптимізацію в нейронні мережі - час від часу вводячи випадкові змінні в параметри, дозволяючи алгоритму досліджувати більше можливих локальних мінімумів [21].

В більш складних завданнях, наприклад в процесі створення алгоритму для гри в Го, навчання з учителем неможливе, тому що нема можливості створити великий набір промаркованих даних. Через це ми не можемо надати алгоритму інформації про те, чи правильна його дія чи ні. Щоб вирішити цю проблему, було запроваджено навчання з підкріпленням - тип навчання, при якому алгоритм отримує винагороду, кількісну оцінку своїх дій.

2.3 Рекурентні нейронні мережі

Рекурентні нейронні мережі — це клас штучних нейронних мереж, у яких з’єднання між вузлами утворюють орієнтований граф, що дозволяє мережі мати внутрішній стан. Це створює можливість обробляти послідовності даних, зберігаючи інформацію про попередні елементи послідовності. На відміну від нейронних мереж прямого поширення, RNN можуть використовувати свою внутрішню пам’ять для обробки довільних послідовностей входів. Це робить їх особливо корисними для задач, де контекст або порядок даних має значення, таких як обробка природної мови, розпізнавання мовлення та прогнозування часових рядів [22]. Рекурентна мережа в розгорнутому стані представлена на рисунку 2.4.

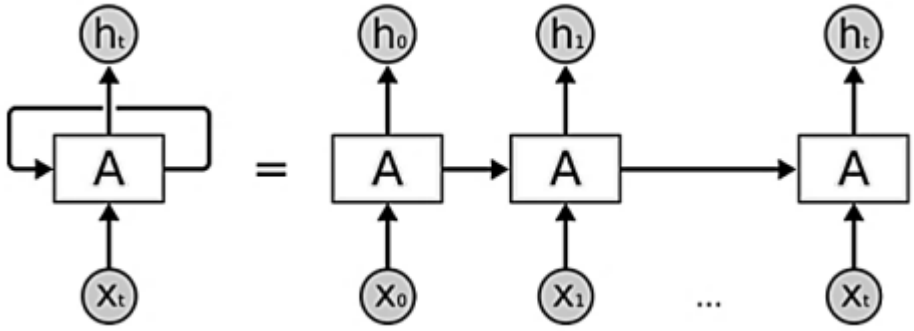

Рисунок 2.4 – Рекурентна мережа у розгортці

Існує кілька типів архітектур рекурентних нейронних мереж, кожна з яких має свої особливості:

– Повнорекурентна мережа – основна архітектура, де кожен вузол з’єднаний з кожним іншим вузлом. Ця архітектура дозволяє зберігати інформацію про всі попередні стани, але може бути складною для тренування через проблему зникнення градієнта.

– Мережі Елмана та Джордана – включають додаткові вузли для зберігання стану. Мережа Елмана має контекстні вузли, які зберігають попередній стан, тоді як мережа Джордана використовує вихідні вузли для зберігання стану [23].

– Довга короткочасна пам’ять (LSTM) – вирішує проблему зникнення градієнта, дозволяючи зберігати інформацію на довгий час. LSTM використовує спеціальні вузли, які можуть зберігати та передавати інформацію через багато часових кроків, що робить їх ефективними для обробки довгих послідовностей.

– Вентильний рекурентний вузол (GRU) – спрощена версія LSTM, яка також ефективно зберігає інформацію. GRU має менше параметрів, що робить їх швидшими для тренування, але вони все ще здатні зберігати довготривалі залежності [24].

Даний тип мереж є потужним інструментом для вирішення задач, де важливо враховувати тимчасову залежність та послідовність даних. Ці мережі відрізняються від звичайних нейронних мереж наявністю зворотних зв'язків, що дозволяє їм мати «пам'ять» про попередні стани, що надзвичайно корисно для обробки часових рядів та інших послідовних даних. Завдяки цьому RNN застосовуються в ряді різних областей, де важлива прогнозна здатність та обробка даних з контекстом. Прикладами таких застосувань можуть бути обробка EEG сигналів на основі моделей лінійного випадкового процесу [25] чи аналіз властивосте мультиваріативних умовних лінійних процесів [26].

Однією з основних сфер застосування рекурентних нейронних мереж є обробка природної мови. RNN використовуються для таких завдань, як машинний переклад, розпізнавання мовлення, генерація тексту, а також в аналізі емоцій та настроїв у текстах. У цих задачах RNN дозволяють моделювати взаємозв'язки між словами в реченнях, враховуючи контекст попередніх та майбутніх слів.

Іншою важливою сферою застосування є фінансовий аналіз, де RNN використовуються для прогнозування курсів валют, цін на акції та інших економічних індикаторів. Завдяки здатності рекурентних мереж працювати з часовими рядами, вони можуть ефективно враховувати тренди, сезонні коливання та інші фактори, що впливають на фінансові ринки [27].

У медичній сфері РНМ застосовуються для аналізу медичних зображень, прогнозування розвитку хвороб, а також для створення систем підтримки прийняття рішень, які допомагають лікарям на основі історії хвороби пацієнта. Крім того, РНМ активно використовуються для обробки біоінформатичних даних, таких як послідовності ДНК, що дозволяє виявляти генетичні маркери хвороб.

Ще однією цікавою галуззю є робототехніка, де рекурентні нейронні мережі застосовуються для планування рухів, прийняття рішень та взаємодії з навколишнім середовищем. Мережі здатні обробляти інформацію про попередні стани та передбачати наступні дії в реальному часі [28].

RNN мають деякі обмеження, такі як проблема зникнення градієнта, що ускладнює тренування мережі на довгих послідовностях. Для вирішення цієї проблеми були розроблені архітектури LSTM та GRU, які дозволяють ефективно зберігати інформацію на довгий час. Крім того, існують інші удосконалення, такі як використання регуляризації для запобігання перенавчанню та застосування методів нормалізації для покращення стабільності тренування.

2.4 Архітектура LSTM

Довга короткочасна пам’ять є спеціалізованим типом рекурентних нейронних мереж, створеним для вирішення проблеми згасання градієнта, яка властива звичайним RNN. У традиційних рекурентних мережах важлива інформація часто втрачається під час навчання, особливо при обробці довгих

послідовностей. Це відбувається через те, що градієнти зменшуються до дуже малих значень, ускладнюючи оновлення ваг мережі. Як наслідок, класичні RNN мають обмежені можливості для роботи із залежностями на великих часових інтервалах [29].

Щоб подолати це обмеження, було розроблено LSTM. Основна особливість цієї архітектури полягає у використанні клітинного стану та спеціальних механізмів, які називаються гейтами. Клітинний стан виступає як довготривала пам’ять, яка передає ключову інформацію через мережу. Гейти, у свою чергу, визначають, які дані необхідно зберегти, а які – забути. Завдяки цьому LSTM здатна ефективно працювати з довгими послідовностями, утримуючи потрібний контекст протягом тривалого часу. Це дозволяє уникнути проблеми згасаючих градієнтів, яка ускладнює навчання традиційних RNN.

LSTM знаходить широке застосування в задачах обробки послідовних даних, таких як текст, аудіо або часові ряди. У сфері обробки природної мови ця технологія допомагає враховувати складні залежності між словами в текстах, дозволяючи зберігати контекст не тільки попередніх, але й майбутніх слів. Крім того, LSTM активно використовують для автоматизованого перекладу, розпізнавання мови, створення текстів, фінансового прогнозування і навіть аналізу біологічних даних. Їх здатність утримувати інформацію про попередні стани робить їх незамінними для задач, де контекст має вирішальне значення.

2.4.1 Основні компоненти LSTM

LSTM складається з кількох ключових компонентів, які працюють разом для зберігання та обробки інформації [30]:

– Комірка пам’яті (Cell State) – основний елемент, який зберігає інформацію протягом тривалого часу. Комірка пам’яті дозволяє LSTM зберігати контекстну інформацію, необхідну для обробки послідовностей.

– Вхідний вентиль (Input Gate) – контролює, яка нова інформація буде збережена в комірці пам’яті. Вхідний вентиль використовує сигмоїдну функцію активації для визначення, які значення оновлювати.

– Забувальний вентиль (Forget Gate) – вирішує, яку інформацію з комірки пам’яті слід забути. Забувальний вентиль також використовує сигмоїдну функцію активації для визначення, які значення видаляти.

– Вихідний вентиль (Output Gate) - контролює, яка інформація з комірки пам’яті буде використана для обчислення вихідного значення. Вихідний вентиль визначає, які значення передавати далі в мережу.

Нейронна мережа на кожному етапі виконує наступні дії:

1. Оновлення забувального вентилю – визначає, яку частину попереднього стану комірки слід забути. Фор

2. Оновлення вхідного вентилю: Визначає, яку нову інформацію слід додати до стану комірки.

3. Оновлення стану комірки: Оновлює стан комірки, комбінуючи старий стан, помножений на забувальний вентиль, з новою інформацією, помноженою на вхідний вентиль.

4. Оновлення вихідного вентилю: Визначає, яку частину стану комірки слід використовувати для обчислення вихідного значення.

2.5 Обробка природньої мови

Обробка природної мови є однією з найважливіших галузей досліджень у сфері штучного інтелекту та комп'ютерних наук, що фокусується на взаємодії між комп'ютерами та людською мовою. Задачі NLP включають автоматичний аналіз, розуміння та генерацію природної мови, що дозволяє машинам розуміти текст і говорити з користувачами на природній мові [31]. На рисунку 2.5 зображені основні задачі NLP.

Рисунок 2.5 – Основні задачі NLP.

Розвиток NLP почався ще в середині 20-го століття, з перших спроб створення машинного перекладу текстів у 1950-х роках. Протягом декількох десятиліть методи NLP еволюціонували від простих статистичних моделей до складних нейронних мереж, здатних обробляти великий обсяг текстових даних з високою точністю [32].

Перші підходи до NLP використовували методи лінгвістичного аналізу та статистики. Зокрема, були створені моделі на основі n-грам, які використовувалися для прогнозування наступного слова в реченні на основі попередніх n слів. Проте ці моделі мали обмежену ефективність при роботі з довгими текстами, оскільки не враховували контекст на великих відстанях.

З початку 2010-х років розвиток глибинного навчання відкрив нові горизонти для NLP. Використання нейронних мереж, зокрема рекурентних нейронних мереж, довготривалої короткочасної пам'яті та трансформерів значно покращило якість обробки текстових даних [33].

Сучасні методи NLP знайшли широке застосування у різних галузях, зокрема:

- NLP використовується для покращення точності пошукових результатів, враховуючи контекст та намір користувача. Це забезпечує більш релевантні результати та покращує користувацький досвід.
- Завдяки моделям на основі RNN, LSTM та трансформерів, системи автоматичного перекладу досягли високої якості перекладів, забезпечуючи зручність для користувачів по всьому світу [34].
- NLP використовується для автоматичного аналізу текстів у соціальних мережах, відгуках та коментарях, що дозволяє виявляти настрої користувачів і тренди. Це є важливим інструментом для маркетингу та досліджень громадської думки.
- Завдяки NLP, чат-боти та віртуальні асистенти здатні підтримувати змістовні діалоги з користувачами, забезпечуючи високий рівень обслуговування та автоматизації [35].
- Моделі на основі глибинного навчання, зокрема LSTM та трансформери, використовуються для автоматичного створення текстів, таких як статті, вірші, музичні тексти та сценарії. Це відкриває нові можливості для творчості та контент-маркетингу [36].

Незважаючи на значні досягнення, NLP стикається з рядом викликів, які потребують подальших досліджень. Одним з основних викликів є проблема упередженості моделей, яка виникає через упередження в тренувальних даних. Це може призводити до неправильних результатів, виданих мережею. Для вирішення цієї проблеми важливо використовувати збалансовані та різноманітні текстові корпуси, а також розробляти методи для виявлення та зменшення упереджень.

Ще одним викликом є необхідність забезпечення контекстної зв'язності та логічності згенерованих текстів на великих відстанях. Хоча сучасні моделі

значно покращили якість генерації тексту, все ще існують випадки, коли модель може втрачати контекст або створювати нелогічні речення.

Перспективи досліджень у сфері NLP включають розвиток нових архітектур моделей, інтеграцію багатомовних підходів, а також використання навчання з підкріпленням для покращення адаптації моделей до контексту та завдань користувачів.

2.6 Роль статистики у задачах NLP

Статистика відіграє ключову роль у роботі та розвитку мовних моделей, допомагаючи їм аналізувати, розуміти та створювати текст природною мовою. Основна концепція полягає в тому, щоб застосовувати ймовірнісні підходи для моделювання послідовностей слів та виявлення мовних закономірностей [37].

Перші мовні моделі будувалися виключно на статистичних методах. Вони використовували принципи теорії ймовірностей для оцінки частоти слів та їхніх комбінацій у текстах [38]. Серед найбільш популярних підходів були:

- N-грамові моделі, які прогнозують наступне слово на основі кількох попередніх (N-1 слів). Цей метод вважається одним із найпростіших у статистичному моделюванні.
- Марковські процеси, що є узагальненням N-грамових моделей. Вони припускають, що наступний стан залежить лише від поточного, тому підходять для задач із короткими залежностями між елементами.
- Частотний аналіз, де оцінюється частота слів у тексті або документі. Цей підхід став основою для традиційних способів представлення тексту у вигляді векторів.

Ймовірнісні моделі визначають імовірність появи певних послідовностей слів у тексті, що дозволяє передбачати наступне слово або оцінювати

"природність" тексту. Важливими характеристиками таких моделей є ймовірнісний розподіл слів та оцінка вірогідності послідовностей.

У статистичних методах, які застосовуються в NLP, важливу роль відіграє також аналіз сигналів. Завдяки цьому можна створювати ефективні архітектури нейронних мереж [39].

Хоча сучасні мовні моделі переважно ґрунтуються на нейронних мережах, статистичні принципи залишаються актуальними, зокрема в таких аспектах:

- Векторні представлення слів. Методи на кшталт Word2Vec чи GloVe [40] базуються на статистичному аналізі контексту, в якому використовуються слова, враховуючи частоту їх спільного вживання. Це дозволяє створювати векторні подання, що відображають значення слова через його зв’язки з іншими.
- Механізм уваги. У трансформерах, хоча вони й побудовані на основі глибокого навчання, статистика використовується для обчислення ваг залежностей між словами. Це дозволяє визначати, на які частини тексту слід звертати більше уваги [41].
- Оптимізація параметрів. Під час навчання моделей застосовуються статистичні алгоритми оптимізації, як-от градієнтний спуск. Стохастичні методи також допомагають уникнути застрягання в локальних мінімумах, сприяючи ефективнішому налаштуванню моделі [42].

2.7 Висновок до другого розділу

В другому розділі кваліфікаційної роботи:

– Описано застосування глибинних мереж;

– Досліджено всі етапи розробки моделей.

3 ПРАКТИЧНА ЧАСТИНА

3.1 Огляд інструментів, використаних в роботі

В даній роботі мною були використані такі інструменти як мова програмування Python, бібліотека NumPy, а також середовище розробки Jupyter Notebook. Опис кожного з цих інструментів наведено в наступних підрозділах.

3.1.1 Мова програмування Python

Python — це одна з найпопулярніших мов програмування у світі, яка відома своєю простотою, читабельністю та широкими можливостями застосування. Її створив Гвідо ван Россум наприкінці 1980-х років, і завдяки зрозумілому та лаконічному синтаксису Python швидко здобула прихильність серед розробників. Основною ідеєю мови було зробити код простим для читання, зручним для написання та забезпечити можливість швидкої розробки [43].

Python підтримує різні парадигми програмування, зокрема об'єктно-орієнтовану, функціональну та імперативну. Її сильні сторони включають великий набір стандартних бібліотек і модулів, а також активну екосистему сторонніх пакетів. Це робить Python універсальним інструментом для вирішення багатьох завдань, починаючи від створення вебзастосунків і закінчуючи аналізом даних чи розробкою моделей машинного навчання.

Особливо Python став важливим у сфері глибинного навчання завдяки таким популярним бібліотекам, як PyTorch, Keras і Scikit-learn [44], які значно спрощують доступ до складних алгоритмів і моделей. У роботі з даними Python також посідає ключове місце завдяки бібліотекам NumPy та Pandas, які забезпечують ефективні інструменти для числових обчислень і роботи з

таблицями. Для візуалізації даних широко використовуються Matplotlib і Seaborn, що допомагає аналізувати результати та знаходити закономірності [45].

Завдяки своїй доступності, гнучкості та великій кількості інструментів, Python фактично став стандартом у глибинному навчанні. Активна спільнота розробників і багатий вибір ресурсів роблять Python чудовим вибором для дослідників, інженерів та розробників, які працюють у сфері штучного інтелекту.

3.1.2 Бібліотека Numpy

NumPy є однією з основних бібліотек Python для роботи з багатовимірними масивами і математичними обчисленнями. Розроблена в кінці 2000-х років, NumPy забезпечує потужні інструменти для маніпуляції масивами, що робить її незамінною в галузях наукових обчислень, аналізу даних, і зокрема в глибинному навчанні [46].

NumPy забезпечує багатий набір функцій для роботи з масивами, включаючи створення, індексацію, об'єднання та маніпуляцію масивами. Далі приведені кілька основних можливостей NumPy:

- Багатовимірні масиви. Основою NumPy є об'єкт ndarray, який представляє собою багатовимірний масив. Цей об'єкт дозволяє легко виконувати різноманітні операції на масивах, такі як додавання, віднімання, множення та інші математичні операції.
- Універсальні функції. NumPy забезпечує універсальні функції, які виконують елементні операції на масивах. Ці функції оптимізовані для високої продуктивності і можуть обробляти великі обсяги даних набагато швидше, ніж стандартні цикли Python.
- Індексація та маніпуляції. NumPy підтримує різні методи індексації, включаючи зрізи, логічну індексацію та індексацію з масивами цілих чисел. Це дозволяє ефективно обирати та обробляти підмасиви даних.

- Лінійна алгебра. NumPy містить модуль linalg, який забезпечує функції для виконання операцій лінійної алгебри, таких як обчислення визначників, розв'язування систем лінійних рівнянь, обчислення власних значень та багато інших.
- Псевдовипадкові числа. NumPy містить модуль random для генерації псевдовипадкових чисел, що є важливим чинником для різних статистичних моделей і алгоритмів машинного навчання.

NumPy є фундаментальною бібліотекою для багатьох інших бібліотек Python, які використовуються у глибинному навчанні, таких як TensorFlow, PyTorch та Keras. Ці бібліотеки часто використовують NumPy для основних операцій з масивами та обчисленнями [47]. Далі подано основні застосування NumPy в глибинному навчанні:

- NumPy широко використовується для підготовки даних перед подальшою обробкою в моделях глибинного навчання. Цей процес включає нормалізацію, масштабування, перетворення та аугментацію даних.
- NumPy використовується для ініціалізації ваг у нейронних мережах. Псевдовипадкові числа, згенеровані за допомогою NumPy, забезпечують початкові значення для ваг, що є критично важливим для ефективного навчання моделей.
- NumPy використовується для обчислення градієнтів під час процесу зворотного поширення у навчанні нейронних мереж. Це дозволяє ефективно оновлювати ваги мережі на основі похибки.
- Після навчання моделі NumPy використовується для аналізу результатів, включаючи обчислення статистичних показників, побудову графіків і візуалізацію даних.

3.1.3 Середовище розробки Jupyter Notebook

Jupyter Notebook — це зручне інтерактивне середовище, яке дає можливість розробникам поєднувати виконуваний код, текст, математичні формули, графіки та мультимедіа в одному документі. Завдяки своїй гнучкості та простоті використання, Jupyter Notebook здобув популярність серед науковців, дослідників і спеціалістів з обробки даних [48].

Одна з головних переваг цієї платформи — можливість виконувати код прямо в документі, отримуючи результати в реальному часі. Це дуже корисно для тестування ідей, налагодження програм та проведення експериментів з даними. Платформа також інтегрується з бібліотеками для створення візуалізацій, такими як Matplotlib, Seaborn і Plotly, що дозволяє будувати якісні графіки та діаграми безпосередньо в блокноті.

Попри те, що Python є основною мовою для Jupyter Notebook, платформа підтримує й інші мови, наприклад, R, Julia та SQL, через використання спеціальних ядер (kernels). Можливість комбінувати код із текстовими коментарями, формулами в LaTeX і зображеннями робить цей інструмент ідеальним для підготовки наукових звітів, створення навчальних матеріалів та технічної документації. Крім того, завдяки інтеграції з хмарними платформами, такими як Google Colab і JupyterHub, розробники можуть легко працювати в командах, обмінюючись документами й результатами в режимі реального часу.

Jupyter Notebook став ключовим інструментом у дослідженнях і розробці моделей глибинного навчання [49]. Його популярність пояснюється зручною інтеграцією з бібліотеками машинного навчання, такими як TensorFlow, PyTorch і Keras, що дозволяє розробникам швидко створювати прототипи, перевіряти та вдосконалювати моделі. Цей інструмент також ідеально підходить для попередньої обробки даних, аналізу та оцінки результатів моделей. Інтерактивні

звіти, які можна створювати в Jupyter Notebook, дозволяють ефективно документувати процес роботи і ділитися отриманими результатами з іншими.

Jupyter Notebook — це незамінний інструмент для роботи з даними та складними моделями. Його інтерактивний підхід, підтримка різних мов програмування та здатність поєднувати різні елементи в одному документі роблять цю платформу ідеальною для дослідників і розробників.

3.2 Фреймворк глибинного навчання

Для виконання завдання було вирішено не використовувати існуючі фреймворки глибинного навчання, а написати свій власний. В подальшому тексті опишемо деякі фрагменти цього фреймворку.

В коді лістингу 3.1 реалізовано функцію обчислення крос-ентропійної втрати для нейронної мережі. Спочатку він обчислює вихід softmax, нормалізуючи експоненційні значення вхідних даних. Потім функція створює цільовий розподіл за допомогою one-hot кодування цільових індексів. Крос-ентропійна втрата обчислюється як середнє значення негативного логарифму ймовірностей правильних класів. Якщо функція автоматичної диференціації активована, вона створює об'єкт Tensor з активованою автоматичною диференціацією та зберігає вихід softmax і цільовий розподіл для подальшого використання в обчисленнях градієнтів. Якщо autograd не активована, функція просто повертає об'єкт Tensor із обчисленою втратою [50]. Повний код створеного фреймворку розміщений в додатку А.

Лістинг 3.1 – Код фреймворку

```
def cross_entropy(self, target_indices):
 temp = np.exp(self.data)
 softmax_output = temp / np.sum(temp,
                                axis=len(self.data.shape)-1,
                                keepdims=True)
 t = target_indices.data.flatten()
 p = softmax_output.reshape(len(t),-1)
 target_dist = np.eye(p.shape[1])[t]
 loss = -(np.log(p) * (target_dist)).sum(1).mean()
 if(self.autograd):
     out = Tensor(loss,
                  autograd=True,
                  creators=[self],
                  creation_op="cross_entropy")
     out.softmax_output = softmax_output
     out.target_dist = target_dist
     return out
 return Tensor(loss)
```

В лістингу 3.2 представлені два класи: SGD і Linear. Клас SGD реалізує алгоритм стохастичного градієнтного спуску (SGD) для оптимізації параметрів нейронної мережі. Він має методи zero, що обнуляє градієнти всіх параметрів, та step, який оновлює параметри за допомогою градієнтів і заданого коефіцієнта навчання (alpha).

Клас Linear успадковує від класу Layer і представляє шар нейронної мережі з лінійним перетворенням. Він ініціалізує ваги та, за необхідності, зміщення, додаючи їх до списку параметрів з підтримкою автоматичної диференціації. Метод forward виконує лінійне перетворення входу, додаючи зміщення, якщо воно використовується.

Лістинг 3.2 – Код фреймворку

```
class SGD(object):
 def __init__(self, parameters, alpha=0.1):
       self.parameters = parameters
      self.alpha = alpha
 def zero(self):
        for p in self.parameters:
            p.grad.data *= 0
 def step(self, zero=True):
   for p in self.parameters:
       p.data -= p.grad.data * self.alpha
       if(zero):
           p.grad.data *= 0

class Linear(Layer):
    def __init__(self, n_inputs, n_outputs, bias=True):
        super().__init__()
        self.use_bias = bias
       W    =    np.random.randn(n_inputs,    n_outputs)    *
np.sqrt(2.0/(n_inputs))
        self.weight = Tensor(W, autograd=True)
        if(self.use_bias):
                    self.bias  =  Tensor(np.zeros(n_outputs),
autograd=True)
        self.parameters.append(self.weight)
```

В лістингу 3.3 представлено два класи. Клас Sequential представляє контейнер для послідовних шарів нейронної мережі. Він дозволяє додавати шари до списку layers за допомогою методу add, а також виконує всі шари послідовно у методі forward, передаючи вихід одного шару як вхід для наступного. Метод get_parameters збирає параметри всіх шарів у контейнері, повертаючи їх у вигляді списку. Клас Embedding представляє векторні представлення слів для

словника заданого розміру і розмірності. Він ініціалізує матрицю ваг, де кожен рядок відповідає векторному представленню слова. Метод forward забезпечує вибір відповідних векторів для входу за допомогою індексного вибору, що дозволяє перетворювати індекси слів у їх векторні представлення для подальшої обробки в нейронній мережі.

Лістинг 3.3 – Код фреймворку

```
class Sequential(Layer):
    def __init__(self, layers=list()):
        super().__init__()
        self.layers = layers
    def add(self, layer):
        self.layers.append(layer)
    def forward(self, input):
        for layer in self.layers:
            input = layer.forward(input)
        return input
    def get_parameters(self):
        params = list()
        for l in self.layers:
            params += l.get_parameters()
        return params

class Embedding(Layer):
    def __init__(self, vocab_size, dim):
        super().__init__()
        self.vocab_size = vocab_size
        self.dim = dim
        self.weight = Tensor((np.random.rand(vocab_size, dim)
- 0.5) / dim, autograd=True)
        self.parameters.append(self.weight)
    def forward(self, input):
```

```
        return self.weight.index_select(input)
```

В лістингу 3.4 представлено клас RNNCell. Він представляє собою окрему комірку рекурентної нейронної мережі (RNN). Він ініціалізується з кількістю входів (n_inputs), кількістю прихованих вузлів (n_hidden) і кількістю виходів (n_output). Клас також приймає параметр activation, який визначає активаційну функцію, якою може бути сигмоїда, або тангенс.

Комірка включає три основні лінійні шари (Linear): w_ih (для входу), w_hh (для прихованих станів) і w_ho (для виходу). У методі forward, комірка обчислює новий прихований стан (new_hidden) шляхом комбінації вхідних даних та попереднього прихованого стану, після чого застосовується активаційна функція. Вихідна інформація обчислюється через лінійний шар w_ho. Метод init_hidden ініціалізує приховані стани нулями для початкового запуску мережі.

Лістинг 3.4 – Код фреймворку

```
class RNNCell(Layer):
    def __init__(self, n_inputs, n_hidden, n_output, activation='sigmoid'):
        super().__init__()
        self.n_inputs = n_inputs
        self.n_hidden = n_hidden
        self.n_output = n_output
        if(activation == 'sigmoid'):
            self.activation = Sigmoid()
        elif(activation == 'tanh'):
            self.activation == Tanh()
        else:
            raise Exception("Non-linearity not found")
        self.w_ih = Linear(n_inputs, n_hidden)
        self.w_hh = Linear(n_hidden, n_hidden)
        self.w_ho = Linear(n_hidden, n_output)
```

```
            self.parameters += self.w_ih.get_parameters()
            self.parameters += self.w_hh.get_parameters()
            self.parameters += self.w_ho.get_parameters()
        def forward(self, input, hidden):
            from_prev_hidden = self.w_hh.forward(hidden)
            combined = self.w_ih.forward(input) + from_prev_hidden
            new_hidden = self.activation.forward(combined)
            output = self.w_ho.forward(new_hidden)
            return output, new_hidden
        def init_hidden(self, batch_size=1):
            return    Tensor(np.zeros((batch_size,self.n_hidden)),
autograd=True)
```

В лістингу 3.5 представлено реалізацію основної частини коду для нашої моделі, а саме клас, який комірку довготривалої короткочасної пам'яті (LSTM), яка є спеціальним типом рекурентної нейронної мережі. Він ініціалізується з кількістю входів (n_inputs), прихованих вузлів (n_hidden) і виходів (n_output). Комірка включає кілька лінійних шарів: xf, xi, xo, і xc для обробки вхідних даних, а також hf, hi, ho, і hc для обробки попереднього прихованого стану.

Метод forward обчислює новий прихований стан і новий клітинний стан за допомогою гейтів забування (f), вхідних гейтів (i), вихідних гейтів (o) та нових кандидатів на клітинний стан (g). Об'єднуючи ці стани, метод обчислює вихід комірки та нові стани. Метод init_hidden ініціалізує приховані стани та клітинні стани нулями для початкового запуску комірки.

Лістинг 3.5 – Код фреймворку

```
class LSTMCell(Layer):
 def __init__(self, n_inputs, n_hidden, n_output):
   super().__init__()
   self.n_inputs = n_inputs
   self.n_hidden = n_hidden
```

```
self.n_output = n_output
self.xf = Linear(n_inputs, n_hidden)
self.xi = Linear(n_inputs, n_hidden)
self.xo = Linear(n_inputs, n_hidden)
self.xc = Linear(n_inputs, n_hidden)
self.hf = Linear(n_hidden, n_hidden, bias=False)
self.hi = Linear(n_hidden, n_hidden, bias=False)
self.ho = Linear(n_hidden, n_hidden, bias=False)
self.hc = Linear(n_hidden, n_hidden, bias=False)
self.w_ho = Linear(n_hidden, n_output, bias=False)
self.parameters += self.xf.get_parameters()
self.parameters += self.xi.get_parameters()
self.parameters += self.xo.get_parameters()
self.parameters += self.xc.get_parameters()
self.parameters += self.hf.get_parameters()
self.parameters += self.hi.get_parameters()
self.parameters += self.ho.get_parameters()
self.parameters += self.hc.get_parameters()
self.parameters += self.w_ho.get_parameters()
```

3.3 Реалізація мережі

На основі написаного раніше фреймворку було створено мережу, частину коду якої представлено в лістингу 3.6. В цьому коді ми визначаємо модель LSTM з використанням векторних представлень слів слів для генерації тексту. Спочатку кодується текстовий файл, створюється словник і відповідні індекси слів. Векторні представлення слів створюються з використанням Embedding класу. Модель LSTM ініціалізується з відповідними розмірами вхідних і вихідних шарів [51]. Код ініціалізації мережі наведений в додатку Б.

Код також визначає функцію generate_sample, яка генерує зразок тексту заданої довжини, починаючи з певного початкового символу. Після цього дані

поділяються на батчі для навчання з використанням зворотного поширення помилки крізь час (BPTT). Оптимізація здійснюється за допомогою стохастичного градієнтного спуску (SGD), а функція втрат обчислюється з використанням крос-ентропії.

Лістинг 3.6 – Код мережі

```
np.random.seed(0)

f = open('text.txt','r')
raw = f.read()
f.close()

vocab = list(set(raw))
word2index = {}
for i,word in enumerate(vocab):
    word2index[word]=i
indices = np.array(list(map(lambda x:word2index[x], raw)))

embed = Embedding(vocab_size=len(vocab),dim=512)
model = LSTMCell(n_inputs=512, n_hidden=512,
n_output=len(vocab))
model.w_ho.weight.data *= 0

criterion = CrossEntropyLoss()
optim = SGD(parameters=model.get_parameters() +
embed.get_parameters(), alpha=0.05)
```

Після цього за допомогою коду, представленого у лістингу 3.7 було проведено навчання мережі. Цей фрагмент коду реалізує процес навчання моделі нейронної мережі за допомогою методу стохастичного градієнтного спуску. Функція train виконує ітеративне навчання моделі протягом заданої кількості

ітерацій, оновлюючи параметри моделі та мінімізуючи втрати. Кожна ітерація складається з кількох батчів, де кожен батч ділиться на менші тимчасові відрізки.

Під час кожного батчу ініціалізуються приховані стани моделі, обчислюються втрати для кожного тимчасового відрізку, і проводиться зворотне поширення помилки для оновлення ваг моделі. Кількість втрат (loss) акумулюється, і відстежується мінімальне значення втрати на кожній ітерації. Параметр alpha, що визначає швидкість навчання, зменшується після кожної ітерації для покращення стабільності навчання. Результати, включаючи генерований текст на кожному першому батчі, виводяться на екран для моніторингу процесу навчання.

Лістинг 3.7 – Код, який відповідає за навчання мережі на тестових даних

```
def train(iterations=400, min_loss=1000):
    for iter in range(iterations):
        total_loss = 0
        n_loss = 0

        hidden = model.init_hidden(batch_size=batch_size)
        batches_to_train = len(input_batches)
        for batch_i in range(batches_to_train):

            hidden = (Tensor(hidden[0].data, autograd=True), 
Tensor(hidden[1].data, autograd=True))

            losses = list()
            for t in range(bptt):
                input = Tensor(input_batches[batch_i][t], 
autograd=True)
                rnn_input = embed.forward(input=input)
                output, hidden = 
model.forward(input=rnn_input, hidden=hidden)
```

```
                    target   =   Tensor(target_batches[batch_i][t],
autograd=True)
                    batch_loss = criterion.forward(output, target)

                    if(t == 0):
                        losses.append(batch_loss)
                    else:
                        losses.append(batch_loss + losses[-1])

                loss = losses[-1]
                loss.backward()
                optim.step()
                total_loss += loss.data / bptt
                epoch_loss = np.exp(total_loss / (batch_i+1))
```

Після 150 ітерацій, виконання яких зайняло 12 годин реального часу, було отримано мережу, показник втрат якої складає 11.2145. Дана мережа здатна генерувати текст на основі попереднього символу.

Приклад згенерованого мережею тексту представлено на рисунку 3.1.

```
[37]: print(generate_sample(n=10000, init_char='t'))

 speak will ene te the trume for trume be will e will not din the seell men, ifill will e well drest
oul will is to ll weepo the will sist to come to l will pees lord, will not winge not lor trume for
the lor trument of the seeve tres should Leed brink with true thee wit despell tere tud sone the kin
g entrest in the dres e ll hen will exe tuns ould will peese will exe twe barrie oul well e will not
s will sis ell seeme the blore the tuugh will sisee will sist to come trume two clore trut in to tus
in tweepout I see tiell with will sisene exe trumentertll eee will sisell nee trespirit it pard, wil
l not wing you shell seem will peak will e twe en teell my lord, you will expies I will not s out in
thave the see there pres ffertter
To dor te the will see well do
To will silk will love will sisee tere the seq-ll see expres to ll wish e will love ent it love the
seme tie thetertun welll seeme the wite ting in the seemexty oun will nee time the seexe to lor true
trese my bar will e to with true tres sone trumpir the seeve tremy dee time tremin  will in the king
you to-e the true my lord, to ll winge truin the seell e to cll not dist of trus of to the seqe the
pout till every powe te ll seelll lor tus of the seem with the sis eme the subar the will sisee will
seeme exe thee bake the see will peak will looe twe tunentert pit sis es of there be one these my cl
ove all not lord, will not winge not lord, will not winglone tere till e we the not ge trull will se
empe the will sisee thee noble pord in to cll with twe tere their with e will de trese we twe the tr
umpell wees till expon to lik hill sis es e the will dest tion my lor trumpart soull see timing come
```

Рисунок 3.1 – Приклад запиту до мережі.

Проаналізуємо згенерований текст. Перш за все, звернемо увагу на структуру та зміст тексту. Незважаючи на те, що текст містить рядки, які зовні нагадують природну мову, він все ж має багато невідповідностей і семантичних помилок. Наприклад, такі слова як "trume", "weepo", "exe" та інші не мають сенсу в англійській мові, а фрази "speak will ene te the trume" або "lord, will not winge not lor trume" є граматично неправильними і беззмістовними.

Такі помилки можуть бути пояснені особливостями роботи моделей типу LSTM. LSTM-комірки мають механізми для збереження та відновлення інформації на довгих відстанях у тексті, що дозволяє їм моделювати залежності між словами. Однак, навіть ці механізми мають свої обмеження і можуть допускати помилки, особливо при роботі з великими обсягами даних або складними структурами мови.

Цей текст демонструє кілька позитивних моментів роботи нейронної мережі. Вона змогла згенерувати текст, що містить певні структури речень, пробіли, знаки пунктуації та інші мовні елементи. Використання слів "lord", "will", "king" і інших вказує на те, що мережа спробувала вбудувати в текст семантичні зв'язки, типові для англійської мови. Це свідчить про те, що модель здатна до певної міри враховувати контекст і створювати текст, який може бути схожим на природну мову.

Однак, основним недоліком згенерованого тексту є відсутність зв'язного змісту і логічної послідовності. Багато речень є беззмістовними, а деякі фрази повторюються з незначними змінами. Це може бути пов'язано з тим, що модель не змогла достатньо добре навчитися на тренувальних даних або не змогла врахувати всі контекстуальні зв'язки між словами.

Крім того, згенерований текст містить багато повторюваних фраз, що свідчить про можливу проблему з перенавчанням моделі. Перенавчання може призвести до того, що модель починає запам'ятовувати тренувальні дані замість того, щоб навчатися генералізувати на нових даних. Проблема перенавчання

виникла через обмеженість тренувального набору даних. Це може бути виправлено шляхом збільшення різноманітності тренувальних даних або використання методів регуляризації під час навчання моделі.

Отже, аналіз згенерованого тексту показує, що модель має потенціал для генерації тексту, схожого на природну мову, але все ще має значні обмеження.

3.4 Висновок до третього розділу

В третьому розділі кваліфікаційної роботи:

– Спроєктовано і розроблено модель глибинного навчання для генерації тексту;

– Протестовано текст, згенерований моделлю.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Проведення інструктажів з охорони праці

Проведення інструктажів з охорони праці є важливим елементом забезпечення безпеки робочого середовища для розробників нейронних мереж. Оскільки така робота передбачає тривалу взаємодію з комп'ютерним обладнанням, спеціалізованим програмним забезпеченням і часто реалізується в умовах підвищеної інтелектуальної напруги, питання охорони праці набувають особливого значення. Інструктажі є основним інструментом ознайомлення працівників із правилами безпечної роботи, профілактики травматизму та запобігання впливу шкідливих виробничих факторів [52].

Інструктажі забезпечують систематичне інформування працівників про можливі ризики під час виконання їхніх професійних обов'язків. У контексті роботи з нейронними мережами це включає дотримання правил роботи з обчислювальним обладнанням, серверними системами, а також розуміння вимог інформаційної безпеки. Інструктажі дозволяють мінімізувати ризики, пов'язані з технічними несправностями, електробезпекою та кіберзагрозами. Крім того, вони сприяють профілактиці професійного вигорання та перевтоми, що є поширеними проблемами серед фахівців галузі штучного інтелекту.

Процес проведення інструктажів охоплює кілька ключових етапів. Спочатку необхідно визначити специфіку роботи розробників нейронних мереж і виявити потенційні ризики, пов'язані з їхньою діяльністю. На основі цього складаються програми інструктажів, які містять інформацію про загальні правила охорони праці, специфічні для конкретного підприємства або проєкту вимоги, а також рекомендації з організації робочого процесу. Під час

інструктажу працівників знайомлять з нормативно-правовими документами, що регламентують охорону праці, а також із внутрішніми правилами підприємства.

Окрему увагу варто приділити організації робочих місць розробників. Інструктажі повинні охоплювати питання ергономіки робочого простору, налаштування обладнання для зниження фізичного навантаження та підтримання оптимального мікроклімату. Розробники повинні знати, як уникати перевтоми, правильно організовувати перерви та виконувати вправи для зняття м’язового напруження. Також важливими є рекомендації з профілактики зорового стомлення, особливо під час роботи з великими обсягами даних і складними візуалізаціями.

У роботі з нейронними мережами актуальними є ризики, пов’язані з інформаційною безпекою. Інструктажі мають включати правила захисту конфіденційної інформації, роботи з ліцензійним програмним забезпеченням, а також запобігання кібератакам. Це особливо важливо в контексті розробки алгоритмів, що обробляють великі масиви даних, включаючи персональну та корпоративну інформацію.

Типи інструктажів:

1. Вступний інструктаж проводиться під час прийняття працівника на роботу та спрямований на ознайомлення з основними правилами охорони праці, особливостями роботи підприємства та основними ризиками, що можуть виникати.

2. Первинний інструктаж на робочому місці: охоплює специфіку виконання конкретних завдань, використання обладнання та дотримання правил техніки безпеки.

3. Повторний інструктаж проводиться періодично, щоб актуалізувати знання працівників про правила охорони праці, враховуючи зміни в робочих процесах чи законодавстві.

4. Позаплановий інструктаж необхідний у разі змін у технологічних процесах, оновлення обладнання чи виникнення аварійних ситуацій.

5. Цільовий інструктаж: здійснюється перед виконанням робіт підвищеної небезпеки або в разі реалізації нових проєктів.

Ефективність інструктажів значною мірою залежить від їхньої регулярності, повноти інформації та способу подачі матеріалу. Використання сучасних технологій, таких як інтерактивні тренінги, симуляції та відеоуроки, дозволяє зробити інструктажі більш доступними та зрозумілими для працівників. Також важливо враховувати зворотний зв’язок: працівники повинні мати можливість ставити запитання, ділитися пропозиціями щодо покращення умов праці та отримувати кваліфіковані відповіді від фахівців з охорони праці.

Роль роботодавця в організації інструктажів є визначальною. Він має забезпечити створення якісної програми навчання, залучення компетентних фахівців і контроль за дотриманням працівниками правил охорони праці. Водночас відповідальність працівників полягає у свідомому ставленні до власної безпеки та активній участі в навчальному процесі.

Загалом, інструктажі з охорони праці для розробників нейронних мереж є необхідним заходом для створення безпечного та комфортного робочого середовища. Вони сприяють збереженню здоров’я працівників, підвищенню ефективності їхньої діяльності та мінімізації ризиків, пов’язаних із виконанням професійних обов’язків.

4.2 Загальні вимоги безпеки з охорони праці для користувачів ПК

Робота за комп’ютером є невід’ємною частиною сучасного трудового процесу, що вимагає дотримання відповідних заходів безпеки для забезпечення здоров’я і продуктивності працівників. Загальні вимоги з охорони праці для користувачів ПК спрямовані на мінімізацію ризиків, пов’язаних із тривалою

роботою за комп’ютером, включаючи фізичне, психологічне та інформаційне навантаження. Впровадження цих заходів дозволяє створити комфортне і безпечне робоче середовище для працівників [54].

В Україні питання охорони праці при використанні ПК регламентують наступні документи:

1. Кодекс законів про працю України — основний закон, що регулює трудові відноси, які включають в себе і охорону праці.

2. Державні санітарні правила і норми (ДСанПіН 3.3.2.007-98) — встановлюють вимоги до організації робочих місць працівників.

Закон України "Про охорону праці"— визначає основні принципи охорони праці, зокрема під час роботи з обладнанням і комп’ютерною технікою.

3. Закон України "Про охорону праці"— визначає основні принципи охорони праці, зокрема під час роботи з обладнанням і комп’ютерною технікою.

4. Норми часу безперервної роботи з ПК — наказ Міністерства охорони здоров’я України про обмеження безперервного часу роботи за комп’ютером для різних категорій працівників.

5. ISO 9241 (міжнародний стандарт) — стосується ергономіки програмного забезпечення і техніки, що може бути застосовано для покращення умов праці.

Робоче місце користувачів ПК повинно бути організоване відповідно до ергономічних вимог. Стіл і стілець мають бути належної висоти, щоб забезпечити зручну позицію працівника. Оптимальне розташування монітора, клавіатури та миші дозволяє знизити навантаження на опорно-руховий апарат і уникнути розвитку професійних захворювань, таких як тунельний синдром зап’ястків. Освітлення робочого місця має бути достатнім, щоб уникнути перенапруження зору [53].

Температура в приміщенні повинна підтримуватися на рівні 18-22°C, а вологість — у межах 40-60%. Це забезпечує комфортні умови для роботи та

сприяє підтримці працездатності. Вентиляція приміщення є важливим аспектом санітарно-гігієнічних вимог, оскільки свіже повітря покращує концентрацію уваги і запобігає виникненню головного болю та втоми.

Окрім ергономіки і мікроклімату, важливу роль відіграють перерви в роботі. Регулярні перерви через кожні 1,5-2 години допомагають знизити навантаження на очі та м’язи. Протягом перерв рекомендується виконувати вправи для зняття м’язової напруги, такі як розтяжки або кругові рухи шиєю, а також вправи для релаксації зору, наприклад, погляд у далечінь.

Технічне обладнання повинно відповідати сучасним стандартам. Монітори мають бути з низьким рівнем мерехтіння і високою роздільною здатністю, що знижує навантаження на зір. Клавіатура та миша повинні бути ергономічно спроектованими, а електричні кабелі — належним чином захищеними від механічних пошкоджень.

Для забезпечення інформаційної безпеки необхідно використовувати ліцензійне програмне забезпечення, яке регулярно оновлюється для захисту від вірусів і кіберзагроз. Захист даних передбачає резервне копіювання важливих файлів та встановлення антивірусних програм. Крім того, доступ до комп’ютерної мережі повинен бути захищений паролями та іншими засобами аутентифікації.

Організація робочого процесу повинна враховувати психофізіологічні аспекти. Для зниження стресу та перевтоми рекомендується чергування видів діяльності, а також проведення тренінгів зі стресостійкості. Виконання вправ для релаксації дозволяє підтримувати емоційний баланс і знижувати напруження.

Загальні вимоги безпеки з охорони праці для користувачів ПК мають на меті створення сприятливого робочого середовища, яке мінімізує ризики для здоров’я і підвищує ефективність роботи. Дотримання цих вимог є ключовим фактором для підтримки високого рівня продуктивності та забезпечення здоров’я працівників у довгостроковій перспективі.

4.3 Фактори, що впливають на функціональний стан користувачів комп’ютера

Фактори, що впливають на функціональний стан користувачів комп’ютера, є багатогранними та мають значний вплив на різні аспекти здоров’я людини — фізичний, психоемоційний та когнітивний. Тривале використання комп’ютерів стає невід’ємною частиною сучасного життя, однак цей прогрес супроводжується ризиками, які потребують об’єктивної оцінки та систематичних заходів профілактики.

Одним із ключових факторів, що впливають на функціональний стан, є організація робочого місця. Зручне та правильно обладнане робоче місце є основою для підтримки фізичного комфорту та зниження навантаження на опорно-руховий апарат. Монітор має бути розташований на рівні очей, щоб уникнути нахилів голови, які можуть призводити до перенапруження м’язів шиї. Відстань до екрана повинна становити 50–70 см, а його нахил — 15–20° від вертикалі. Робочий стіл повинен бути достатньо широким для зручного розміщення обладнання, а стілець — регульованим, із підтримкою для попереку та правильною висотою для ніг. Дослідження підтверджують, що ергономічно обладнане робоче місце може знизити ризик виникнення захворювань опорно-рухового апарату на 50%.

Особливе значення має освітлення робочої зони. Недостатнє або надто яскраве світло викликає перенапруження очей, сприяючи розвитку синдрому комп’ютерного зору. Найкращим варіантом є використання природного освітлення, але за його відсутності необхідно обирати лампи з нейтральною кольоровою температурою (4000–5000 К), які забезпечують комфортне сприйняття без відблисків. Важливо уникати віддзеркалень на екрані монітора, оскільки це збільшує зорове навантаження. За даними Американської

оптометричної асоціації, оптимальне освітлення робочого місця може знизити ймовірність розвитку синдрому комп'ютерного зору на 25%.

Тривалість і характер роботи за комп'ютером також значно впливають на функціональний стан користувачів. Безперервна робота протягом тривалого часу без перерв негативно позначається на фізичному та психічному стані. За рекомендаціями, кожні 50–60 хвилин слід робити короткі перерви тривалістю 5–10 хвилин. У цей час доцільно виконувати фізичні вправи, спрямовані на розслаблення м'язів, що найбільше навантажуються, та покращення кровообігу. Проведені дослідження показують, що регулярні короткі перерви можуть підвищити продуктивність праці на 10-15% та знизити рівень стресу.

Малорухливий спосіб життя, пов'язаний із тривалою роботою за комп'ютером, є ще одним вагомим фактором ризику. Тривале сидіння сприяє застійним явищам у кровоносній системі, що може призводити до розвитку варикозного розширення вен, тромбозу та підвищення ризику серцево-судинних захворювань. Щоб уникнути цього, важливо впроваджувати активні перерви та виконувати спеціальні вправи для підтримання рухливості суглобів і активізації кровообігу. Наприклад, ходьба на місці або легкі розтягувальні вправи допомагають уникнути застою крові та покращують загальний тонус організму. Дослідження показують, що регулярна фізична активність протягом робочого дня знижує ризик серцево-судинних захворювань на 30%.

Важливим аспектом є якість обладнання, яке використовується під час роботи за комп'ютером. Застарілі монітори з низькою роздільною здатністю та частотою оновлення зображення можуть спричиняти перенапруження очей і розвиток зорових порушень. Сучасні монітори оснащені технологіями для зменшення синього випромінювання, що допомагає знизити навантаження на очі. Використання антивідблискових екранів, а також спеціальних окулярів для роботи за комп'ютером є ефективними профілактичними заходами. За даними досліджень, використання сучасних моніторів з технологією зменшення синього

світла може знизити втому очей на 20%. Психоемоційне навантаження, яке супроводжує роботу за комп’ютером, є ще одним важливим фактором. Постійний доступ до великого обсягу інформації, необхідність швидкого прийняття рішень і багатозадачність сприяють розвитку стресу, емоційного вигорання та тривожних розладів. Щоб мінімізувати ці ризики, важливо організувати робочий процес із чіткими межами між роботою та відпочинком, забезпечити належну кількість часу для відновлення та відпочинку, а також уникати роботи в надмірно інтенсивному режимі. Дослідження показують, що чітке розмежування робочого та особистого часу може знизити ризик емоційного вигорання на 50%. Ще одним фактором, що впливає на функціональний стан користувачів комп’ютера, є навколишнє середовище. Сприятливі умови в приміщенні, де проводиться робота, суттєво покращують самопочуття та продуктивність. Температура повітря повинна бути в межах 18–22°C, вологість — 40–60%. Надмірна сухість або вологість повітря може викликати дискомфорт і негативно позначатися на здоров’ї, тому слід використовувати зволожувачі чи осушувачі повітря залежно від ситуації. Якісна вентиляція або очищення повітря також сприяють покращенню загального функціонального стану. За даними досліджень, оптимальні умови повітря в приміщенні можуть підвищити продуктивність праці на 5-10%. Електромагнітне випромінювання, яке генерують електронні пристрої, є об’єктом уваги багатьох досліджень. Хоча сучасні комп’ютери відповідають стандартам безпеки, тривалий контакт із технікою може викликати суб’єктивний дискомфорт. Для мінімізації впливу електромагнітного випромінювання слід дотримуватись рекомендованої відстані від монітора та інших пристроїв, використовувати якісне обладнання та періодично робити перерви у роботі. Дослідження підтверджують, що дотримання цих рекомендацій може знизити ризик негативного впливу електромагнітного випромінювання на 30%. Особистісні характеристики користувачів також мають значення. Люди з розвиненими навичками

самоконтролю, стресостійкості та тайм-менеджменту легше адаптуються до вимог, пов’язаних із тривалою роботою за комп’ютером. Водночас, недостатній рівень організованості може сприяти накопиченню стресу та зниженню продуктивності. Регулярна фізична активність, правильне харчування та достатній сон є невід’ємними складовими для підтримки функціонального стану на високому рівні. Дослідження показують, що регулярна фізична активність може підвищити загальний рівень енергії та продуктивності на 20%. - розшир цей текст

Важливим аспектом є якість обладнання, яке використовується під час роботи за комп’ютером. Застарілі монітори з низькою роздільною здатністю та частотою оновлення зображення можуть спричиняти перенапруження очей і розвиток зорових порушень. Дослідження показують, що перенапруження очей може призвести до головного болю, затуманення зору та навіть пошкодження зорового апарату при тривалому впливі. Сучасні монітори оснащені технологіями для зменшення синього випромінювання, що допомагає знизити навантаження на очі. Використання антивідблискових екранів, а також спеціальних окулярів для роботи за комп’ютером є ефективними профілактичними заходами. За даними досліджень, використання сучасних моніторів з технологією зменшення синього світла може знизити втому очей на 20%. Це особливо важливо для людей, які проводять за комп'ютером більше 4 годин на день.

Психоемоційне навантаження, яке супроводжує роботу за комп’ютером, є ще одним важливим фактором. Постійний доступ до великого обсягу інформації, необхідність швидкого прийняття рішень і багатозадачність сприяють розвитку стресу, емоційного вигорання та тривожних розладів. Це може призводити до зниження концентрації, проблем зі сном та підвищеної втомлюваності. Щоб мінімізувати ці ризики, важливо організувати робочий процес із чіткими межами між роботою та відпочинком, забезпечити належну кількість часу для

відновлення та відпочинку, а також уникати роботи в надмірно інтенсивному режимі. Дослідження показують, що чітке розмежування робочого та особистого часу може знизити ризик емоційного вигорання на 50%. Регулярні перерви, фізичні вправи та практики релаксації, такі як медитація або йога, можуть суттєво знизити рівень стресу і покращити загальний стан.

Ще одним фактором, що впливає на функціональний стан користувачів комп'ютера, є навколишнє середовище. Сприятливі умови в приміщенні, де проводиться робота, суттєво покращують самопочуття та продуктивність. Температура повітря повинна бути в межах 18–22°C, вологість — 40–60%. Надмірна сухість або вологість повітря може викликати дискомфорт і негативно позначатися на здоров'ї, тому слід використовувати зволожувачі чи осушувачі повітря залежно від ситуації. Якісна вентиляція або очищення повітря також сприяють покращенню загального функціонального стану. За даними досліджень, оптимальні умови повітря в приміщенні можуть підвищити продуктивність праці на 5-10%. Крім того, використання живих рослин у приміщенні може покращити якість повітря і створити сприятливу атмосферу для роботи.

Електромагнітне випромінювання, яке генерують електронні пристрої, є об'єктом уваги багатьох досліджень. Хоча сучасні комп'ютери відповідають стандартам безпеки, тривалий контакт із технікою може викликати суб'єктивний дискомфорт. Електромагнітні поля можуть впливати на нервову систему, викликати головний біль та втому. Для мінімізації впливу електромагнітного випромінювання слід дотримуватись рекомендованої відстані від монітора та інших пристроїв, використовувати якісне обладнання та періодично робити перерви у роботі. Дослідження підтверджують, що дотримання цих рекомендацій може знизити ризик негативного впливу електромагнітного випромінювання на 30%. Використання захисних екранів та правильне розташування електронних пристроїв може додатково знизити цей ризик.

Особистісні характеристики користувачів також мають значення. Люди з розвиненими навичками самоконтролю, стресостійкості та тайм-менеджменту легше адаптуються до вимог, пов’язаних із тривалою роботою за комп’ютером. Водночас, недостатній рівень організованості може сприяти накопиченню стресу та зниженню продуктивності. Регулярна фізична активність, правильне харчування та достатній сон є невід’ємними складовими для підтримки функціонального стану на високому рівні. Дослідження показують, що регулярна фізична активність може підвищити загальний рівень енергії та продуктивності на 20%. Крім того, підтримка здорового способу життя, включаючи правильне харчування та регулярні фізичні вправи, сприяє загальному поліпшенню здоров’я та підвищенню стійкості до стресу.

Таким чином, збереження функціонального стану користувачів комп’ютера вимагає комплексного підходу, який враховує як технічні, так і фізичні, психоемоційні та екологічні фактори. Застосування сучасних технологій та дотримання рекомендацій щодо здорового способу життя можуть суттєво знизити ризики, пов’язані з тривалою роботою за комп'ютером, і забезпечити високий рівень продуктивності та здоров’я.

4.4 Висновок до четвертого розділу

У розділі «Охорона праці та безпека в надзвичайних ситуаціях» проаналізовано фактори, що впливають на функціональний стан користувачів комп’ютера. Опрацьовано документи щодо охорони праці, які діють на території України.

ВИСНОВКИ

В результаті виконання магістерської кваліфікаційної роботи було розроблено модель глибинного навчання для генерації тексту.

В першому розділі кваліфікаційної роботи:

– Розглянуто предметну область обробки природньої мови

– Проаналізовано основні алгоритми в даній області

– Досліджено основні моделі конкурентів

– Обґрунтовано вибір архітектури для нейронної мережі

– Сформовано вимоги до розроблюваної моделі

В другому розділі кваліфікаційної роботи:

– Описано застосування глибинних мереж

– Досліджено всі етапи розробки моделей

В третьому розділі кваліфікаційної роботи:

– Спроєктовано і розроблено модель глибинного навчання для генерації тексту

– Протестовано текст, згенерований моделлю

У розділі «Охорона праці та безпека в надзвичайних ситуаціях» проаналізовано фактори, що впливають на функціональний стан користувачів комп’ютера.

ДЖЕРЕЛА

1. Що таке обробка природної мови (NLP) та як вона може використовуватися у бізнесі [Електронний ресурс] – Режим доступу до ресурсу: https://metinvest.digital/ua/page/1052.

2. NLP: Current Trends and Future Directions [Електронний ресурс] – Режим доступу до ресурсу: https://www.linkedin.com/checkpoint/challengesV2/AQF7R6iFmfrWYAAAAZPd-Onm2gybFNvlYli4-NZ59nOxdq3r0prC8s9p2LVml4MbhPG8AHAuFHd6PBKN_NP7Gy0t3AtOW5cdgA?ut=3YkehjQqJH8rA1.

3. Word Embedding using Word2Vec [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/python-word-embedding-using-word2vec/.

4. What is GPT-3? [Електронний ресурс] – Режим доступу до ресурсу: https://www.techtarget.com/searchenterpriseai/definition/GPT-3.

5. Ethical concerns mount as AI takes bigger decision-making role in more industries [Електронний ресурс] – Режим доступу до ресурсу: https://news.harvard.edu/gazette/story/2020/10/ethical-concerns-mount-as-ai-takes-bigger-decision-making-role/.

6. The biggest problem with GPT-4 is philosophical: What is truth? And do we trust AI to tell us? [Електронний ресурс] – Режим доступу до ресурсу: https://bigthink.com/the-future/biggest-problem-gpt-4-philosophical-what-is-truth/.

7. What is RNN (Recurrent Neural Network)? [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/what-is/recurrent-neural-network/.

8. What is LSTM – Long Short Term Memory? [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory/.

9. What Is a Transformer Model? [Електронний ресурс] – Режим доступу до ресурсу: https://blogs.nvidia.com/blog/what-is-a-transformer-model/.

10. What is GPT? [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/what-is/gpt/.

11. BERT language model [Електронний ресурс] – Режим доступу до ресурсу: https://www.techtarget.com/searchenterpriseai/definition/BERT-language-model.

12. T5 [Електронний ресурс] – Режим доступу до ресурсу: https://huggingface.co/docs/transformers/model_doc/t5.

13. The Ultimate Guide to Building Your Own LSTM Models [Електронний ресурс] – Режим доступу до ресурсу: https://www.projectpro.io/article/lstm-model/832.

14. Make your own neural network / Tarik Rashid, 2019. – 272 с.

15. Luis Serrano. Grokking Machine Learning, 2023. – 512 с.

16. Fryz M., Mlynko B. Property Analysis of Conditional Linear Random Process as a Mathematical Model of Cyclostationary Signal. 2nd International Workshop on Information Technologies: Theoretical and Applied Problems (ITTAP 2022). Ternopil, Ukraine: CEUR Workshop Proceedings, 2022. Vol. 3309. P. 77–82.

17. Alexey Grigorev. Machine Learning Bookcamp. Build a portfolio of real-life projects, 2021. – 496 с

18. Як ми вчимося. Чому мозок навчається краще, ніж машина… Поки що / пер. з англ. Юлія Костюк. – 3-тє вид. – К. :Лабораторія, 2023. – 288 с

19. Natural and Artificial Intelligence: A brief introduction to the interplay between AI and neuroscience research [Електронний ресурс] – Режим доступу до ресурсу: https://www.sciencedirect.com/science/article/pii/S0893608021003683.

20. Multi-Layer Neural Network [Електронний ресурс] – Режим доступу до ресурсу: http://deeplearning.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/.

21. What is gradient descent? [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/think/topics/gradient-descent.

22. Introduction to Recurrent Neural Networks [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/.

23. Difference between Jordan, Elman and normal RNN [Електронний ресурс] – Режим доступу до ресурсу: https://datascience.stackexchange.com/questions/82416/difference-between-jordan-elman-and-normal-rnn.

24. https://www.geeksforgeeks.org/gated-recurrent-unit-networks/ [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/gated-recurrent-unit-networks/.

25. Fryz M., Scherbak L., Mlynko B., Mykhailovych T. Linear Random Process Model-Based EEG Classification Using Machine Learning Techniques. Proceedings of the 1st International Workshop on Computer Information Technologies in Industry 4.0 (CITI 2023). Ternopil, Ukraine: CEUR Workshop Proceedings, 2023. Vol. 3468. P. 126–132.

26. M. Fryz, B. Mlynko, Property analysis of multivariate conditional linear random processes in the problems of mathematical modelling of signals, Technology Audit and Production Reserves, 3/2(65), 2022, pp. 29–32.

27. AI In Finance: Time Series Forecasting with Recurrent Neural Networks [Електронний ресурс] – Режим доступу до ресурсу: https://www.signitysolutions.com/tech-insights/time-series-forecasting-with-rnn.

28. Recurrent neural network for motion trajectory prediction in human-robot collaborative assembly [Електронний ресурс] – Режим доступу до ресурсу: https://www.sciencedirect.com/science/article/abs/pii/S0007850620300998.

29. Understanding Architecture of LSTM [Електронний ресурс] – Режим доступу до ресурсу: https://www.analyticsvidhya.com/blog/2021/01/understanding-architecture-of-lstm/.

30. An Intuitive Explanation of LSTM [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@ottaviocalzone/an-intuitive-explanation-of-lstm-a035eb6ab42c.

31. Natural Language Processing (NLP) – Overview [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/natural-language-processing-overview/.

32. A Brief History of Natural Language Processing [Електронний ресурс] – Режим доступу до ресурсу: https://www.dataversity.net/a-brief-history-of-natural-language-processing-nlp/.

33. Словник NLP [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/stinopys/%D1%81%D0%BB%D0%BE%D0%B2%D0%BD%D0%B8%D0%BA-nlp-b0fab1027551.

34. Exploring Natural Language Processing (NLP) in Translation [Електронний ресурс] – Режим доступу до ресурсу: https://www.shaip.com/blog/nlp-in-translation/.

35. What are NLP chatbots and how do they work? [Електронний ресурс] – Режим доступу до ресурсу: https://www.zendesk.com/blog/nlp-chatbot/.

36. Text Generation NLP – Everything You Need To Know / Python Code To Get Started [Електронний ресурс] – Режим доступу до ресурсу: https://spotintelligence.com/2022/12/19/text-generation-nlp/.

37. Fundamentals of Statistical Natural Language Processing [Електронний ресурс] – Режим доступу до ресурсу: https://www.proxet.com/blog/fundamentals-of-statistical-natural-language-processing.

38. Бабак В. П., Марченко М. Є., Фриз. Б. Г. Теорія ймовірностей, випадкові процеси та математична статистика. К.: Техніка, 2004. 288 с.

39. M. Fryz, B. Mlynko, Determination of the characteristic function of discrete-time conditional linear random process and its application, Scientific Journal of TNTU. — Tern.: TNTU, 2023. — Vol 109. — No 1. — P. 16–23.

40. NLP — Word Embedding & GloVe [Електронний ресурс] – Режим доступу до ресурсу: https://jonathan-hui.medium.com/nlp-word-embedding-glove-5e7f523999f6.

41. Transformer Attention Mechanism in NLP [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/transformer-attention-mechanism-in-nlp/.

42. Neural networks and statistical techniques: A review of applications [Електронний ресурс] – Режим доступу до ресурсу: https://www.sciencedirect.com/science/article/abs/pii/S0957417407004952?__cf_chl_tk=nhhMjiamBzsEVB5EwXrkCHoUKIFsLCmLjsR6OU_WZD0-1734599769-1.0.1.1-OvKUGdsIHDUcx3Qujo9P.nL0wOl5J9yuylTXZkZ1RQQ.

43. Ерік Маттес. Пришвидшений курс Python / Ерік Маттес. – Львів: Видавництво Старого Лева, 2021. – 600 с.

44. Python for Machine Learning [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/python-for-machine-learning/.

45. Python for Mathematicians [Електронний ресурс] – Режим доступу до ресурсу: https://www.math.purdue.edu/~bradfor3/ProgrammingFundamentals/Python/.

46. Python Numpy [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/python-numpy/.

47. NumPy Tutorial: Your First Steps Into Data Science in Python [Електронний ресурс] – Режим доступу до ресурсу: https://realpython.com/numpy-tutorial/.

48. JupyterLab: A Next-Generation Notebook Interface [Електронний ресурс] – Режим доступу до ресурсу: https://jupyter.org/.

49. Jupyter for Data Science [Електронний ресурс] – Режим доступу до ресурсу: https://docs.jupyter.org/en/latest/use/use-cases/data_science.html.

50. Andrew W. Trask. Grokking Deep Learning, 2019. – 336 с.

51. Text Generation Using LSTM [Електронний ресурс] – Режим доступу до ресурсу: https://bansalh944.medium.com/text-generation-using-lstm-b6ced8629b03.

52. Методичні вказівки для написання розділу «Безпека життєдіяльності, основи охорони праці» в кваліфікаційних роботах здобувачів освітнього рівня „бакалавр”. Для студентів всіх форм навчання рівень вищої освіти перший (бакалаврський) / укл. : О. Я. Гурик , І. Б. Окіпний. – Тернопіль : ТНТУ імені Івана Пулюя, 2021. - 20 с.

53. Методичні вказівки до лабораторної роботи № 8 з дисципліни „Основи охорони праці” Дослідження факторів, які впливають на освітленість робочих місць у приміщенні / укл. : О.Я. Гурик, О.І. Король, В.С. Сенчишин. - Тернопіль : ТНТУ імені Івана Пулюя, 2013. - 24 с.

54. Вовк Ю. Я. Охорона праці в галузі. Навчальний посібник / Ю. Я. Вовк, І. П. Вовк – Тернопіль: ФОП Паляниця В.А. – 2015. – 172 с.

ДОДАТКИ

Додаток А

Вихідний код фреймворку глибинного навчання

```
import numpy as np

class Tensor (object):

    def __init__(self,data,
                 autograd=False,
                 creators=None,
                 creation_op=None,
                 id=None):

        self.data = np.array(data)
        self.autograd = autograd
        self.grad = None

        if(id is None):
            self.id = np.random.randint(0,1000000000)
        else:
            self.id = id

        self.creators = creators
        self.creation_op = creation_op
        self.children = {}

        if(creators is not None):
            for c in creators:
                if(self.id not in c.children):
                    c.children[self.id] = 1
                else:
                    c.children[self.id] += 1

    def all_children_grads_accounted_for(self):
        for id,cnt in self.children.items():
            if(cnt != 0):
                return False
        return True

    def backward(self,grad=None, grad_origin=None):
        if(self.autograd):
```

```
            if(grad is None):
                grad = Tensor(np.ones_like(self.data))

            if(grad_origin is not None):
                if(self.children[grad_origin.id] == 0):
                    return
                    print(self.id)
                    print(self.creation_op)
                    print(len(self.creators))
                    for c in self.creators:
                        print(c.creation_op)
                    raise Exception("зворотнє поширення помилки
вже виконане")
                else:
                    self.children[grad_origin.id] -= 1

            if(self.grad is None):
                self.grad = grad
            else:
                self.grad += grad

            assert grad.autograd == False

            if(self.creators is not None and
               (self.all_children_grads_accounted_for() or
                grad_origin is None)):

                if(self.creation_op == "add"):
                    self.creators[0].backward(self.grad, self)
                    self.creators[1].backward(self.grad, self)

                if(self.creation_op == "sub"):

self.creators[0].backward(Tensor(self.grad.data), self)

self.creators[1].backward(Tensor(self.grad.__neg__().data), self)

                if(self.creation_op == "mul"):
                    new = self.grad * self.creators[1]
                    self.creators[0].backward(new , self)
                    new = self.grad * self.creators[0]
                    self.creators[1].backward(new, self)
```

```
                if(self.creation_op == "mm"):
                    c0 = self.creators[0]
                    c1 = self.creators[1]
                    new = self.grad.mm(c1.transpose())
                    c0.backward(new)
                    new = self.grad.transpose().mm(c0).transpose()
                    c1.backward(new)

                if(self.creation_op == "transpose"):

self.creators[0].backward(self.grad.transpose())

                if("sum" in self.creation_op):
                    dim = int(self.creation_op.split("_")[1])

self.creators[0].backward(self.grad.expand(dim,

self.creators[0].data.shape[dim]))

                if("expand" in self.creation_op):
                    dim = int(self.creation_op.split("_")[1])
                    self.creators[0].backward(self.grad.sum(dim))

                if(self.creation_op == "neg"):
                    self.creators[0].backward(self.grad.__neg__())

                if(self.creation_op == "sigmoid"):
                    ones = Tensor(np.ones_like(self.grad.data))
                    self.creators[0].backward(self.grad * (self *
(ones - self)))

                if(self.creation_op == "tanh"):
                    ones = Tensor(np.ones_like(self.grad.data))
                    self.creators[0].backward(self.grad * (ones -
(self * self)))

                if(self.creation_op == "index_select"):
                    new_grad =
np.zeros_like(self.creators[0].data)
                    indices_ =
self.index_select_indices.data.flatten()
                    grad_ = grad.data.reshape(len(indices_), -1)
                    for i in range(len(indices_)):
```

```
                    new_grad[indices_[i]] += grad_[i]
                self.creators[0].backward(Tensor(new_grad))

            if(self.creation_op == "cross_entropy"):
                dx = self.softmax_output - self.target_dist
                self.creators[0].backward(Tensor(dx))

    def __add__(self, other):
        if(self.autograd and other.autograd):
            return Tensor(self.data + other.data,
                          autograd=True,
                          creators=[self,other],
                          creation_op="add")
        return Tensor(self.data + other.data)

    def __neg__(self):
        if(self.autograd):
            return Tensor(self.data * -1,
                          autograd=True,
                          creators=[self],
                          creation_op="neg")
        return Tensor(self.data * -1)

    def __sub__(self, other):
        if(self.autograd and other.autograd):
            return Tensor(self.data - other.data,
                          autograd=True,
                          creators=[self,other],
                          creation_op="sub")
        return Tensor(self.data - other.data)

    def __mul__(self, other):
        if(self.autograd and other.autograd):
            return Tensor(self.data * other.data,
                          autograd=True,
                          creators=[self,other],
                          creation_op="mul")
        return Tensor(self.data * other.data)

    def sum(self, dim):
        if(self.autograd):
            return Tensor(self.data.sum(dim),
                          autograd=True,
```

```
                            creators=[self],
                            creation_op="sum_"+str(dim))
        return Tensor(self.data.sum(dim))

    def expand(self, dim,copies):

        trans_cmd = list(range(0,len(self.data.shape)))
        trans_cmd.insert(dim,len(self.data.shape))
        new_data =
self.data.repeat(copies).reshape(list(self.data.shape) +
[copies]).transpose(trans_cmd)

        if(self.autograd):
            return Tensor(new_data,
                            autograd=True,
                            creators=[self],
                            creation_op="expand_"+str(dim))
        return Tensor(new_data)

    def transpose(self):
        if(self.autograd):
            return Tensor(self.data.transpose(),
                            autograd=True,
                            creators=[self],
                            creation_op="transpose")

        return Tensor(self.data.transpose())

    def mm(self, x):
        if(self.autograd):
            return Tensor(self.data.dot(x.data),
                            autograd=True,
                            creators=[self,x],
                            creation_op="mm")
        return Tensor(self.data.dot(x.data))

    def sigmoid(self):
        if(self.autograd):
            return Tensor(1 / (1 + np.exp(-self.data)),
                            autograd=True,
                            creators=[self],
                            creation_op="sigmoid")
        return Tensor(1 / (1 + np.exp(-self.data)))
```

```
    def tanh(self):
        if(self.autograd):
            return Tensor(np.tanh(self.data),
                          autograd=True,
                          creators=[self],
                          creation_op="tanh")
        return Tensor(np.tanh(self.data))

    def index_select(self, indices):

        if(self.autograd):
            new = Tensor(self.data[indices.data],
                         autograd=True,
                         creators=[self],
                         creation_op="index_select")
            new.index_select_indices = indices
            return new
        return Tensor(self.data[indices.data])

    def softmax(self):
        temp = np.exp(self.data)
        softmax_output = temp / np.sum(temp,
                                       axis=len(self.data.shape)-
1,
                                       keepdims=True)
        return softmax_output

    def cross_entropy(self, target_indices):

        temp = np.exp(self.data)
        softmax_output = temp / np.sum(temp,
                                       axis=len(self.data.shape)-
1,
                                       keepdims=True)

        t = target_indices.data.flatten()
        p = softmax_output.reshape(len(t),-1)
        target_dist = np.eye(p.shape[1])[t]
        loss = -(np.log(p) * (target_dist)).sum(1).mean()

        if(self.autograd):
            out = Tensor(loss,
```

```
                            autograd=True,
                            creators=[self],
                            creation_op="cross_entropy")
                out.softmax_output = softmax_output
                out.target_dist = target_dist
                return out

            return Tensor(loss)


        def __repr__(self):
            return str(self.data.__repr__())

        def __str__(self):
            return str(self.data.__str__())

    class Layer(object):

        def __init__(self):
            self.parameters = list()

        def get_parameters(self):
            return self.parameters


    class SGD(object):

        def __init__(self, parameters, alpha=0.1):
            self.parameters = parameters
            self.alpha = alpha

        def zero(self):
            for p in self.parameters:
                p.grad.data *= 0

        def step(self, zero=True):

            for p in self.parameters:

                p.data -= p.grad.data * self.alpha

                if(zero):
                    p.grad.data *= 0
```

```
class Linear(Layer):

    def __init__(self, n_inputs, n_outputs, bias=True):
        super().__init__()

        self.use_bias = bias

        W = np.random.randn(n_inputs, n_outputs) *
np.sqrt(2.0/(n_inputs))
        self.weight = Tensor(W, autograd=True)
        if(self.use_bias):
            self.bias = Tensor(np.zeros(n_outputs), autograd=True)

        self.parameters.append(self.weight)

        if(self.use_bias):
            self.parameters.append(self.bias)

    def forward(self, input):
        if(self.use_bias):
            return
input.mm(self.weight)+self.bias.expand(0,len(input.data))
        return input.mm(self.weight)


class Sequential(Layer):

    def __init__(self, layers=list()):
        super().__init__()

        self.layers = layers

    def add(self, layer):
        self.layers.append(layer)

    def forward(self, input):
        for layer in self.layers:
            input = layer.forward(input)
        return input

    def get_parameters(self):
```

```
        params = list()
        for l in self.layers:
            params += l.get_parameters()
        return params


class Embedding(Layer):

    def __init__(self, vocab_size, dim):
        super().__init__()

        self.vocab_size = vocab_size
        self.dim = dim

        self.weight = Tensor((np.random.rand(vocab_size, dim) -
0.5) / dim, autograd=True)

        self.parameters.append(self.weight)

    def forward(self, input):
        return self.weight.index_select(input)


class Tanh(Layer):
    def __init__(self):
        super().__init__()

    def forward(self, input):
        return input.tanh()


class Sigmoid(Layer):
    def __init__(self):
        super().__init__()

    def forward(self, input):
        return input.sigmoid()


class CrossEntropyLoss(object):

    def __init__(self):
        super().__init__()
```

```
    def forward(self, input, target):
        return input.cross_entropy(target)


class RNNCell(Layer):

    def __init__(self, n_inputs, n_hidden, n_output,
activation='sigmoid'):
        super().__init__()

        self.n_inputs = n_inputs
        self.n_hidden = n_hidden
        self.n_output = n_output

        if(activation == 'sigmoid'):
            self.activation = Sigmoid()
        elif(activation == 'tanh'):
            self.activation == Tanh()
        else:
            raise Exception("Non-linearity not found")

        self.w_ih = Linear(n_inputs, n_hidden)
        self.w_hh = Linear(n_hidden, n_hidden)
        self.w_ho = Linear(n_hidden, n_output)

        self.parameters += self.w_ih.get_parameters()
        self.parameters += self.w_hh.get_parameters()
        self.parameters += self.w_ho.get_parameters()

    def forward(self, input, hidden):
        from_prev_hidden = self.w_hh.forward(hidden)
        combined = self.w_ih.forward(input) + from_prev_hidden
        new_hidden = self.activation.forward(combined)
        output = self.w_ho.forward(new_hidden)
        return output, new_hidden

    def init_hidden(self, batch_size=1):
        return Tensor(np.zeros((batch_size,self.n_hidden)),
autograd=True)

class LSTMCell(Layer):
```

```
    def __init__(self, n_inputs, n_hidden, n_output):
        super().__init__()

        self.n_inputs = n_inputs
        self.n_hidden = n_hidden
        self.n_output = n_output

        self.xf = Linear(n_inputs, n_hidden)
        self.xi = Linear(n_inputs, n_hidden)
        self.xo = Linear(n_inputs, n_hidden)
        self.xc = Linear(n_inputs, n_hidden)

        self.hf = Linear(n_hidden, n_hidden, bias=False)
        self.hi = Linear(n_hidden, n_hidden, bias=False)
        self.ho = Linear(n_hidden, n_hidden, bias=False)
        self.hc = Linear(n_hidden, n_hidden, bias=False)

        self.w_ho = Linear(n_hidden, n_output, bias=False)

        self.parameters += self.xf.get_parameters()
        self.parameters += self.xi.get_parameters()
        self.parameters += self.xo.get_parameters()
        self.parameters += self.xc.get_parameters()

        self.parameters += self.hf.get_parameters()
        self.parameters += self.hi.get_parameters()
        self.parameters += self.ho.get_parameters()
        self.parameters += self.hc.get_parameters()

        self.parameters += self.w_ho.get_parameters()

    def forward(self, input, hidden):

        prev_hidden = hidden[0]
        prev_cell = hidden[1]

        f = (self.xf.forward(input) +
self.hf.forward(prev_hidden)).sigmoid()
        i = (self.xi.forward(input) +
self.hi.forward(prev_hidden)).sigmoid()
        o = (self.xo.forward(input) +
self.ho.forward(prev_hidden)).sigmoid()
```

```
        g = (self.xc.forward(input) +
self.hc.forward(prev_hidden)).tanh()
        c = (f * prev_cell) + (i * g)

        h = o * c.tanh()

        output = self.w_ho.forward(h)
        return output, (h, c)

    def init_hidden(self, batch_size=1):
        init_hidden = Tensor(np.zeros((batch_size,self.n_hidden)),
autograd=True)
        init_cell = Tensor(np.zeros((batch_size,self.n_hidden)),
autograd=True)
        init_hidden.data[:,0] += 1
        init_cell.data[:,0] += 1
        return (init_hidden, init_cell)
```

Додаток Б

Вихідний код розробленої мережі

```
from dl_framework import *
import sys,random,math
import numpy as np
import sys
np.random.seed(0)

f = open('text.txt','r')
raw = f.read()
f.close()

vocab = list(set(raw))
word2index = {}
for i,word in enumerate(vocab):
    word2index[word]=i
indices = np.array(list(map(lambda x:word2index[x], raw)))

embed = Embedding(vocab_size=len(vocab),dim=512)
model = LSTMCell(n_inputs=512, n_hidden=512, n_output=len(vocab))
model.w_ho.weight.data *= 0

criterion = CrossEntropyLoss()
optim = SGD(parameters=model.get_parameters() +
embed.get_parameters(), alpha=0.05)

def generate_sample(n=30, init_char=' '):
    s = ""
    hidden = model.init_hidden(batch_size=1)
    input = Tensor(np.array([word2index[init_char]]))
    for i in range(n):
        rnn_input = embed.forward(input)
        output, hidden = model.forward(input=rnn_input,
hidden=hidden)
        output.data *= 15
        temp_dist = output.softmax()
        temp_dist /= temp_dist.sum()

        m = output.data.argmax()
        c = vocab[m]
        input = Tensor(np.array([m]))
```

```
        s += c
    return s

batch_size = 16
bptt = 25
n_batches = int((indices.shape[0] / (batch_size)))

trimmed_indices = indices[:n_batches*batch_size]
batched_indices = trimmed_indices.reshape(batch_size,
n_batches).transpose()

input_batched_indices = batched_indices[0:-1]
target_batched_indices = batched_indices[1:]

n_bptt = int(((n_batches-1) / bptt))
input_batches =
input_batched_indices[:n_bptt*bptt].reshape(n_bptt,bptt,batch_size
)
target_batches =
target_batched_indices[:n_bptt*bptt].reshape(n_bptt, bptt,
batch_size)
def train(iterations=400, min_loss=1000):
    for iter in range(iterations):
        total_loss = 0
        n_loss = 0

        hidden = model.init_hidden(batch_size=batch_size)
        batches_to_train = len(input_batches)
        for batch_i in range(batches_to_train):

            hidden = (Tensor(hidden[0].data, autograd=True),
Tensor(hidden[1].data, autograd=True))

            losses = list()
            for t in range(bptt):
                input = Tensor(input_batches[batch_i][t],
autograd=True)
                rnn_input = embed.forward(input=input)
                output, hidden = model.forward(input=rnn_input,
hidden=hidden)

                target = Tensor(target_batches[batch_i][t],
autograd=True)
```

```
                batch_loss = criterion.forward(output, target)

                if(t == 0):
                    losses.append(batch_loss)
                else:
                    losses.append(batch_loss + losses[-1])

            loss = losses[-1]

            loss.backward()
            optim.step()
            total_loss += loss.data / bptt
            epoch_loss = np.exp(total_loss / (batch_i+1))
            if(epoch_loss < min_loss):
                min_loss = epoch_loss
                print()

            log = "\r Iteration:" + str(iter)
            log += " - Alpha:" + str(optim.alpha)[0:5]
            log += " - Batch
"+str(batch_i+1)+"/"+str(len(input_batches))
            log += " - Min Loss:" + str(min_loss)[0:5]
            log += " - Loss:" + str(epoch_loss)
            if(batch_i == 0):
                log += "\n" + generate_sample(n=70,
init_char='t').replace("\n"," ")
            if(batch_i % 1 == 0):
                sys.stdout.write(log)
        optim.alpha *= 0.99
train(130, min_loss = 1000)
```

Додаток В

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Тернопільський національний технічний університет імені Івана Пулюя (Україна)
Університет імені П'єра і Марії Кюрі (Франція)
Маріборський університет (Словенія)
Технічний університет у Кошице (Словаччина)
Вільнюський технічний університет ім. Гедимінаса (Литва)
Наукове товариство ім. Т.Шевченка

АКТУАЛЬНІ ЗАДАЧІ СУЧАСНИХ ТЕХНОЛОГІЙ

Збірник
тез доповідей

XIII Міжнародної науково-практичної конференції молодих учених та студентів
11-12 грудня 2024 року

УКРАЇНА
ТЕРНОПІЛЬ – 2024

А43
Актуальні задачі сучасних технологій : зб. тез доповідей XIII міжнар. наук.-практ. конф. Молодих учених та студентів, (Тернопіль, 11-12 грудня 2024) / М-во освіти і науки України, Терн. націон. техн. ун-т ім. І. Пулюя [та ін.]. – Тернопіль: ФОП Паляниця В. А., 2024. – 543.
ISBN 978-617-7875-88-7

ПРОГРАМНИЙ КОМІТЕТ

Голова: Митник Микола Мирославович – к.т.н., доцент, Ректор ТНТУ ім. І. Пулюя. (Україна)

Заступник голови: Марущак Павло Орестович – д.т.н., проф. ТНТУ ім. І. Пулюя. (Україна)

Вчений секретар: Гладьо Сергій Володимирович – к.т.н., ст.викл. ТНТУ ім. І. Пулюя. (Україна)

Члени: Вухерер Т. – професор факультету інженерної механіки Маріборського університету (Словенія); Вінаш Я. – професор кафедри технології металів Технічного університету у Кошице (Словаччина); Прентковскіс О. – декан факультету Вільнюського технічного університету ім. Гедимінаса (Литва); Стахович Ф. – завідувач кафедри обробки матеріалів тиском Жешувського політехнічного університету ім. Лукасевича (Польща); Андрейків О. – д.т.н., професор кафедри механіки Львівського національного університету ім. І. Франка, член-корр. НАН України.

Адреса оргкомітету:
ТНТУ ім. І. Пулюя, м. Тернопіль, вул. Руська, 56, 46001,
тел. 0971640355, факс (0352) 255798
E-mail: **sergiiglado1@gmail.com**
Редагування, оформлення, верстка: Гладьо С.В.

СЕКЦІЇ КОНФЕРЕНЦІЇ, ЯКІ ПРЕДСТВЛЕНІ В ЗБІРНИКУ

- фізико-технічні основи розвитку нових технологій;
- нові матеріали, міцність і довговічність елементів конструкцій;
- сучасні технології в будівництві, машино- та приладобудуванні;
- сучасні технології на транспорті;
- електротехніка та енергозбереження;
- фундаментальні проблеми харчових, біо- та нанотехнологій;
- економічні та соціальні аспекти нових технологій;
- комп'ютерно-інформаційні технології та системи зв'язку.

Матеріали XIII Міжнародної науково-практичної конференції молодих учених та студентів
«АКТУАЛЬНІ ЗАДАЧІ СУЧАСНИХ ТЕХНОЛОГІЙ» – Тернопіль, 11-12 грудня 2024 року

УДК 004.8

І.М. Климко
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

ОПТИМІЗАЦІЯ НАВЧАННЯ LSTM-МОДЕЛІ ДЛЯ ГЕНЕРАЦІЇ ТЕКСТУ НА РІВНІ СИМВОЛІВ

I. M. Klymko

OPTIMIZATION OF LSTM MODEL TRAINING FOR CHARACTER-LEVEL TEXT GENERATION

У цій роботі представлено підхід до навчання LSTM-моделі для генерації тексту на рівні символів. Основною метою є покращення якості генерованого тексту шляхом оптимізації процесу навчання. Для цього текстовий корпус зчитується з файлу та перетворюється у послідовність індексів, що відповідають унікальним символам. Створюється словник символів для перетворення тексту в числові дані.

Використовується шар вбудовування для перетворення індексів символів у векторні представлення, а LSTM-комірка налаштовується для обробки послідовностей векторів та генерації наступних символів. В якості функції втрат використовується крос-ентропія, яка вимірює різницю між передбаченими та фактичними символами, а оптимізатор стохастичного градієнтного спуску використовується для оновлення параметрів моделі.

Модель навчається на текстах Шекспіра (рис. 1), що дозволяє їй вивчати складні мовні структури та стилістичні особливості класичної літератури. Навчання відбувається на пакетах даних, де кожен пакет містить послідовності символів. Застосовується метод зворотного поширення помилок через час для ефективного оновлення параметрів. Після кожної ітерації навчання модель генерує зразки тексту для оцінки якості навчання, використовуючи температурне масштабування для контролю за випадковістю генерованих символів.

```
BIONDELLO:
Marry, that it may not pray their patience.'

KING LEAR:
The instant common maid, as we may less be
a brave gentleman and joiner: he that finds us with wax
And owe so full of presence and our fooder at our
staves. It is remorsed the bridal's man his grace
for every business in my tongue, but I was thinking
that he contends, he hath respected thee.
```

Рисунок 1. Фрагмент навчального тексту

Висновок. Результати показують, що модель демонструє здатність генерувати послідовності символів, що імітують структуру та стиль вихідного тексту (рис. 2). Оптимізація параметрів навчання, таких як швидкість навчання та регуляризація, значно покращує якість генерованого тексту. Представлений підхід до оптимізації навчання LSTM-моделі для генерації тексту на рівні символів показує перспективні результати. Подальші дослідження можуть зосередитися на вдосконаленні архітектури моделі та методів регуляризації для досягнення кращих результатів.

```
BRUTUS:
Youll will not winge not lore the bll we wnee tin the sell my lord, not gre wet no save fev Whall we
sell my lord, you gr will not winge not lore th spees to liking to trumpin the se will ere see wit be
well dest tiemy be will pet-ll me to lousell e tespill weepe tres swee twill wise to ll wing in the s
th the well will nee in eve clordere tere bake thee be ou will sisee time in the king, in the louse s
ul will eee will wing end see twhere be for te will pone nee till my, will not with the weepn the de
there be will we well de.

All not s not lord ou come of to come for the e my lord, will not to tur not lovere time thee so ve l
ll e will not winge neme the bll not good Thall we we to she will look trume be will not with the wee
l with the sill e te the suin sin te the nevery eye will sisee will not with to gear to  or will e to
ot ell winge of to seve ll teking in the seemy cll tre powith tieme ting quightree twill eee tin eve
dispe the will dere the seve ble Dor the will sisee there be will sish e will love endll ness,
Bull well e trese where be ill e tespe the seve bll tertsell expo theese will see
And will ere to likill lore tin evere the barrius ould will petire the will lor see iell wis  ut it l

DUKENIO:

MARKILLAKN:
All expill we we in the dare to cllow eor trull wite tin he will d tre to the re with e will sise twe
portter'd will not s of there be will not with the seem barre time twe would will sise twees will sis
tise tweepo the woull dispe the seeming enen thenee to good will sispe poteit my lord, ll wore the wi
then sing every be will dist pormink his come ente tilll sise twe brind the king in the king, in tees
lord, not gre with truse will eOLU5:
```

Рисунок 2. Фрагмент згенерованого моделлю тексту

Література

1. Литвин В. В. Глибинне навчання: навч. посіб. / В. В. Литвин, Р. М. Пелещак, В. А. Висоцька; М-во освіти і науки України, Нац. ун-т «Львів. політехніка». – Львів : Вид-во Львів. політехніки, 2021. – 264 с.: іл.

2. Foster D. Generative Deep Learning, 2nd Edition / David Foster., 2023

3. Шаховська Н. Б. Системи штучного інтелекту : навч. посіб. / Н. Б. Шаховська, Р. М. Камінський, О. Б. Вовк; М-во освіти і науки України, Нац. ун-т «Львів. політехніка». – Львів : Вид-во Львів. політехніки, 2018. – 392 с.: іл.

4. Шаховська Н. Б. Системи штучного інтелекту : навч. посіб. / Н. Б. Шаховська, Р. М. Камінський, О. Б. Вовк; М-во освіти і науки України, Нац. ун-т «Львів. політехніка». – Львів : Вид-во Львів. політехніки, 2018. – 392 с.: іл.

5. Карташов В.В. Методичні вказівки до виконання лабораторних робіт з курсу «Основи штучного інтелекту», Тернопіль, ТНТУ, 2017. 63 с.

6. I. Goodfellow. Deep Learning (Adaptive Computation and Machine Learning series) / I. Goodfellow, Y. Bengio, A. Courville., 2016.

7. What is text generation? [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/topics/text-generation.

8. NLP using RNN — Can you be the next Shakespeare? [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/analytics-vidhya/nlp-using-rnn-can-you-be-the-next-shakespeare-27abf9af523.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
імені ІВАНА ПУЛЮЯ
НАУКОВЕ ТОВАРИСТВО ім. ШЕВЧЕНКА

М А Т Е Р І А Л И

XII НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

18–19 грудня 2024 року

ТЕРНОПІЛЬ
2024

УДК 001
М34

ПРОГРАМНИЙ КОМІТЕТ

ОРГАНІЗАЦІЙНИЙ КОМІТЕТ

Матеріали XII науково-технічної конфції «Інформаційні моделі, системи та технології» Тернопільського національного технічного університету імені Івана Пулюя, (Тернопіль, 18–19 грудня 2024 р.). – Тернопіль : Тернопільський національний технічний університет імені Івана Пулюя, 2024. – 238 с.

М34

Адреса оргкомітету: ТНТУ ім. І. Пулюя, м. Тернопіль, вул. Руська, 56, 46001,
тел. (0352) 52-41-33, факс (0352) 25-49-83.
E-mail: conffis2024@gmail.com
Редагування, оформлення та верстка: Надія Крива

СЕКЦІЇ КОНФЕРЕНЦІЇ, ЯКІ ПРЕДСТВЛЕНІ В ЗБІРНИКУ

- Математичне моделювання
- Інформаційні системи та технології, кібербезпека
- Комп'ютерні системи та мережі
- Програмна інженерія та моделювання складних розподілених систем
- Новітні фізико-технічні та освітні технології

В збірнику надруковано тези доповідей XII науково-технічної конференції «Інформаційні моделі, системи та технології» (Тернопіль, 18–19 грудня 2024 р.) за такими науковими напрямками: математичне моделювання; інформаційні системи та технології, кібербезпека; комп'ютерні системи та мережі; програмна інженерія та моделювання складних розподілених систем; новітні фізико-технічні та освітні технології.

Розрахований на науковців, викладачів та студентів вузів.

За зміст тез та дотримання норм академічної доброчесності відповідальність несе автор.

УДК 004.8
Б. Б. Млинко, к.т.н., доц.; І. М. Климко
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

ЗАСТОСУВАННЯ МЕТОДІВ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ ОБРОБКИ ПРИРОДНЬОЇ МОВИ ЗАСОБАМИ ГЛИБИННОГО НАВЧАННЯ

UDC 004.8
B. B. Mlynko, Ph.D., Assoc. Prof., I. M. Klymko

APPLICATION OF ARTIFICIAL INTELLIGENCE METHODS FOR NATURAL LANGUAGE PROCESSING USING DEEP LEARNING TECHNIQUES

Генерація тексту на основі методів глибинного навчання є одним із ключових напрямів у сучасній обробці природної мови. Враховуючи зростаючі вимоги до автоматизованого створення текстового контенту, актуальним стає завдання розробки моделей, здатних створювати послідовності символів із врахуванням синтаксичних, семантичних і структурних особливостей тексту. LSTM мережі показують високу ефективність у задачах моделювання послідовностей завдяки здатності запам'ятовувати довгострокові залежності. Однак основними викликами при їх застосуванні залишаються оптимізація навчання, подолання ефекту "забування" контексту та забезпечення балансу між узгодженістю і різноманітністю згенерованого тексту.

Метою дослідження було вдосконалити процес генерації тексту за допомогою LSTM-мереж шляхом впровадження методів оптимізації параметрів, поліпшення обробки навчальних даних та адаптації підходів до семплювання, що забезпечує генерування тексту із реалістичною структурою і змістом.

Для досягнення поставленої мети було розроблено експериментальну LSTM-модель, що працює на рівні символів. На етапі підготовки даних виконано токенізацію тексту на символи та побудову словника, що дозволяє перетворювати текст у послідовності числових індексів. Модель включає шар вбудовування, який знижує розмірність входу, а також LSTM-комірки, що обробляють дані послідовно. Навчання здійснювалося за допомогою алгоритму зворотного поширення похибки через час, при цьому дані поділено на батчі фіксованої довжини для ефективного опрацювання.

У процесі оптимізації параметрів моделі використано стохастичний градієнтний спуск із поступовим зменшенням швидкості навчання, що забезпечує стабільне сходження. Для запобігання перенавчанню та зниження ризику переваги локальних мінімумів застосовано методи регуляризації, такі як обмеження норми ваг та масштабування градієнтів. Генерація тексту здійснювалася на основі семплювання із використанням softmax-розподілу. Цей підхід дозволив регулювати баланс між детермінованістю вибору символу та різноманітністю вихідного тексту.

Результати експериментів продемонстрували, що оптимізація параметрів моделі та впровадження регуляризації дозволили покращити якість згенерованого тексту. Модель виявила здатність підтримувати базові патерни англійської мови, відтворювати структуру речень та зберігати контекст на рівні декількох символів. Під час навчання спостерігалося поступове зниження перплексії, що свідчить про ефективне опрацювання довгострокових залежностей.

Література

1. I. Goodfellow. Deep Learning (Adaptive Computation and Machine Learning series) / I. Goodfellow, Y. Bengio, A. Courville., 2016.