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: Моделювання української мови з використанням нейронних мереж

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

(підпис) Хом'як А.С. (прізвище та ініціали)

Керівник (підпис) Фриз М.Є. (прізвище та ініціали)

Нормоконтроль (підпис) Мацюк О.В. (прізвище та ініціали)

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

Рецензент (підпис) Осухівська Г.М. (прізвище та ініціали)

Тернопіль
2022

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)
«26» травня 2022 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Магістр
(назва освітнього ступеня)

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

студенту Хом'яку Андрію Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Моделювання української мови з використанням нейронних мереж

Керівник роботи Фриз Михайло Євгенович, к.т.н, доцент кафедри КН
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «28» жовтня 2021 року № *4/7-909*

2. Термін подання студентом завершеної роботи

3. Вихідні дані до роботи наукові джерела щодо теми кваліфікаційної роботи

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ, 1 Аналітичний огляд в області досліджень, 2 Методологія моделювання української мови, 3 Розробка нейромережевої мовної моделі, 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	Дмитроца Л.П., доцент		
Безпека в надзвичайних ситуаціях	Клепчик В.М., проректор		

7. Дата видачі завдання 27 вересня 2021

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	28.10.21-30.10.21	*Виконано*
2.	Аналіз літературних джерел	31.10.21-07.11.21	*Виконано*
3.	Обґрунтування актуальності дослідження	08.11.21-20.11.21	*Виконано*
4.	Аналіз предмету дослідження та предметної області	21.11.21-07.01.22	*Виконано*
5.	Оформлення розділу «Аналітичний огляд в області	08.01.22-20.01.22	*Виконано*
	досліджень»		
6.	Оформлення розділу «Методологія моделювання	21.01.22-28.02.22	*Виконано*
	української мови»		
7.	Оформлення розділу «Розробка нейромережевої мовної	01.03.22-29.03.22	*Виконано*
	моделі»		
8.	Виконання завдання з підрозділу «Охорона праці»	30.03.22-15.04.22	*Виконано*
9.	Виконання завдання з підрозділу «Безпека в	16.04.22-29.04.22	*Виконано*
	назвичайних ситуаціях»		
10.	Оформлення кваліфікаційної роботи	30.04.22-05.05.22	*Виконано*
11.	Нормоконтроль	06.05.22	*Виконано*
12.	Перевірка на плагіат	10.05.22	*Виконано*
13.	Попередній захист кваліфікаційної роботи	12.05.22	*Виконано*
14.	Захист кваліфікаційної роботи	27.05.22	

Студент _____ Хом'як А.С.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Фриз М.Є.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Моделювання української мови з використанням нейронних мереж // Кваліфікаційна робота освітнього рівня «Магістр» // Хом’як Андрій Сергійович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп’ютерно-інформаційних систем і програмної інженерії, кафедра комп’ютерних наук, група СНнм-61 // Тернопіль, 2022 // с. – , рис. – , додат. – , бібліогр. – 106.

Ключові слова: мовна модель, нейронна мережа, штучний інтелект, генерація тексту, обробка природної мови.

Проведено ґрунтовний аналіз публікацій на тему моделювання природної мови та української зокрема, зібрано інформацію про алгоритми та нейромережеві архітектури, які дозволяють це зробити. Розглянуто вимоги до процесу тренування нейронних мереж - від отримання даних до процесу оптимізації обраної моделі, а також інструменти, які дозволяють ці вимоги задовольнити. Запропоновано використовувати архітектуру GPT-2 для моделювання української мови.

Розроблено програмний код, для проведення експериментів з нейронними мережами в галузі моделювання природних мов, а також генерування текстів на їх основі.

Натреновано нейронну мовна модель, що показує кращі результати, ніж схожі публікації, які використовували аналогічні обчислювальні потужності при тренуванні, що доводить перспективність використання архітектури Трансформер для задач обробки і генерації природної мови.

ANNOTATION

Modeling of the Ukrainian language using neural networks // Qualification work of the educational level "Master" // Khomiak Andrii // Ternopil National Technical University named after Ivan Puluj, Faculty of Computer Information Systems and Software Engineering, Department of Computer Science, group SNnm-61 // Ternopil, 2022 // p. - , fig. - , appendix. - , bibliogr. - .

Key words: language model, neural network, artificial intelligence, text generation, natural language processing.

A thorough analysis of publications on the topic of modeling natural language and Ukrainian in particular, collected information on algorithms and neural network architectures that allow this. The requirements for the process of training neural networks - from the acquisition of the data to the process of optimizing the selected model, as well as tools that meet these requirements. It is proposed to use the GPT-2 architecture for modeling the Ukrainian language.

Developed software code for experiments with neural networks in the field of natural language modeling, as well as generating texts based on them.

A neural language model has been trained, which shows better results than similar publications that used similar computing power in training, which proves the viability of using the Transformer architecture for natural language processing and generation.

СПИСОК СКОРОЧЕНЬ

API (англ. applied programming interface) – прикладний програмний інтерфейс

ASR (англ. Automatic Speech Recognition) – автоматичне розпізнавання мови

BERT, BART, RoBERTa, ALBERT, DistillBERT, XLNET, DistillGPT, T5, DALL-E, DALL-E 2, CLIP – назви нейромережевих архітектур

BPE (англ. Byte-Pair Encoding) – алгоритм кодування пар байтів

CBOW (англ. Continuous Bag-Of-Words) – алгоритм «послідовний мішок слів»

CLIP – нейромережа для обробки тексту та зображень

CPU (англ. Central Processing Unit) – центральний процесор

CV (англ. Computer Vision) – машинний зір

DVC – система контролю версій даних

GPT, GPT-2, GPT-3 (англ. Generative Pre-trained Transformer) – сімейство нейромережевих архітектур на основі Transformer

GPU (англ. Graphics Processing Unit) – графічний процесор

gRPC – протокол виклику відалених процедур

GRU (англ. Gated Recurrent Unit) – нейромережева архітектура

HTML – мова розмітки гіпертексту

HTTP (англ. hypertext transfer protocol) – протокол передачі гіпертексту

IDE (англ. Integrated Development Environment) – інтегроване середовище розробки

JIT (англ. just in time) – компіляція «на льоту»

LM (англ. Language Model)– мовна модель

LSTM (англ. Long Short-Term Memory) – нейромережева архітектура довгої короткочасної пам’яті

NLP (англ. Natural Language Processing)– обробка природної мови

NSP (англ. Next Sentence Prediction)– передбачення наступного речення

OOV (англ. out-of-vocabulary) – слово поза словником

POS (англ. Part Of Speech) – частина мови

ReLU (англ. Rectified Linear Unit) – назва функції випрямленої лінійної активації

REST (англ. Representative State Transfer) – архітектура мережевого обміну даними через репрезентативну передачу стану

RNN (англ. Recurrent Neural Network) – рекурентна нейрона мережа

SGD (англ. Stochastic Gradient Descent) – алгоритм стохастичного градієнтного спуску

SOTA (англ. state-of-the-art) – найсучасніший підхід

TAPAS (англ. Table Parsing) – нейронна мережа для опрацювання табличних даних

TPU (англ. Tensor Processing Unit) – тензорний процесор

YAML – текстова мова розмітки

ЗМІСТ

ВСТУП

Актуальність теми роботи. Все частіше зустрічаються застосування алгоритмів обробки природної мови - «розумні» клавіатури з автоматичним виправленням помилок, аналіз настроїв користувачів соцмереж щодо певного бренду, чат-боти, системи розпізнавання усної мови в аудіозаписах, машинний переклад та інтелектуальні асистенти. Більшість з цих застосунків покладаються на розуміння мови, її семантики та граматики. Перспективність створення таких систем для певної мови пропорційна до кількості користувачів, які нею користуються, тому більшість застосунків орієнтовані на мови глобального вжитку – англійську, іспанську, хінді, китайську тощо. Оскільки Україна рухається в напрямку цифровізації суспільства, все частіше виникатиме потреба в створенні систем, які покладатимуться на розуміння української, тому створення нових моделей та методів аналізу і опрацювання цієї мови є актуальною науковою проблемою.

Метою дослідження є вивчення можливості покращення розуміння української мови автоматичними системами шляхом використання сучасних нейромережевих архітектур.

Щоб досягти цієї мети, слід виконати такі **завдання**:

- здійснити аналіз наукових публікацій на цю тему;
- проаналізувати існуючі методи моделювання мов;
- розробити програмне забезпечення на основі розробленої методології.

Об’єктом дослідження є процес опрацювання природної мови.

Предмет дослідження – використання нейромережевих архітектур для моделювання української мови..

Наукова новизна роботи полягає у застосуванні великого корпусу текстів української мови OSCAR-uk в поєднанні з архітектурою GPT-2 для тренування великої мовної моделі, що покращило основну метрику таких моделей - перплексність, використовуючи при цьому аналогічну до попередніх підходів кількість обчислювальних ресурсів.

Практичне значення одержаних результатів: під час виконання кваліфікаційної роботи було розроблено сукупність програмних рішень для тренування та оцінки якості нейронних мовних моделей, що дозволяє проводити експерименти щодо дослідження ефективності різних нейромережевих архітектур а також генерувати на їх основі тексти з довільного контексту.

Апробація результатів магістерської роботи. Окремі результати роботи представлені на 3 наукових конференціях:

1. IX Науково-технічна конференція “Інформаційні моделі, системи та технології” на теми “Аналіз нейромережевих моделей природної мови” та “Використання нейронних мереж для дослідження стану рослин в розумних теплицях” (див. додаток А).

2. MoMLeT+DS 2021: 3rd International Workshop on Modern Machine Learning Technologies and Data Science на тему “Usage of Artificial Intelligence Systems and Working with the Neural Network in Assessing the Condition of Plants in Smart Greenhouses” (див. додаток Б)

3. IV Міжнародна студентська науково-технічна конференція «Природничі та гуманітарні науки. Актуальні питання” на тему “Використання системи штучного інтелекту при оцінюванні стану рослин в розумних теплицях” (див. додаток В).

1 АНАЛІТИЧНИЙ ОГЛЯД В ОБЛАСТІ ДОСЛІДЖЕНЬ

1.1 Огляд задачі моделювання мови

Мовна модель (англ. Language Model, LM) - математична модель, яка дозволяє отримати імовірнісний розподіл для всіх можливих послідовностей токенів (слів, лексем) для певної мови - тобто поставити у відповідність кожному тексту (словосполученню, реченню, абзацу тощо) імовірність того, що він зустрінеться в цій мові. Зокрема, така модель дозволяє серед кількох можливих варіантів послідовностей визначити, яка з них є найімовірнішою.

Ця модель є корисною для багатьох задач в галузі обробки природної мови (англ. Natural Language Processing, NLP), а саме розпізнавання мови (англ. speech recognition), машинного перекладу, аналізу настроїв, сумаризації тексту, систем відповідей на питання, виправлення помилок в тексті та генерації мови.

1.1.1 Класичні методи

Одні з перших застосувань для мовних моделей були пов'язані з автоматичним розпізнаванням усної мови (англ. Automatic Speech Recognition, ASR). У 1980-х широкого застосування отримали ASR системи, які складалися з двох компонентів. Перший - акустичний модуль - займається аналізом звукового ряду, відбором слів-кандидатів та оцінкою їх схожості зі звуковим рядом. Другий - власне мовна модель - аналізує слова кандидати та оцінює лінгвістичну ймовірність того що слово-кандидат може бути продовженням існуючого контексту з попередніх слів. Отримані таким чином для кожного слова-кандидата ймовірності (акустична і лінгвістична) перемножуються,

даючи кінцевий критерій оцінки для кожного слова-кандидата [1]. Дослідники використовували підходи засновані на n-грамах та прихованих моделях Маркова [2] щоб оцінювати імовірність послідовностей токенів. Такі моделі використовували простий статистичний аналіз текстів через розбиття на групи слів фіксованої довжини та обчислення ймовірностей зустріти одну групу після іншої, що є відносно простим у імплементації, однак це накладає обмеження на розмір таких груп через значні вимоги до обчислювальних ресурсів, а також не дозволяє аналізувати семантику слововживань через поверхневість простого частотного аналізу.

1.1.2 Нейронні мережі

З розвитком обчислювальних потужностей, штучні нейронні мережі [3] показали себе як ефективний спосіб вирішення когнітивно-складних задач, таких як машиний зір[4], класифікація[5], регресія тощо - тобто таких які не мають простих аналітичних рішень. Оскільки отримані моделі показали себе як рівні або кращі в кількох задачах, ніж людські експерти [6], природно що НМ стали перспективними напрямками розвитку в обробці природної мови, зокрема для виявлення спаму [7], визначення настроїв[8] тощо.

У своїй роботі A Neural Probabilistic Language Model [9] автори запропонували нейронні мережі як вирішення "прокляття розмірності" (the curse of dimensionality), і показали що нейронні мережі підходять як для вивчення векторних репрезентацій слів (word embeddings) так і для функції розподілу ймовірності послідовностей.

Однак ця та інші аналогічні реалізації зіткнулася зі значними обчислювальними обмеженнями - останній етап в цій архітектурі (шар $softmax$), який отримує на вхід активації передостаннього шару та видає

вектор з розподілом ймовірності наступного слова по всьому словнику, є обчислювально дорогим. А саме, для мережі, яка оперує словником розміром k та має вихідний шар розміром d, складність обчислення $softmax$ - $O(dk)$, тож для великих словників (десятки тисяч токенів) це обчислення займає більшість часу тренування.

Робота Collobert, R., & Weston, J. (2008) [10] показала, що вивчені векторні репрезентації (за умови навчання на достатньо великому обсязі даних) несуть синтаксичну і семантичну інформацію, і можуть бути використані в суміжних задачах. В цій же роботі автори запропонували нову функцію витрат, яка допомагає вирішити проблему з затратами softmax та cross-entropy loss.

1.1.3 Рекурентні нейронні мережі та seq2seq

Рекурентні нейронні мережі (RNN) прийшли на заміну звичайним багатошаровим перцептронам, і дозволили використати послідовну природу тексту в архітектурі мережі. Чудовою демонстрацією можливостей цього підходу є стаття від Andrej Karpathy під назвою The Unreasonable Effectiveness of Recurrent Neural Networks [11], в якій автор демонструє застосування RNN для моделювання текстів Шекспіра, статей Вікіпедії, наукових праць з алгебраїчної геометрії у форматі LaTeX та сирцевого коду ядра Linux.

Принциповою відмінністю між рекурентними моделями і багатошаровими перцептронами є наявність ітеративних активацій одного і того ж шару, який бере вихід з попередньої ітерації на вхід поточної разом з вектором поточного елементу послідовності, як наведено на рисунку 1.2.

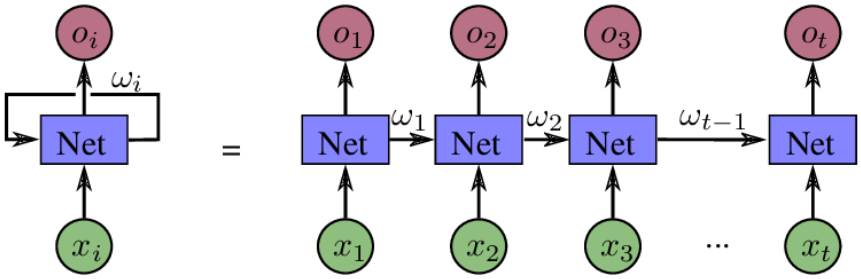


Рисунок 1.1 - Архітектура рекурентної нейронної мережі, де x_i, w_i, o_i - вхідний вектор, активація та вихід на i-ій ітерації [12]

RNN і їх модифікацій (Gated Recurrent Unit [13] та Long-Short Term Memory [14]) стали основними для багатьох підходів в NLP, зокрема seq2seq [15]. Seq2seq (англ. sequence-to-sequence, послідовність-до-послідовності) є загальною назвою фреймворку, який передбачає задачу трансляції однієї послідовності довільної ненульової довжини в іншу довільну послідовність ненульової довжини. На рисунку 1.2 показано гнучкість цього формулювання - воно дозволяє використовувати один алгоритм для багатьох різноманітних задач пов’язаних з послідовностями.

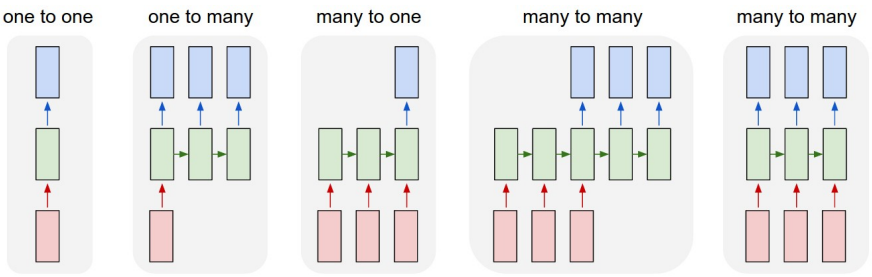


Рисунок 1.2 - Варіації seq2seq фреймворку[16]

Зокрема, в термінах цього підходу можна описати такі сценарії застосування як:

- машинний переклад (послідовність слів однієї мови транслюється в послідовність слів іншої мови)
- класифікація тексту (послідовність слів транслюється в послідновність довжини 1, де єдиний елемент відповідає присвоєному послідовності класу)
- генерація тексту (вхідна послідовність - контекст для генерації, вихідна - продовження тексту)
- розпізнавання усної мови (вхідна послідовність - спектрограма усної мови, вихідна - набір розпізнаних токенів)
- генерація опису зображень (вхідний елемент - репрезентація зображення, вихідний - опис вмісту зображення) тощо.

Конкретні імплементації ввели в NLP поняття енкодера і декодера (аналогічно до автоенкодерів в галузі CV [17, 18]) - окремих компонентів мережі (зазвичай це окремі шари), які відповідають за “стиснення” (кодування) і “розпакування” (розкодування) векторних репрезентацій даних відповідно. Гнучкість цього підходу пов'язана з тим, що на виході з енкодера, яким може бути довільна мережа, отримуємо векторну репрезентацію довільних даних (тексту, аудіо, зображень, таблиць, сирцевого коду, відео тощо) і декодер перетворює (декодує) цю репрезентацію в вихідну послідовність (наприклад текст). Зокрема, це дозволило Google у 2016 році повністю перейти на seq2seq архітектуру для машинного перекладу в Google Translate [19].

Енкодери і декодери не обмежені одношаровими RNN/LSTM, і представлені зокрема багатошаровими [20] і двонапрямленими [21] LSTM а

також згортковими мережами [22] - останні були запозичені з галузі машинного зору. На відміну від зображень, де згортка є операцією над двовимірним клаптиком зображення і оперує кількома каналами одночасно, застосування її для тексту передбачає лише темпоральну згортку по словах, як показано на рисунку 1.3.

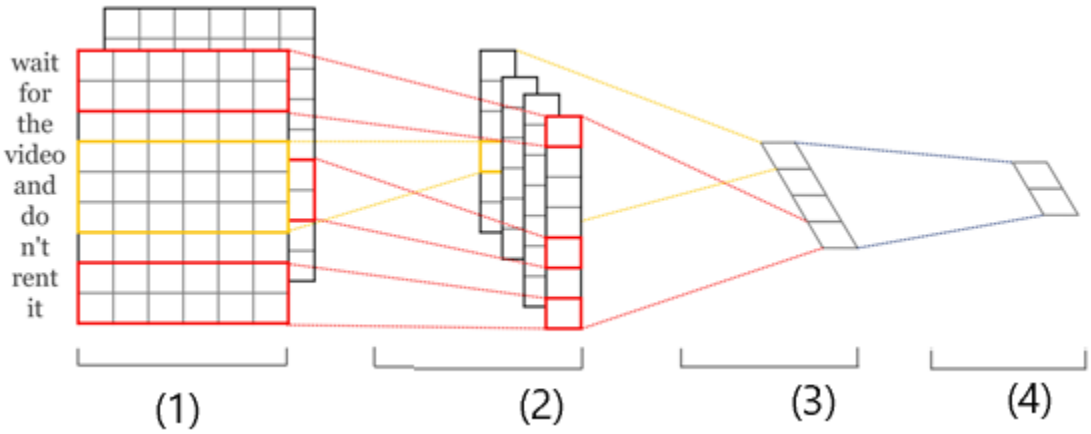


Рисунок 1.3 - Використання згорткових нейронних мереж для обробки тексту. (1) – $n \times k$ вектори речень; (2) – згортковий шар; (3) – max-over-time пулінг; (4) – перцептрон з dropout і softmax виходом [23]

Такий підхід особливий тим, що операція згортки (і її диференціювання) значно краще паралелізується, ніж послідовне за своєю природою зворотне поширення помилки по часу в рекурентних нейронних мережах. Однак згорткові НМ страждають від обмеженого рецептивного поля - нейрони в проміжних шарах отримують інформацію тільки з обмежених регіонів вхідних даних. Це пояснюється тим, що операція згортки бере до уваги лише обмежений контекст з суміжних слів, тому щоб гарантувати наявність нейронів які беруть до уваги весь контекст глибина мережі має зростати лінійно з довжиною вхідної послідовності, що значно ускладнює

обчислення. Є кілька підходів, покликаних боротися з цим, зокрема dilated convolutions [24].

1.1.4 Механізм уваги і Трансформери

Значний приріст в точності моделей вдалося отримати з використанням механізму уваги [25]. До того основною проблемою архітектури енкодер-декодер було те, що весь вміст оригінальної послідовності необхідно перетворити в єдиний вектор контексту, що призводить до неминучої втрати інформації. Механізм уваги обходить цю проблему дозволяючи декодеру використовувати проміжні результати енкодера (зважене середнє з різними наборами коефіцієнтів) на кожному кроці декодування, що дає змогу різним частинам вхідної послідовності по різному впливати на контекст для частин вихідної послідовності, як показано на рисунку 1.4.

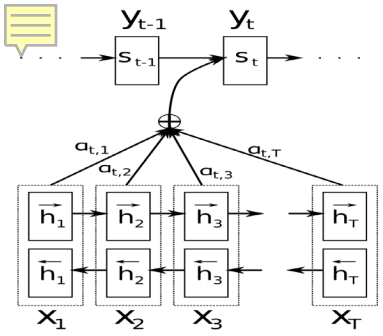


Рисунок 1.4 - Механізм уваги в двошаровій двонапрямленій рекурентній нейронній мережі, де x_i - елемент вхідної послідовності, h_i і h_{i^*} - i-ті приховані стани прямого і зворотного напрямку, α_{ji} - i-та активація на момент часу j, s_j - вихідний стан на момент часу j, y_j - вихідний вектор станом на момент часу j [26]

Цей механізм по особливому корисний, бо коефіцієнти уваги дозволяють отримати інформацію про те, які саме частини вхідних даних модель вважає релевантними при прийнятті рішень. Це підсилюється ще тим, що механізм уваги не обмежений послідовностями - його можна використовувати будь-де де модель по різному зважує важливість вхідних даних.

Революцію в розвитку NLP зробила публікація Attention is All You Need (Увага це все що треба) [27], в якій дослідники описали архітектуру Трансформер, яка не використовує рекурентних зв'язків, швидко тренується і показує SOTA результати на більшості бенчмарків в галузі NLP. Як видно з назви, цей підхід повністю базується на використанні само-уваги (self-attention). Щоб зберегти інформацію про послідовність, Трансформер використовує позиційне кодування для вхідних і вихідних послідовностей.

Розвитком архітектури Трансформерів стали сім'ї моделей OpenAI GPT та BERT.

OpenAI GPT [28] є імплементацією Трансформера, і заклала основи сучасного підходу до тренування NLP моделей, який базується на двох етапах:

1. Unsupervised pre-training (попереднє тренування без вчителя) мовної моделі на величезному корпусі даних (зазвичай гігабайти тексту з Інтернет-ресурсів)

2. Model fine-tuning (дотренування моделі) на кінцевій задачі (наприклад класифікації тексту).

В свою чергу, цей підхід заснований на проторованому в галузі машинного зору підході під назвою Transfer Learning (перенесення знань), в якому основа моделі тренується в класифікації на мільйонах зображень.

Практика показує, що попереднє тренування без вчителя на достатньо великій кількості даних дозволяє моделі вивчити високо репрезентативні абстракції для багатьох інших задач, після чого потрібна лише незначна адаптація моделі (наприклад останніх шарів) щоб отримати state-of-the-art результати на кінцевих задачах.

Нащадками GPT стали GPT-2 [29] та GPT-3 [30] - кожна наступна модель тренована на ще більшій кількості даних, при чому якість їх мовних моделей стала настільки високою, що автори висловили занепокоєння щодо використання їх в цілях масового написання спаму, а GPT-3 може у деяких випадках пройти письмову версію тесту Тюрінга [31].

BERT [32] (і його нащадки BART [33], RoBERTa [34], ALBERT [35] тощо) є також варіантами трансформерів, які використовують підхід Masked LM модель тренується в моделюванні мови за умови, що маскуються довільні токени будь-де в послідовності, а не замасковані слова є контекстом для передбачення (GPT маскує лише завершення послідовності і контекстом для передбачення є всі попередні елементи).

Також BERT отримує на вхід пари речень і вчиться передбачати, чи йде друге речення після першого (Next Sentence Prediction, NSP), що додатково покращує якість кінцевої моделі на деяких задачах. Однак наступні роботи (RoBERTA, XLNET, ALBERT) не використовують NSP.

Спільними рисами усіх Трансформерів є їхня обчислювальна складність, тому значного розвитку отримали спроби дистилювати великі моделі в зменшені варіанти GPT (Distil GPT) та BERT (RoBERTA, ALBERT, DilstilBERT тощо).

Таким чином, можна навести загальну класифікацію нейромережевих архітектур для моделювання природних мов за основним принципом їхньої роботи:

- Багатошарові перцептрони
- Згорткові нейронні мережі
- Рекурентні нейронні мережі
- Трансформери

Іншою характеристикою таких моделей є гранулярність процесу моделювання:

- Word-level (моделюється розподіл імовірності цілих слів)
- Subword-level (моделюється розподіл імовірності частин слів)
- Character-level (моделюється розподіл ймовірностей окремих символів)

1.2 Аналіз нейромережевих моделей української мови

Станом на час написання цієї роботи, у публічному доступі є інформація про кілька мовних моделей для української. Першою є архітектура, запропонована у “Statistical and neural language models for the Ukrainian Language” [36], що використовує тришарову LSTM та приватний тренувальний набір даних. Автори цієї роботи наводять результати тестування цієї моделі для генерації української мови, зокрема повідомляється про значення перплексності 268.

Дещо новішою є модель, опублікована дослідниками з групи lang-uk, що використовує архітектуру BERT[37], проте є розрахованою на доменно-специфічні задачі, такі як відповіді на питання та класифікація релевантності

відповідей, що унеможливлює порівняння цієї моделі з іншими підходами, що генерують текст.

Існує також дві імплементації генераторів тексту на основі вищезгаданого BERT - багатомовна модель mBERT[32] та модель, натренована компанією YouScan на основі RoBERTA та описана в їхньому блозі [38]. Однак жодна з них не повідомляє значення метрик, отриманих при тренування що значно ускладнює порівняння цих підходів з попередніми публікаціями на тему моделювання української мови, тому результати цієї роботи будуть порівнюватися з відповідниками, опублікованими у праці “Statistical and neural language models for the Ukrainian Language”.

1.3 Формулювання науково-технічної задачі

Результатом огляду науково-технічних публікацій є формулювання таких задач цієї роботи:

1. Зібрати дані для моделі, наявні у вільному доступі, які б репрезентативно відображали вжиток української мови її носіями у якомога більшому обсязі.
2. Обрати архітектуру мовної моделі, яка найкраще підходить для вирішення задачі моделювання з урахуванням особливостей української мови.
3. Обрати критерії оцінювання якості отриманої моделі.
4. Спроектувати та розробити вільне ПЗ для тренування та оцінювання якості української мовної моделі.
5. Провести тренування та порівняти отримані результати з попередніми підходами до розв'язання цієї задачі.

1.4 Висновки до першого розділу

У цьому розділі описано проблематику створення мовних моделей, наведено приклади сценаріїв їх використання, здійснено огляд публікацій на релевантну тематику. Встановлено, що ключовими аспектами моделювання є забезпечення моделі репрезентативними даними та висока здатність архітектури моделі формувати високорівневі зв’язки між концептами в текстах. Як наслідок, встановлено тенденцію щодо збільшення розмірів і складності архітектури нейронних мереж, які беруть участь у моделюванні.

Аналіз пов'язаних публікацій показав, що важливим аспектом для спрощення майбутніх досліджень є оприлюднення результатів тренування моделі у вигляді коду, даних та метрик.

2 МЕТОДОЛОГІЯ МОДЕЛЮВАННЯ УКРАЇНСЬКОЇ МОВИ

2.1 Дані

2.1.1 Датасет

Моделювання мови передбачає вивчення розподілу послідовностей токенів у великому корпусі, і чим більше репрезентативних прикладів використовується тим якіснішою буде модель [29, 30].

Більшість публікацій у галузі обробки природної мови працювали з англійською мовою. Це, в свою чергу, стимулювало розвиток датасетів і призвело до появи багатьох відкритих англомовних джерел даних - від вручну анотованих семантичних структур Penn Tree Bank[39] до нерозміченого півекзабайтного дампу Інтернету Common Crawl[40].

Сучасні підходи вкупі зі значним розвитком обчислювальних потужностей за останнє десятиліття показали велику ефективність застосування нерозмичених даних для тренування великих мовних моделей, що дозволяє уникнути потреби у великій кількості ресурсів для ручної анотації великих корпусів.

Щодо української мови, то станом на час написання цієї роботи, у вільному доступі є такі корпуси:

- дамп Вікіпедії [41], який містить майже 2 ГБ нерозміченого тексту з матеріалів проекту українською мовою. Більшість тексту написана в науковому стилі та стосується багатьох тем (природничі науки, історія, мистецтво тощо).

- Корпус законів та правових актів[42], що містить понад 560МБ тексту юридичного характеру, доступний у токенізованій та лематизованій версіях.
- Браунський корпус української мови[42], який виник як результат ініціативи щодо створення україномовного аналогу Brown Corpus[43] обсягом 1 млн слововживань. До складу корпусу входять тексти, які є репрезентативними щодо сучасного вживання української в електронному середовищі.
- Корпус UberText[44], що містить 6 ГБ тексту з 11 українських періодичних онлайн-джерел та Вікіпедії. Щоб уникнути юридичних обмежень, текст розбитий на речення та перемішаний.
- OSCAR Corpus UK[45] - найбільший за обсягом набір текстів українською мовою, зібраний Common Crawl та виділений шляхом автоматичного визначення мови[46]. До нього входять 28ГБ тексту українською - загалом понад 2 млрд слововживань.

У цій роботі обрано OSCAR UK як основний тренувальний набір даних, оскільки цей датасет містить найбільше прикладів вжитку української та є найбільш різноманітним з усіх наявних у вільному доступі.

2.1.2 Розбиття даних

Щоб оцінити якість моделі необхідно сформувати набір даних, на якому буде перевірятися її ефективність. Важливим аспектом підбору такого набору даних є його репрезентативність щодо загального розподілу даних у домені задачі. Також критичною є вимога щодо відсутності перетину між тестовим набором та набором для тренування, оскільки існуватиме ризик того

що модель зможе “запам’ятати”[47] тренувальну вибірку і покаже не репрезентативну ефективність на цих даних.

Існує кілька способів це зробити[48]:

- random train/test split (випадкове розбиття на тренувальну і тестову вибірку). Цей підхід базується на припущенні, що всі наявні дані належать до одного розподілу, тому випадкові вибірки також матимуть ідентичний розподіл.
- train/validation/test split (розбиття на тренувальну, валідаційну та тестові вибірки). Цей підхід схожий на попередній, однак використовує одну додаткову вибірку для валідації або підбору гіперпараметрів. Фінальна оцінка якості моделі проводиться шляхом тренування моделі з оптимальними параметрами, підібраними шляхом тестування на валідаційному наборі, на всіх даних окрім тестових. Таким чином можна більш достовірно стверджувати про те що модель є дійсно кращою а не лише вибирати ті моделі яким “пощастило” показати дещо кращий результат на тестовій вибірці
- Cross-validation (крос-валідація). Такий підхід є узагальненням train/validation/test split і полягає в кількаразовому застосуванні розбиття на тренувальний і валідаційний набори (зокрема шляхом k-fold split[49]). При достатній кількості ітерацій, такий підхід дозволяє провести статистичний аналіз якості гіперпараметрів.

Оскільки великі розміри датасету сильно уповільнюють тренування, ця робота використовує лише розбиття на тренувальний і тестувальний набори. Вибір розміру тренувальної та тестувальної вибірок залежить від конкретної задачі та розміру наявних даних. Оскільки в нас в наявності є мільйони речень, тестувальна вибірка може бути доволі великою, але чим більшою вона буде тим помаліше відбуватиметься оцінка моделі під час тренування. Тому було

обрано вибірку розміром 95% всього датасету для тренування, а решта 5% прикладів було віднесено до тестувальної вибірки. Це становить 7393256 та 389118 прикладів відповідно.

2.1.3 Препроцесинг

Моделювання мови пов'язане з великою розмірністю вхідних даних, тому ефективне подання тексту є важливим чинником при розробці алгоритмів для його аналізу. Існує кілька способів подати текст у чисельному вигляді (провести векторизацію):

- One-hot-encoding. Цей метод полягає у тому, що текст розбивається на слова (токени) і для кожного слова рахується їх кількість у корпусі, після чого обирається фіксований розмір словника N і кожному з N найуживаніших токенів ставиться у відповідність цілочисельний індекс від 0 до $N-1$. Решта слів, які не потрапили до словника, заміняються на спеціальний токен OOV (out of vocabulary), якому присвоюється індекс N. Наступним кроком є присвоєння кожному з отриманих $N+1$ токенів вектора $o \in R^{n+1}$, де $o_i = \mathbb{1}(i = k)$, k - індекс токена, $\mathbb{1}$ – функція-індикатор. Така репрезентація дозволяє уникнути надання індексу слів у словнику ваги, оскільки всі отримані вектори, на відміну від самих індексів, мають одинакову величину. Перетворення послідовності з M слів зводиться до створення матриці $M \times N$, де кожний j-тий рядок відповідає one-hot вектору j-го слова.

- Bag-of-words. Цей метод полягає у перетворенні послідовності з M токенів на $N+1$ розмірний вектор, де N - розмір словника. Основною відмінністю від one-hot encoding є сумування M векторів, що еквівалентно підрахуванню для кожного токена в словнику (і окремо OOV токена) їх кількості у вхідній послідовності. Отримані таким чином вектори дозволяють

одинаково опрацьовувати тексти різної довжини. Існує кілька розширень цієї ідеї, зокрема метод tf-idf [50]. Цей підхід дозволяє надати ваги певним токенам, які можуть мати додаткову інформаційну цінність, та зменшити вагу токенів, які є малоінформативними (частки, прийменники, сполучники). Простота обчислень такої векторної репрезентації дозволяє, зокрема, використовувати tf-idf для ранжування результатів пошукових запитів [51].

- Bag of character n-grams [52]. Цей метод полягає у використанні n-грам символьного рівня для обчислення векторної репрезентації кожного слова як суми векторних репрезентацій кожної символьної n-грами в ньому, є простим в обчисленні і дозволяє використовувати інформацію про структуру всередині слова.

- Byte-Pair Encoding [53]. Цей метод полягає в рекурсивній заміні найчастішої пари послідовних символів на символ заміни доти, доки всі пари у послідовності не будуть унікальними. Простота та ефективність стиснення інформації дозволяє алгоритмам обробки інформації працювати зі стиснутою репрезентацією вхідних даних, яка зберігає інформацію про subword-level структуру слова. Для мов, які мають розширений алфавіт або текстів, які використовують кодування UTF-8 чи UTF-16 можна використовувати BPE байтового рівня (byte-level BPE, BBPE) [54].

- Word2vec. Розвиток ідеї перетворення слів у векторні репрезентації зі збереженням семантики ліг в основу word2vec [55]. Цей алгоритм не використовує глибокого навчання і нелінійності і покладається на певний окіл (контекст з n попередніх та n наступних) слів для того щоб ітеративно покращувати вектор цільового слова. Автори назвали цей підхід Continuous Bag of Words (CBOW). Також було запропоновано зворотній підхід

- “skip-gram”, який полягає в покращенні векторів контексту на основі поточного слова. На рисунку 2.1 показано схему роботи цих підходів.

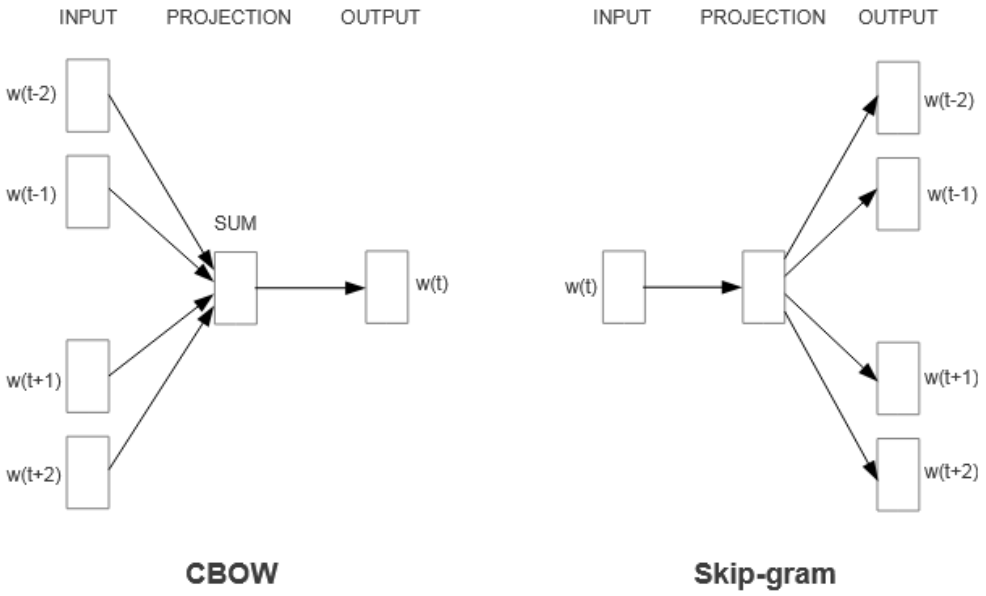


Рисунок 2.1 - Різниця між CBOW та skip-gram підходами в алгоритмі word2vec [56]

Дослідники навели приклади складних семантичних зв'язків, які така відносно нескладна модель успішно вловлює - маючи векторні репрезентації слів, можна показати що різниця між цими репрезентаціями відповідає змістовому відношенню між словами. Зокрема, різниця між репрезентацією слів “король” і “королева” дуже близька до різниці між “чоловік” і “жінка”. Це дозволяє користуватися своєрідною семантичною арифметикою в просторі векторних репрезентацій, наприклад (Athens - Greece) + Oslo = Norway (відношення між векторами слів “Афіни” і “Греція” є відношенням між столицею і країною, додавши вектор “Осло” до цього відношення ми отримуємо вектор, який близький до слова “Норвегія”).

Також існують підходи, покликані зменшити розмірність даних шляхом зменшення граматичної чи семантичної інформації через зведення споріднених слів до спільної форми (канонікалізація). Основними є стемінг та лематизація.

- Стемінг - процес, який дозволяє звести слова до основної форми, позбавленої допоміжних частин слова, таких як суфікс чи закінчення [57]. Під час стемінгу слова "швидко", "швидкий", "швидкі" будуть перетворені до форми "швидк". А слова "бігом", "бігаю", "бігати" взагалі до кореня слова "біг". Є кілька способів імплементації такого алгоритму - пошук за таблицею, відсічення закінчень та суфіксів, стохастичні підходи тощо.
- Лематизація - підхід в обчислювальній лінгвістиці, який дозволяє звести слово до форми, яка зберігає семантику, при цьому не обов'язково зберігаючи граматику [58]. Наприклад, слова “ходить”, “ходитимуть”, “ходити”, “ходила” зводяться до однієї форми - “ходити”. Отримані слова називаються лексемами. Відмінністю лематизації від стемінгу є врахування контексту та семантичних зв'язків між словами, що робить її більш комплексною, однак правильне визначення леми залежить від якості визначення частин мови (part-of-speech tagging, POS tagging).

Використання BPE на неопрацьованому тексті (без канонікалізації) показує себе як ефективний спосіб зменшення розмірності даних без значної втрати граматичної та семантичної інформації[59], тому у цій роботі було обрано саме цей підхід.

2.2 Побудова моделі

2.2.1 Архітектура

Існує багато підходів до моделювання мови з допомогою нейронних мереж, проте більшість останніх досягнень в точності мовних моделей пов'язані з використанням основної архітектури, описаної в Attention Is All You Need, яка отрмала назву Transformer.

Основною її відмінністю від попередніх нейронних мереж є відмова від рекурентних і згорткових шарів та використання шарів multihead attention. Рекурентні мовні моделі зазвичай складаються з двох основних компонентів - енкодера та декодера. Енкодер складається з одного чи більше шарів, які отримують на вхід розріджену (sparse) векторну репрезентацію текстової послідовності та зводить цю репрезентацію до меншої за розмірністю, однак густішою за інформаційним вмістом. Отримані таким чином вектори називаються ембедінгами. Декодер отримує на вхід ембедінги та видає на виході розріджені вектори (зазвичай перетворені через softmax), які дозволяють отримати текстову репрезентацію продовження тексту, який подано на вхід енкодеру.

Звичайні рекурентні нейронні мережі страждають від проблем з парсингом довгих залежностей у тексті, оскільки вектор прихованого стану містить інформацію про весь попередній контекст, що призводить до труднощів з відновленням зв'язку між поточними токенами та тими, що були на самому початку вхідної послідовності. Їх модифікації, такі як gated recurrent unit і long short-term memory network передають інформацію в кілька каналів, що дозволяє полегшити обробку далеких зв'язків, проте все ще впирається у чисельні обмеження при обчисленні градієнтів, оскільки кожна ітерація при

обробці такою мережею є по-суті повторною аплікацією функції перетворення до результату попередньої аплікації, що еквівалентно глибокій нейронній мережі, і градієнти отримані таким чином страждають від ефектів exploding gradient та vanishing gradient.

Exploding gradient (“градієнт, що вибухає”) виникає тоді, коли під час тренування накопичується велике абсолютне значення похибки, яке при обчисленні градієнтів призводить до чисельної нестабільності отриманих значень для кожного шару, що призводить до великої амплітуди самого градієнту, а це призводить погіршення значень параметрів, повторного збільшення похибки і т.д. - тренування розбігається. Існує кілька способів уникнути такого феномену, які зазвичай покликані боротися з зростанням амплітуди градієнту, зокрема - обмеження градієнту. Ця процедура дозволяє встановити верхню і нижню межі для абсолютних значень градієнта для кожного з параметрів нейронної мережі, що унеможливлює різке зростання градієнту за експоненціальним законом, і на практиці показує себе як надійний спосіб для боротьби з exploding gradient в рекуррентних нейронних мережах.

Vanishing gradient (“зникаючий градієнт”) полягає у тому, що виходи перших ітерацій обробки довгої вхідної послідовності при обчисленні значень градієнту для наступного кроку алгоритму градієнтного спуску стають дуже малими або навіть нульовими (через обмеження точності двійкового представлення дійсних числе), що негативно впливає на здатність такої архітектури навчатися.

В звичайних глибоких нейронних мережах для боротьби з цим ефектом використовують так звані skip-звязки, які полягають в простому сумуванні виходів менш глибоких та глибших шарів, що дозволяє градієнту швидеш досягнути до вхідних шарів, що збільшує амплітуду зміни їх параметрів та

пришвидшує навчання. Однак в рекурентних архітектурах це складніше реалізувати, оскільки їхня ефективна глибина може бути динамічною. GRU та LSTM є, по суті, спробами імплементації такого додаткового каналу для поширення градієнту, але з меншою ефективністю.

Альтернативним механізмом, який дозволяє краще поєднати кожну ітерацію з кожною попередньою в рекурентних моделях є attention (“увага”). В архітектурі ендкодер-декодер, енкодер зчитує послідовність векторів $\mathrm{x} = (x_1, \dots, x_{T_x})$ та перетворює у вектор c. При використанні RNN це досягається за допомогою $h_t = f(x_t, h_{t-1})$ та $c = q(h_1, \dots h_{T_x})$, де $h_t \in R^n$ - прихований стан на час t, а c - вектор згенерований з послідовності прихованих станів. f і q - деякі нелінійні функції. В якості f, наприклад, може бути LSTM, а $\mathrm{q}(h_1, \dots h_{T_x}) = \mathrm{h_T}$ [25]. Декодер зазвичай вчиться передбачати наступне слово $\mathrm{y_t}$, на основі вектора контексту c та всіх попередніх передбачених слів $\{y_1, \dots, y_{t'-1}\}$. У випадку з RNN це зводиться до

$$p(y_t|y_1, \dots, y_{t-1}, c) = g(y_{t-1}, s_t, c) \tag{2.1}$$

де g - нелінійна (можливо багатошарова) функція, яка видає імовірність y_t, а s_t - прихований стан. Механізм уваги змінює цю модель, подаючи умовну ймовірність наступним чином:

$$p(y_i, |y_1, \dots, y_{i-1}, \mathrm{x}) = g(y_{i-1}, s_i, c_i), \tag{2.2}$$

де s_i - прихований стан RNN на час i, який обчислюється за формулою

$$s_i = f(s_{i-1}, y_{i-1}, c_i). \tag{2.3}$$

Важливою відмінністю є те, що ймовірність залежить від окремого вектора контексту c_i для кожного цільового слова y_i. Векор контексту залежить від т. зв. анотацій $(h_1, \dots, h_{T_x})$, які генерує енкодер з вхідної послідовності. Кожна анотація h_i містить інформацію стосовно околу i-го токена тексту. Вектор контексту c_i обчислюється як зважена сума анотацій h_i:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j. \tag{2.4}$$

Ваги α_{ij} для кожної анотації рахуються як софтмакс

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} exp(e_{ik})} \tag{2.5}$$

де

$$e_{ij} = a(s_{i-1}, h_j) \tag{2.6}$$

це модель співставлення (alignment model), яка оцінює наскільки вхід у позиції j і вихід у позиції i пов'язані. Функція a - звичайна одношарова нейронна мережа, яка тренується одночасно з рештою параметрів основної мережі. Модель співставлення дозволяє на кожній ітерації “проявляти увагу” (attend to) до певних попередніх слів напряму, що дозволяє значно краще вивчати довготривалі зв'язки між віддаленими словами у вхідній послідовності.

Архітектура трансформерів використовує увагу як основний механізм в енкодері і декодері, та зовсім не використовує послідовних ітерацій. Ключовими будівельними блоками в ній є “багатоголова увага” (multi-head

attention) та “увага через масштабований скалярний добуток” (scaled dot-product attention).

Увага через scaled dot-product оперує трьома входами - Q, K і V, де Q та K - d_k-вимірні вектори “запит” (query) і “ключ” (key) відповідно, а V - d_v-вимірний вектор “значень” (values). Значення уваги визначається за формулою

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.7}$$

яка ефективно масштабується прискорювачами матричних операцій. Графічно ця операція зображена на рисунку 2.2

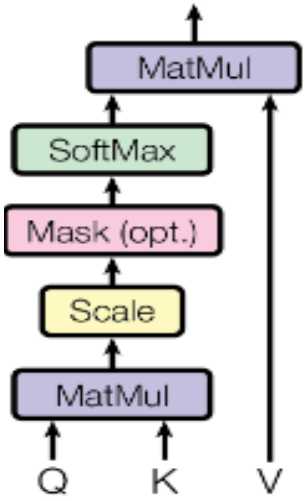


Рисунок 2.2- Обчислення scaled dot-product attention [27]

Багатоголова увага використовує h вивчених проекцій оригінальних Q, K і V, які отримуються шляхом пропуску їх через одношарові перцептрони. Результуючі h значень уваги конкатенуються в один вектор та подаються на вхід одношаровій класичній нейронній мережі, як показано на рисунку 2.3

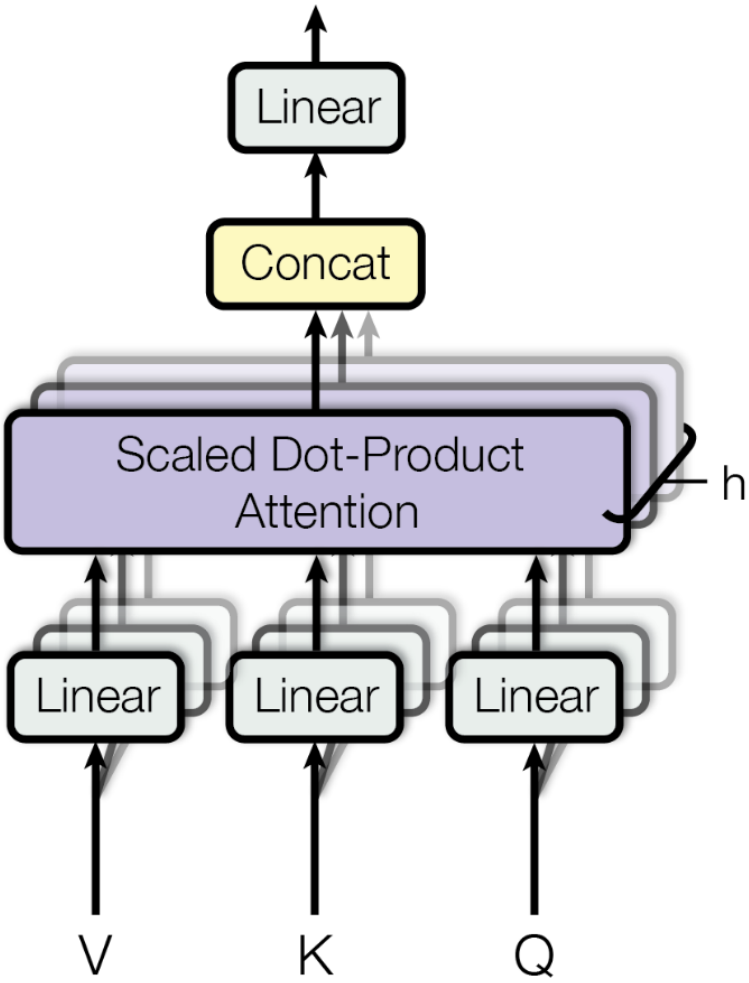


Рисунок 2.3- Схема роботи multi-head attention [27].

Енкодер Трансформера складається з 6 ідентичних шарів, кожен з шарів має два під-шари. Першим є multi-head attention, а другим - проста повнозвязна нейронна мережа з двох шарів та функцією активації ReLU [60] між ними. Між шарами є проміжний зв'язок (residual connection) [61] та шарова нормалізація (layer normalization) [62].

Декодер Трансформера також складається з 6 шарів. На додачу до двох підшарів в кожному шарі енкодера, декодер має додатковий під-шар який застосовує multi-head увагу до виходу енкодера. Аналогічно, цей шар використовує шарову нормалізацію та residual зв'язки. Щоб уникнути

обчислення уваги до майбутніх виходів при обчисленні поточних, застосовується маска та зсув входів на одну позицію вправо, що гарантує залежність поточних передбачень виключно від відомих виходів на момент *i*. Схематично будова Трансформена подана на рисунку 2.4.

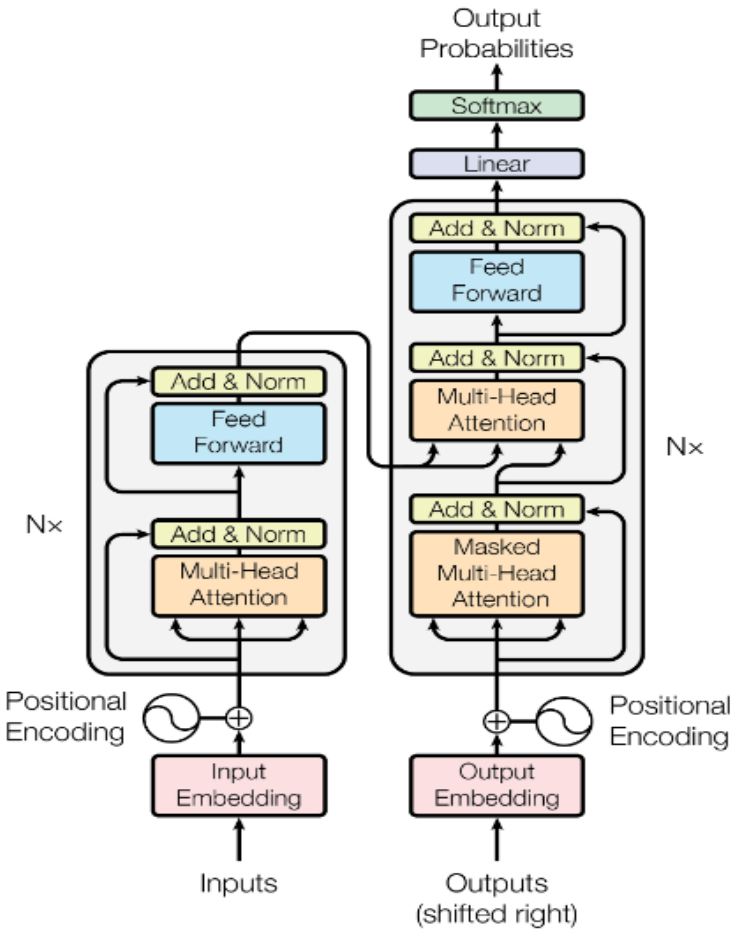


Рисунок 2.4 - Архітектура Трансформера [27]

Хоча Трансформери спершу були застосовані до задачі машинного перекладу, такі моделі як BERT і GPT показали, що якщо замінити вхідні дані з пар “речення” + “перекладене речення” на “замасковане речення” +

“оригінальне речення” то можна ефективно вивчити залежність наступних слів від попереднього контексту - мовну модель.

Для англійської це стало проривом у природності згенерованого тексту та вперше призвело до розгляду етичності використання таких якісних моделей через реальну загрозу масової генерації оманливого контенту, який важко відрізнити від написаного людиною і який може бути згенерований у надзвичайно великих масштабах [63]. Це, зокрема, призвело до обмеження публікації параметрів натренованих великих Трансформерів.

Для вивчення нейронної моделі української мови було обрано підхід, описаний у GPT-2 [64], оскільки він був застосований до схожого обсягу тренувальних даних та показав дуже хороші результати для багатьох мов.

2.2.2 Функція витрат та оптимізація

Тренувальні дані після препроцесингу перетворюються у пари послідовностей з one-hot векторів - замаскований текст на вході і немаскований на виході. Результатом роботи Трансформера є послідовність софтмакс векторів, в яких сума значень в усіх вимірах дорівнює 1, а кожен елемент є невід’ємним, що дозволяє інтерпретувати такий вектор я вектор “впевненості” з відповідними значеннями для кожного токена.

Метою тренування є зробити отримані вектори “впевненості” якомога ближчими до векторів цільових значень, тому функція витрат має бути такою що її значення буде прямувати до нуля коли модель ідеально передбачає тренувальні дані та матиме додатне значення в протилежному випадку. Для цього підходить крос-ентропія:

$$\mathrm{H}(p,q) = -\sum_{x \in X} p \, l\log_2(q) \tag{2.8}$$

де p - цільовий one-hot вектор, а q - вектор передбачень.

Безпосередньо сам алгоритм градієнтного спуску використовує значення функції витрат та обчислення частинних похідних для кожного тренованого параметра в нейронній мережі щоб порахувати значення, на які кожен з параметрів буде змінений. Оскільки обсяги даних роблять неможливим використання всього тренувального набору одночасно, за один крок оптимізації використовується лише випадкова вибірка (батч) з тренувальних даних, на якій відбувається передбачення, обчислення значення функції витрат та зміна параметрів нейронної мережі - такий алгоритм називається стохастичним градієнтним спуском.

Існує кілька варіацій стохастичного градієнтного спуску - з використанням інерції зміни ваг [65], прискореним градієнтом Нестерова [66], адаптивним темпом навчання [67], адаптивним градієнтом [68], адаптивним зниженням ваг [69] тощо. На практиці широко використовуються комбінації цих підходів, одна з них - алгоритм оптимізації Adam, який поєднує ідеї адаптивного темпу навчання RMSprop та адаптивного градієнту Adadelta [70]. Саме цей оптимізатор було обрано для тренування мовної моделі, оскільки він є надійним та присутнім у багатьох пакетах чисельної оптимізації.

2.2.3 Метрики

Для інтерпретації результатів та ходу тренування важливим є можливість отримати чисельну оцінку якості моделі, яку можна порівняти з іншими наявними моделями. Метрики, які для цього використовуються, залежать від конкретної задачі.

Для задач класифікації достатньою мірою, за умови приблизної рівності розмірів класів, часто є точність (accuracy) - кількість правильних

передбачень моделі. У випадках дисбалансу класів (як при класифікації пухлин на доброякісні чи злоякісні) важливими стають специфічність (наскільки надійним є передбачення приналежності до класу) і чутливість (наскільки добре модель виловлює всіх представників класу) або комбіновані метрики, такі як f-міра [71].

Для деяких застосувань в галузі обробки природної мови також підходять ці міри, наприклад для аналізу настроїв у тексті або класифікації новин за жанром. Однак для аналізу генерації мови потрібна міра, яка відображатиме характер близькості загального розподілу отриманого тексту до розподілу натуральних текстів.

В літературі прийнято використовувати перплексність, яка тісно пов'язана з кросентропією (2.8):

$$Perplexity(p,q) = PP(p,q) \coloneqq 2^{H(p,q)} = 2^{-\sum_x plog_2(q)} \qquad (2.9)$$

де p - цільовий one-hot вектор, а q - вектор передбачень.

Для нормалізації результатів, отримане значення ділять на кількість токенів, що дозволяє порівнювати моделі, які оперують послідовностями різної довжини. Існують докази того, що нижче значення перплексії не обов'язково означає природніший текст - особливо в таких мовах, як японська[72], однак наразі немає простого автоматичного способу, який краще корелює і іншими показниками натуральності друкованої мови, таких як швидкість руху очних яблук при читанні, тому ми використовуємо саме цю метрику для оцінки отриманої мовної моделі для української.

2.3 Висновки

На основі аналізу наукових публікацій, для побудови нейронної моделі української мови було обрано:

- архітектуру GPT-2, що заснована на Трансформері
- BPE токенайзер для попередньої обробки текстів
- розбиття на тренувальний і тестувальний набори за принципом 95%/5%
- перплексність як основну метрику для оцінки якості моделі.

3. РОЗРОБКА НЕЙРОМЕРЕЖЕВОЇ МОВНОЇ МОДЕЛІ

3.1 Інструменти для розробки нейронних мереж та аналізу даних

3.1.1 Python

Python - інтерпретована, динамічно типізована, об'єктно-орієнтована мова програмування, яка має легкий синтаксис та можливість виконання коду, скомпільованого на основі сирцевих файлів C та C++. Це дозволяє використовувати високопродуктивні бібліотеки для матричних операцій через високорівневі обертки, такі як NumPy, Sci-kit Learn та SymPy. Існує ціла екосистема пакунків для Python, які дозволяють вирішувати різнопланові задачі аналізу даних - від зчитування вхідних даних з різних джерел (pandas, pyarrow, sqlalchemy) до побудови інтерактивних дашбордів для взаємодії з натренованими моделями машинного начання (streamlit, plotly, dash).

3.1.2 Torch

Розроблений групою Facebook AI Research, Torch (і його обертка PyTorch) є одним з найпопулярніших вільних пакетів для розробки нейронних мереж та імплементації алгоритмів оптимізації [73].

Структурно, PyTorch складається з одноіменного пакету для Python та бекендів на C++, які дозволяють використовувати можливості прискорення обчислень лінійної алгебри з допомогою інструкцій процесора (CPU-бекенди на основі MKL[74], MKL-DNN[75] та OpenMP[76]) та відеоприскорювача (GPU-бекенди на основі Nvidia CUDA[77] та cuDNN [78]).

В залежності від необхідного рівня доступу до процесу обчислень, PyTorch надає різнорівневі API:

- ATen - фундаментальна бібліотека на C++, яка реалізує основний тип даних - Tensor (багатовимірний масив) та операції над ними. Імплементації операцій мають кілька варіацій - в залежності від цільового обчислювального пристрою (CPU чи GPU). Решта компонентів екосистеми PyTorch так чи інакше використовують цю бібліотеку.
- Autograd - бібліотека, яка розширює тип Tensor та надає інструменти для автоматичної диференціації, що, вкупі з побудовою графу обчислень, дозволяє отримувати градієнти.
- C++ PyTorch фронтенд - надає високорівневий C++ інтерфейс для моделювання нейронних мереж та включає ієрархічну систему побудови архітектури моделей, бібліотеку поширених основних будівельних блоків (операції згортки, рекурентні нейронні мережі, нормалізація шарів тощо), API для чисельної оптимізації (такі алгоритми, як SGD, Adam, RMSprop та інші), засоби для роботи з датасетами та пайплайнами даних (включно з паралельною обробкою на багатьох ядрах процесора), інструменти для серіалізації та десеріалізації моделей, автоматичну паралелізацію обчислень на кількох GPU, доступ до JIT компілятора TorchScript тощо. Цей інструментарій є рекомендованою точкою входу для розробки нейронних мереж на C++.
- TorchScript - репрезентація моделей, розроблених на PyTorch, яку може використовувати однойменний компілятор. Заснована на підмножині синтаксису Python, ця мова дозволяє легко конвертувати моделі, розроблені на Python, у компільовану форму, яку може розуміти C++ рантайм, що підвищує ефективність обробки даних та дозволяє запускати їх у середовищі, де інтерпретатор є недоступним.

- Розширення для C++ дозволяють імплементувати власні операції, які недоступні у стандартній бібліотеці перетворень PyTorch.
- TorchServe - веб-сервер для запуску передбачень на натренованих моделях, надає стандарне HTTP REST/gRPC API для виклику обчислень на CPU/GPU.
- PyTorch - Python пакет, який надає високорівневий доступ до можливостей решти стеку Torch, є основним інтерфейсом, з яким працюють розробники та дослідники.

Через інтуїтивність використовуваної моделі “ранніх обчислень” (eager execution) та побудови графу обчислень на льоту (на відміну від попередніх версій таких пакетів, як TensorFlow[79], де використовувалася парадигма статично-компільованого графу) та простоту розширення функціоналу, PyTorch стрімко зростає як найпопулярніший інструмент для проведення академічних досліджень в галузі нейронних мереж [80].

Разом з ростом популярності, зростає також і екосистема інструментів, які дозволяють спростити інші етапи розробки моделей, такі як конвертація між форматами подання (ONNX[81], OpenVINO [82], Triton [83]), відтворюваність та менеджмент експериментів (MLFlow [84], Tensorboard [85]), доменно-специфічні утиліти для оброки фото і відео [86], аудіо [87] та геоданих [88] та ін.

3.1.3 Hugging Face

Компанія Hugging Face [89] займається опенсорсним розвитком екосистеми рішень для обробки природної мови та імплементацією основних нейромережевих архітектур. Ключовим її продуктом є відкрита бібліотека

Transformers [90], яка містить референсні реалізації нейронних моделей (та натреновані параметри) на основі однойменної архітектури.

З розвитком застосування трансформерів, в цій бібліотеці, окрім текстових моделей (таких як GPT, BERT, Transformer XL, T5) з'явилися також алгоритми для обробки візуальної інформації (ViT), аудіо (Wav2Vec2), таблиць (TAPAS) та мультимодальні підходи (CLIP).

Окрім самих моделей та їх тренування, Hugging Face надає також інструменти для пре- і постпроцесингу даних. Зокрема, бібліотека Tokenizers [91] дозволяє проводити токенізацію тексту кількома методами (BPE, word-level, WordPiece, Unigram) через виконання високопродуктивного коду на Rust, який дозволяє за 20 секунд опрацювати гігабайт тексту на серверному процесорі. Отримані таким чином токенізатори є сумісними з моделями Transformers, що спрощує тренування та розгортання таких комплексних алгоритмів.

Простота використання та наявність багатьох сучасних імплементованих архітектур, таких як GPT-2, стали вирішальною причиною вибору саме цих бібліотек для моделювання української мови.

3.1.4 Jupyter та Google Colaboratory

Проект Jupyter [92] ставить за мету створити платформу для інтерактивних обчислень для всіх мов програмування, шляхом розробки вільного програмного забезпечення та відкритих стандартів.

Jupyter підтримує понад 40 мов, проте найчастіше його використовують для написання коду на Python, R, Julia та Scala. Основним його елементом є “записники” (notebooks), які є файловим форматом, що містить код та результати його виконання. Окрім стандартного текстового

виводу та коду, Jupyter notebooks підтримують HTML, Markdown та LaTeX, а також дозволяють створювати інтерактивні віджети для взаємодії з кодом ноутбука.

Взаємодія з записниками відбувається через веб-інтерфейс. Наразі існує два способи запустити його - Jupyter Notebook та Jupyter Lab. Notebook був першою імплементацією взаємодії з кодом в рамках Jupyter, а Lab прийшов йому на зміну, оновивши дизайн та деякий функціонал, зокрема стали доступні вкладки редактора в межах однієї вкладки браузера.

Запуск коду в Jupyter відбувається в т.зв. ядрах (kernels), які містять необхідний рантайм та інтерпретатор, що запускає на сервері обчислення та виводить їхній результат на веб-інтерфейс. За замовчуванням доступне середовище для запуску Python скриптів, для запуску коду на інших мовах необхідно встановити відповідне ядро.

Оскільки Jupyter зручно розгорнути на сервері, зазвичай його використовують як інтегроване середовище розробки (IDE), що може використовувати всі ресурси потужного комп’ютера. Через це Jupyter Notebook набув популярності серед дослідників в галузі високопродуктивних обчислень, аналізу даних та машинного навчання.

Компанії, які надають доступ до хмарних ресурсів, тепер пропонують розробникам готові інсталяції Jupyter Notebook або Jupyter Lab, які під'єднані до потужних процесорів та відеокарт. Одним з таких продуктів є Google Collaboratory (скорочено - Colab)[93].

Colab позиціонується як доступний спосіб запускати експерименти з аналізу даних та машинного навчання у середовищі з виділеними CPU і GPU. Існує кілька тарифних планів, проте навіть безкоштовний надає доступ до потужних відеокарт Nvidia K80, T4, P100 та тензорних обчислювальних

пристроїв TPU. Код записників можна зберігати в хмарному сховищі Google Drive або в GitHub. Доступною також є можливість приєднати вміст диску як папку у віртуальній машині, що запускає самі записники - це дозволяє зберігати артефакти тренування чи аналізу даних для наступного використання.

Оскільки платформа працює за рахунок використання простоюваних ресурсів Google Cloud Platform, існують обеження щодо тривалості обчислень та відсутні будь-які гарантії щодо переривання ядра. Зокрема, безкоштовна версія має верхню межу тривалості запуску коду 24 години, а на практиці рідко вдається тримати записник активним довше ніж 8 годин.

Запропоновані Google Colab ресурси є найдоступнішим способом отримати великі обчислювальні потужності, необхідні для тренування мовних моделей, тому запуск деяких експериментів відбувався саме на цій платформі.

3.1.5 DVC

Окрім, безпосередньо, відслідковування коду експериментів, важливим є збереження інформації про всі етапи тренування нейронних мереж. Ними є:

- Отримання даних
- Перетворення даних
- Розбиття на вибірки тренування, валідації та тестування
- Тренування моделі
- Оцінка якості моделі через обчислення метрик
- Експорт артефактів моделі (гіпер параметри, ваги тощо)

Кожен з цих етапів може залежати від стохастичних параметрів (наприклад розбиття на тренувальну та тестувальну вибірку випадковим

чином), що додає стохастичності до результатів всіх етапів, що йдуть далі. Тому слід з особливою обережністю зберігати проміжні артефакти (перетворені дані, метрики тощо) для покращення колаборації та спрощення відтворення результатів.

Одним з інструментів, які дозволяють це зробити, є Data Version Control (DVC)[94]. Це легкий пакет для Python, який дозволяє описати експеримент у вигляді напрямленого графу обчислень, де самі обчислення можуть бути виконанням довільної програми. Після виконання кожного з етапів у цьому графі, DVC автоматично відслідковує зміни в файловій системі, у якій зберігаються артефакти. Цей інструмент веде власний облік файлів, ставлячи їм у відповідність хеш, за яким кешується вміст файлу. При наступних запусках графу, DVC спочатку рахує хеші файлів, які позначені як вхідні дані для його ланок (чирцеві файли з кодом, параметри, зображення для тренування тощо), і якщо жодне з них не змінилося - DVC витягує кешовану версію результатів цього етапу і продовжує обчислення далі.

Опис графу обчислень відбувається через YAML файли, в яких вказуються самі етапи, команди для їх запуску, залежності та артефакти, які вони продукують. DVC надає простий спосіб запустити весь граф, порівняти метрики з попередніми ітераціями та поділитися результатами роботи шляхом завантаження вмісту кешу на віддалене сховище, таке як Google Cloud Storage чи Amazon S3.

Через свою простоту та легку інтеграцію з системою контролю версій Git, Data Version Control було обрано як основний рушій для запуску експериментів та забезпечення їх відтворюваності.

3.2 Тренування нейронної мережі

3.2.1 Структура проекту

Тренування мовної моделі оформлено у вигляді проекту на Python, який використовує DVC як основний рушій запуску скриптів та зберігання їх результатів.

Кожен з етапів отримання нейронної мовної моделі оформлено у вигляді вузла в графі DVC (див. додаток Г). Вузлами є:

- Отримання даних (download_data)
- Тренування токенайзера (train_tokenizer)
- Препроцесинг датасету (prepare_dataset)
- Тренування моделі та обчислення метрик (train)

Отриманий граф залежностей між етапами DVC подано на рисунку 3.1

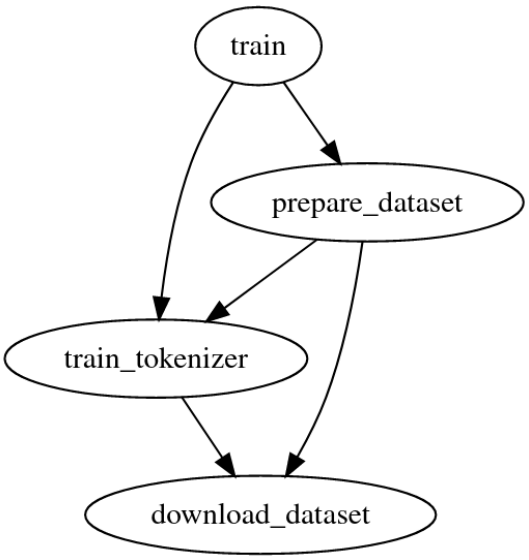


Рисунок 3.1 - Граф залежностей етапів DVC. Напрямок стрілок вказує від залежного етапу до його залежностей.

Самі етапи в DVC оперують файлами, запускають команди які їх опрацьовують та продукують файли-результати. Граф залежностей між файлами подано на рисунку 3.2.

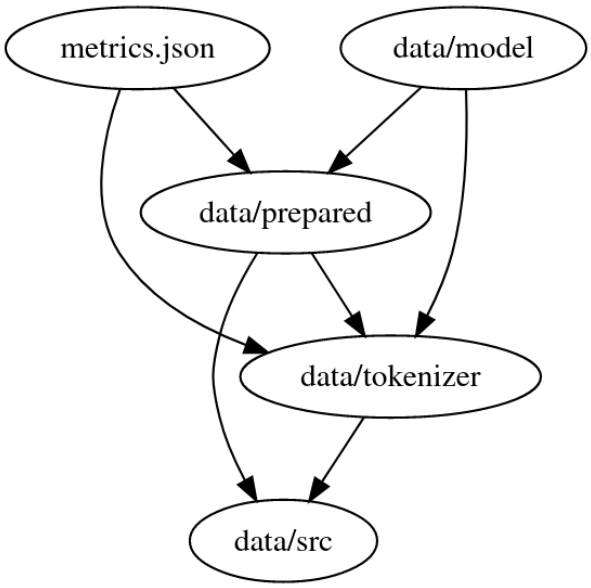


Рисунок 3.2 - Граф залежностей файлів DVC. Напрямок стрілок вказує від залежного файлу до його залежностей.

Файлова структура проекту наведена на рисунку 3.3.

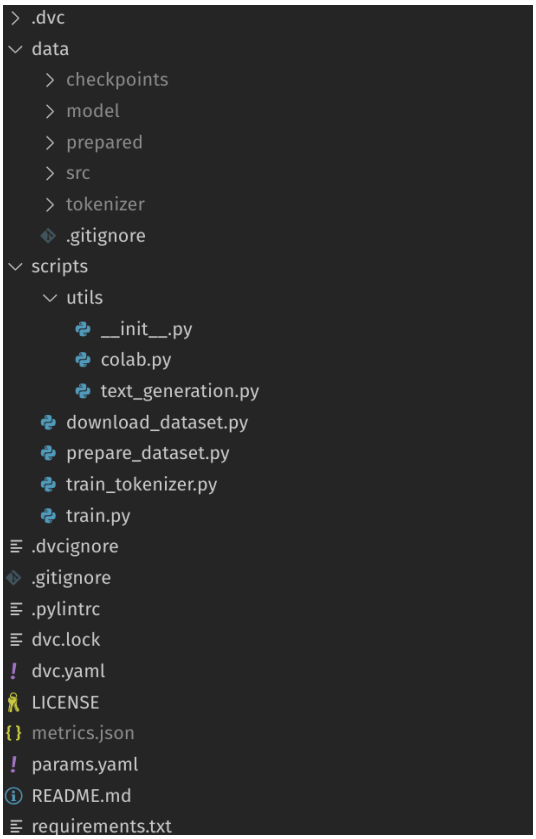


Рисунок 3.3 - Файлова структура проекту

Скрипти, які використовуються в проекті, розміщено в папці scripts/, а дані (оригінальні та перетворені) зберігаються у папці data/. В свою чергу, в scripts/ є підпапка utils/ для додаткових утиліт, таких як генерація тексту. В корені проекту розміщені конфігураційні файли та ліцензія.

3.2.2 Отримання даних

Отримання даних (датасету OSCAR-uk) відбувається у DVC етапі download_dataset і запускається скриптом scripts/download_dataset.py (див. додаток Е). Цей скрипт використовує файл конфігурації parameters.yaml (див. додаток Д) та завантажує необхідний датасет з репозиторію Hugging Face в папку data/src/.

3.2.3 Тренування токенайзера

Наступний етап train_tokenizer використовує раніше отриманий датасет для створення і тренування BPE токенайзера. Це візбувається за рахунок розбиття текстів на батчі розміром 1024 послідовності, розбиття на слова по пробілах, додавання необхідних токенів ([UNK] для OOV-токенів, [SEP] для розділу кількох послідовностей, [PAD] для доповнення довжини послідовності до відповідного розміру моделі, [MASK] для маскування цільових значень при навчанні) та безпосередньо алгоритму стиснення BPE.

Результатом роботи цього етапу є файли тренованого BPE токенайзера, який вивчив частотну характеристику n-грам датасету. Код цього етапу міститься у файлі scripts/train_tokenizer.py (див. додаток Є).

3.2.4 Препроцесинг даних

Отриманий токенайзер використовується для створення перетвореного датасету, який є розбитим на тренувальну та валідаційну вибірки та містить дані у вигляді векторів, отриманих від BPE токенайзера.

На цьому етапі визначається розмір тренувального батчу та відбувається зберігання готових даних на диску, щоб уникнути потреби запуску токенайзера перед кожною ітерацію тренування та пришвидшити процес тренування.

Код цього етапу міститься у скрипті scripts/prepare_dataset.py (див. додаток Ж), а результатом його виконання є приготований датасет у папці data/prepared/.

3.2.5 Тренування моделі та обчислення метрик

Безпосередньо тренування мовної моделі української мови з використанням архітектури GPT-2 відбувається на етапі train, код якого наведено у файлі scripts/train.py (див. додаток З).

Спочатку підвантажується раніше натренований BPE токенайзер. Далі токенайзер загортається в об'єкт DataCollator, який використовує його для формування фінальних батчів тренування у формі тензорів. Наступними підвантажуються попередньо приготовані датасети для моделі.

На основі конфігурації params.yaml створюється сама модель GPT-2, архітектура якої є в бібліотеці Hugging Face. Також конфігурується процес автоматичного збереження проміжних результаттів тренування у папці data/checkpoints/, який дозволяє автоматично продовжувати переване раніше тренування.

Саме тренування відбувається шляхом створення об'єкту Trainer, який поєднує в собі архітектуру моделі, гіпер параметри (кількість епох, темп навчання, параметри оптимізатора Adam тощо), датасет, коллатор та конфігурацію збереження model checkpoints. Виклик методу .train() цього об'єкта запускає тренування моделі, яке використовує наявні ресурси CPU/GPU, автоматично робить збереження прогресу тренування та показує відсоток закінчення тренування. Після успушного завершення, модель та її параметри зберігаються у папці data/models/.

Заключним етапом є обчислення метрик отриманої моделі. Оскільки при тренування функція витрат є крос ентропією, то перпрексію можна обчислити шляхом піднесення 2 до степеню, який рівний значенню функції витрат. Отримана метрика записується у файлі metrics.json.

3.3 Результати тренування

В ході тренування мовної моделі з конфігурацією GPT-2 з наступними гіпер параметрами:

- розмір батчу (batch size) - 10
- кількість тренувальних епох (train epochs) - 1
- темп тренування (learning rate) - 2e-5
- adam_beta1=0.9,
- adam_beta2=0.999,
- adam_epsilon=1e-08,
- max_grad_norm=1.0,
- warmup_ratio=0.0,
- warmup_steps=0,
- weight_decay=0.01,
- vocab_size=50257,
- n_positions=1024,
- n_embd=768,
- n_layer=12,
- n_head=12,
- n_inner=None,
- activation_function="gelu_new",
- resid_pdrop=0.1,
- embd_pdrop=0.1,
- attn_pdrop=0.1,
- layer_norm_epsilon=1e-5,
- initializer_range=0.02,

та загальною кількістю тренованих параметрів 124,4 млн на всьому датасеті OSCAR-uk (понад 2 млрд слововживань) було отримано значення функції витрат 5.45 та perplexity 44. Основне тренування відбувалося на відеокарті Nvidia GTX 1060 6GB та зайняло понад 3 тижні безперервних обчислень. Прототипізація велася на платформі Google Colab, проте неможливість проведення тривалих обчислень в поєднанні з величезним розміром тренувального датасету (понад 26 ГБ) стали основними обмежуючими факторами при виконанні цієї роботи.

Для порівняння, в ранішій публікації, у якій проводилося тренування мовної моделі української [36], найкращі результати було отримано шляхом використання тришарової word-level LSTM (245 млн параметрів) та частково пропрієтарного набору даних (242 млн слововживань) - в роботі повідомляється про значення перплексності 268,5 на корпусі Brown-UK Corpus. Тренування проводилося на відеокарті Nvidia RTX 2080 ti 11GB та займало близько тижня, що приблизно еквівалента кількість обчислень у нашій роботі через трикратну різницю в обчислювальній потужності відеокарт.

Для кращої інтерпретації результатів, є скрипт scripts/utils/text_generation.py (наведено у додатку І), який зчитує контекст для генерації як аргумент командного рядка та виводить 5 варіантів продовження тексту. Ось один з прикладів такої генерації, який отримав в якості контексту “Вчора”:

“Вчора в Івано-Франківській області сталася аварія на шахті Київ-Київ. Внаслідок ДТП загинула дитина. Про це повідомив депутат Київради Володимир Павловський, передає Укрінформ. Причиною аварії в будинку за участю голови Київ-Окука надійшла аварія за участю двох гірників і дві

людини госпіталізовані з пораненням ноги. За останніми даними, в результаті інциденту загинули 44 людини і шестеро одиниць техніки, з них четверо госпіталізовані. На одному із лікарень померла одна дитина. Розпочато кримінальне провадження за статтею 115, 3 ч. Ч.1".

Окрім очевидної нісенітності змісту, легко помітити, що текст написаний у стилі новинної статті у періодиці, має багату лексичну різноманітність, використовує часто вживані журналістами граматичні структури тощо.

Згенеровані тексти мають ознаки розуміння моделлю синтаксису та граматики української мови, але семантична складова все ще є достатньо обмеженою через проблеми зі зберіганням логічної послідовності подій та дійових осіб.

3.4 Висновки

Отримана нейронна мовна модель на основі архітектури GPT-2 показала кращі результати, ніж попередні підходи на основі LSTM, використовуючи при цьому аналогічну кількість обчислювальних потужностей та дозволяє генерувати різноманітні тексти на основі навіть обмеженого контексту. Також, ми показали, що використання subword-level токенізації дозволило отримати велику граматичну і синтаксичну різноманітність, що підтверджує ефективність використання таких підходів як BPE для мов таких мов зі значною інфлекцією як українська.

Однак, отримані речення часто мають ознаки нелогічності та неоднорідності, присутні часті зміни тематики тексту, повтори та дрібні граматичні помилки. Ми вважаємо, що такі результати, в основному, пов'язані

з недотренованістю моделі, і що продовження тренування на кілька епох дозволить отримувати кращі тексти. Таке тренування, однак, вимагає великої кількості обчислювальних ресурсів, і є недоступним для нас з існуючими в наявності апаратними та часовими обмеженнями.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Дослідження можливості створення інструкції з охорони праці для фахівця з інформаційних технологій з використанням нейронних мереж

На кожному підприємстві, незалежно від форми власності та виду економічної діяльності, якщо використовується наймана праця, тобто коли є хоча б один працівник, має бути розроблено інструкції з охорони праці за професіями та видами робіт. Підзаконним актом, який регламентує розробку, введення в дію та перегляд інструкцій з охорони праці є Положення про розробку інструкцій з охорони праці (НПАОП 0.00-4.15-98)[95].

Оскільки тема кваліфікаційної роботи напряму відноситься до генерації тексту та можливості взаємодії з системами штучного інтелекту шляхом використання природної мови, то постає важливе питання визначення меж безпечного застосування цих підходів. Зокрема, слід розглянути можливості застосування нейромережевих компонентів для створення нормативних документів, таких як інструкції з охорони праці для працівників в галузі обробки інформації.

Така інструкція повинна містити, зокрема, загальні положення про робочий процес, обов'язки працівника, вимоги безпеки перед, під час та після виконання роботи та регламентувати порядок дій при настанні аварійної ситуації (ураження струмом, кровотеча, опіки тощо) [96].

Перспективне застосування систем текстогенерації для такого вжитку пов'язане з наступними аспектами:

- Стилістика, граматика та вузькоспеціалізована лексика. Відповідні системи повинні дотримуватися основних вимог до складання таких інструкцій, які не допускають розпливчастих формулювань, використання сленгу, діалектизмів, художніх елементів тощо.
- Форматування та стандартизація. Текст інструкцій, згенерований з використанням систем штучного інтелекту має відповідати чинним вимогам до таких документів (структура, типографія тощо).
- Використання графічних та інших матеріалів. В ілюстративних цілях інструкційні матеріали часто містять додатки у вигляді зображень, таблиць. Оскільки відокремлена текстогенерація обмежена текстовою модальністю, може виникнути необхідність у використанні систем створення зображень на основі тексту.
- Семантична наповненість та логічна однорідність. Результатом роботи алгоритмів мають бути змістовні інструкції, що не суперечать одна одній та здоровому глузду.

Нижче детально розглянуто кожен з аспектів та можливості їх втілення шляхом використання нейронних мереж.

Лексикон, який може вивчити нейронна мовна модель, залежить від різноманітності лексики у тренувальних даних та використаних під час тренування механізмів токенізації вхідних послідовностей, а саме гранулярності цього процесу.

Зокрема, використання підходів з фіксованим, а відтак обмеженим, словником цілих слів (word-level токенізація) може призвести до відсутності певних слів в арсеналі нейронної мережі, що унеможливлює їх використання при генерації. Ця проблема додатково ускладнюється в мовах з великою кількістю інфлекцій (наприклад більшість іменників в українській мові мають

кілька відмінків з різним написанням) та майже унеможливлюється у випадках зі складним словотвором (зокрема в німецькій мові). На практиці це означає, що модель у деяких випадках не зможе згенерувати правильні словоформи, що вимагатиме додаткового втручання з боку оператора такої системи.

Вирішити проблеми з обмеженістю лексикону моделі можна шляхом використання subword-level токенізації, яка вивчає не розподіл цілих слів а складових частин слів. Інформатико-теоретичні підходи, такі як byte-pair encoding (BPE), дозволяють в автоматичному режимі виокремити основні компоненти словотвору шляхом розбиття слів на підпослідовності за їхньою частотою. З утворених токенів можна генерувати навіть словоформи, які не зустрічаються в тренувальному наборі даних, що вирішує проблему лексичної і граматичної повноти.

Стилістика інструкційних матеріалів відрізняється від текстів, які зазвичай використовують для початкового тренування нейронних мереж, оскільки, зазвичай, це великі корпуси з мережі Інтернет. Однак додаткове тренування на прикладах інструкцій дозволяє звузити стилістичне різноманіття результатів генерації [97].

З розвитком великих мовних моделей стало можливим використання prompt programming (англ. “програмування запитів”) напряму, без додаткового тренування на текстах у певному стилі. Такий підхід базується на використанні знань, збережених під час тренування, для розв’язання вузькоспеціалізованих задач через формування запиту для генерації тексту [98]. Це дозволяє, скажімо, подати на вхід моделі запит “Продовжити фразу в стилі інструкційного матеріалу: робоче місце для роботи сидячи має” і отримати згенерований текст, який, окрім простого продовження фрази, додатково буде відповідати вказаному стилю.

Інструкційний матеріал щодо техніки безпеки належить до категорії посадових інструкцій та має відповідати ДСТУ 4163-2020 “Уніфікована система організаційно-розпорядчої документації. Вимоги до оформлення документів”. В цьому стандарті описано оформлення такої документації, що включає в собі, окрім тіла документа, такі елементи як місце для підпису чи печатки, порядкові номери документів, імена посадових осіб та авторів документа, при чому кожен з цих елементів повинен мати своє відведене місце.

У випадку складання такої інструкції людиною, зазвичай використовуються пакети для редагування офісних документів, такі як Microsoft Office або LaTeX - у їх функціонал входить можливість зміни положення тексту, шрифтів тощо. Для генерації інструкції штучним інтелектом необхідно або використовувати готовий, наперед зроблений людиною шаблон, в якому замінити лише основну частину, або генерувати окрім тексту щей додаткову розмітку для документа. Прикладом такого застосування мовних моделей є згенеровані LaTeX документи, отримані шляхом тренування моделі на сирцевих файлах з бази наукових праць arXiv, які мають звичне для публікацій оформлення [11].

Графічні матеріали табличного типу можна генерувати таким же способом, як і будь-яку іншу розмітку, оскільки їхня основна модальність є мовленнєвою за природою. Створення ілюстрацій, однак, є значно складнішою задачею, оскільки окрім відповідності за змістом, зображення мають відповідати ілюстративному стилю та бути в хорошій якості. Генерація зображень також сильно ускладнюється тим, що простір всіх можливих зображень на багато порядків більший, ніж простір зображень, які може

зрозуміти людина, а особливості зорової кори людського мозку додатково ставлять вимоги до естетичності та приємності.

Існує кілька підходів до генерації зображень на основі тексту (text-to-image generation). Більшість з них використовує генеративно-змагальні нейронні мережі, які складаються з двох компонентів - генератора і критика. Генератор вчиться створювати зображення на основі тексту, а критик - відрізняти генеровані зображення від справжніх. При достатньому різноманітні тренувальних даних, такі архітектури можуть створювати навіть фотореалістичні зображення [99,100].

З ростом доступних обчислювальних потужностей зростає і генералізаційна здатність нейромережевих підходів, що дозволило дослідникам з OpenAI створити модель на основі трансформерів, яка тренувалася на зображеннях та анотаціях одночасно. Отрмані таким чином моделі DALL-E [101] і DALL-E 2 [102] демонструють фотореалізм та дуже великий спектр можливостей генерації фото хорошої якості за будь-яким описом.

При створенні інструкції з охорони праці, критично важливими є зміст самого документу та його відповідність до реальних умов праці людини. При тренуванні великих мовних моделей, отримані алгоритми мають ознаки запам’ятовування багатьох фактів та взаємодій у фізичному світі (наприклад що нагрівання деревини може призвести до пожежі), однак наразі немає достатньо підстав вважати, що такі системи мають зачатки причинно-наслідкового мислення, здатного до активного синтезу ідей. Наразі питання можливості виникнення такого мислення в нейронних мережах є предметом багатьох теоретичних і практичних досліджень і залишається відкритим [103,104]. Зокрема, ведуться дослідження можливості накладання інструкцій

на попередньо треновані моделі для покращення надійності таких застосувань мовних моделей [105, 106].

Таким чином, цілком можлива генерація текстів у стилі службових інструкцій з охорони праці для фахівця з інформаційних технологій, а також оформлення таких інструкцій у відповідності до державних стандартів. Однак, наразі немає підходів, які гарантують логічну і семантичну когерентність згенерованих текстів, тому застосування нейронних мовних моделей для генерації такої документації повинне обов'язково супроводжуватися контролем та редактурою з боку відповідальних операторів-людей, які повинні впевнитись у безпечності та повноті отриманих документів.

4.2 Ергономічні вимоги до робочого місця користувача персональним комп'ютером (ПК)

Робота користувача ПК пов'язана з довготривалим перебуванням в сидячому положенні, тому важливо зробити робоче місце якомога комфортнішим. Його ергономічність обумовлена поєднанням багатьох чинників, таких як висота робочої поверхні, кут нахилу лінії зору, зони доступності основних засобів керування ПК тощо, тому задля уникнення хронічних травм пов'язаних з неправильно організованою сидячою роботою важливо дотримуватися вимог до робочого середовища.

Стандарт ДСТУ 8604:2015 встановлює загальні ергономічні вимоги до робочих місць для виконання робіт у положенні сидячи. Його застосовують під час проектування нового та модернізації устаткування й виробничих процесів. Загальні вимоги цього стандарту застосовують під час розробляння

стандартів і нормативно-технічних документів, що встановлюють вимоги ергономіки до конкретних робочих місць.

Згідно зі стандартом, Конструкція робочого місця та взаємне розташування всіх його елементів (сидіння, органів керування, засобів відображення інформації тощо) повинні відповідати антропометричним, фізіологічним і психологічним вимогам, а також характеру виконуваної роботи.

Конструкцією робочого місця повинно бути забезпечено виконання трудових операцій у межах зони досяжності моторного поля. Зони досяжності моторного поля у вертикальній і горизонтальній площинах (як приклад) наведені на рисунках 4.1 і 4.2.

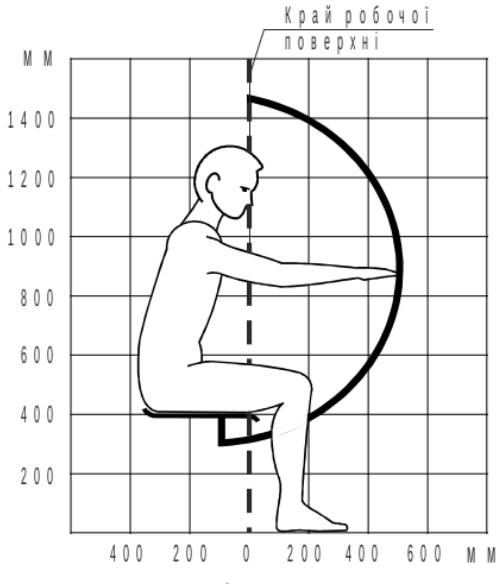


Рисунок 4.1 - Зона досяжності моторного поля у вертикальній площині (для 50-го перцентиля)

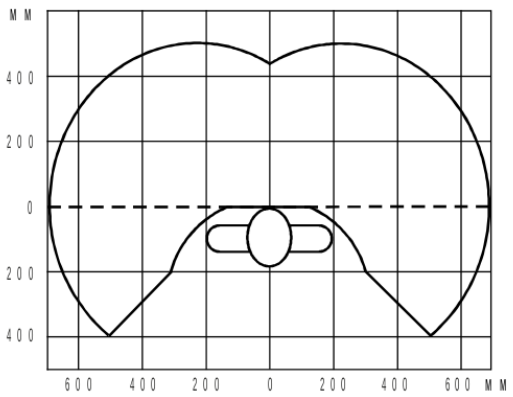


Рисунок 4.2 - Зона досяжності моторного поля у горизонтальній площині при висоті робочої поверхні над підлогою 725 мм (для 50-го перцентиля)

Стандарт вводить поняття частоти виконання операцій та виділяє такі 3 категорії: дуже часто - дві і більше операції за одну хвилину, часто - менше ніж дві операції за хвилину але більше ніж дві операції за одну годину, рідко - не більше двох операцій за одну годину. Відповідно до цих частот відбувається поділ зони досяжності моторного поля на 3 зони: 1 - оптимальна зона моторного поля, 2 - зона легкої досяжності моторного поля, 3 - зона достяжності моторного поля. На рисунку 4.3 подано розбиття на ці зони.

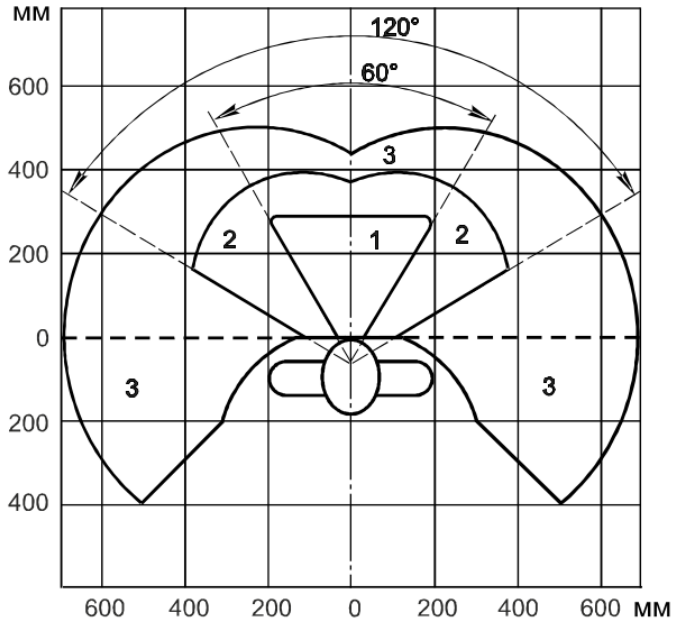


Рисунок 4.3 - Зони для виконання ручних операцій і розміщення органів керування

Наприклад, клавіатуру та мишку слід розміщувати в оптимальній зоні, додаткову часто вживану периферію - і зоні легкої досяжності, а рідковживані пристрої (принтер, сканер тощо) можна розміщувати будь-де в межах зони досяжності моторного поля. Важливо також не допускати ситуації перехрещення рук. Стандартом також регулюється максимальна висота для розміщення пристроїв керування і становить 1100мм.

Важливим аспектами комфортності сидячої роботи за ПК є висота робочої поверхні, сидіння та простору дня ніг. Відповідно до стандарту, такий вид роботи можна охарактеризувати як “друкування, набирання тексту” і виходячи з цього можна підібрати оптимальну висоту робочої поверхні у випадку, коли її не можна регулювати - для жінок це 630мм, для чоловіків - 680мм, а для місць на яких працюють і чоловіки і жінки ця висота становить 655мм.

Стандарт рекомендує використовувати якомога тоншу стільницю - до 30мм, мінімальна товщина має бути зумовлена характеристиками міцності використовуваного матеріалу.

У випадках коли висота робочої поверхні не може бути достатньо низькою слід використовувати підставку для ніг. Повинна бути можливість регулювати висоту такої підставки, її ширина і довжина мають бути не менше 300мм і 400мм відпповідно, а її поверхня має бути рифленою. Доцільно також передбачити 10мм ботик по передньому краю. На рисунку 4.4 подано характеристики розміру для простору дня ніг.

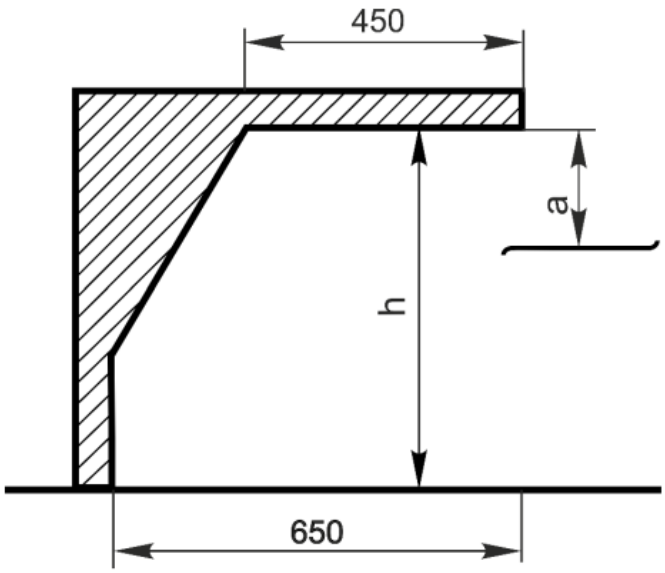


Рисунок 4.4 - Простір для ніг. a – відстань від поверхні сидіння до нижнього краю робочої поверхні (не менше ніж 150 мм); h – висота простору для ніг (не менше ніж 600 мм); (ширина простору для ніг не менше ніж 500 мм)

Окремої уваги заслуговує розміщення засобів відображення інформації. Основним аспектом зручності їх використання є їх близькість до

нормальної лінії погляду (ця лінія на 15° нижче горизонтальної лінії погляду). Аналогічно до зон моторної досяжності, виділяють зони зорового спостереження, як показано на риснку 4.5.

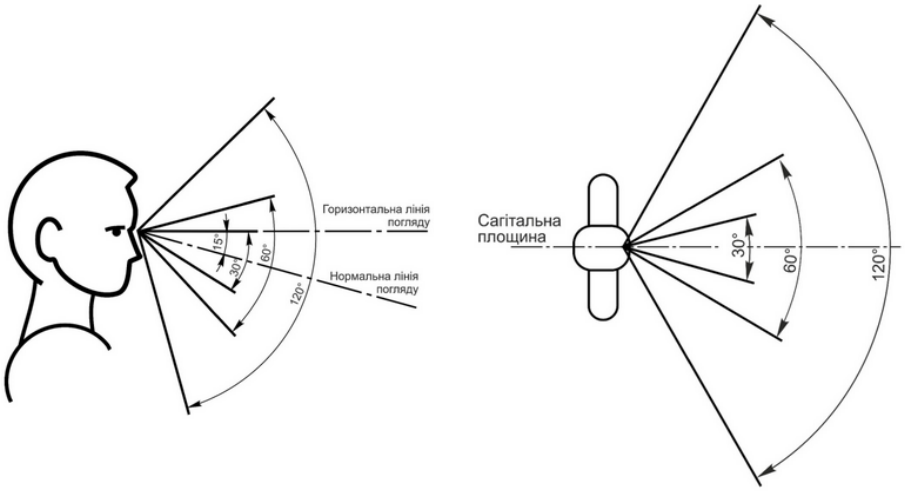


Рисунок 4.5 - Зони зорового спостереження

Дуже часто використовувані засоби відображення інформації, які вимагають точного і швидкого зчитування показань, повинно розміщувати у вертикальній площині під кутом ± 15° від нормальної лінії погляду та в горизонтальній площині під кутом ± 15° від сагітальної площини. Часто використовувані засоби відображення інформації, що вимагають менш точного і швидкого зчитування показань, допускається розташовувати у вертикальній площині під кутом ± 30° від нормальної лінії погляду й у горизонтальній площині під кутом ± 30° від сагітальної площини. Відповідно, основний дисплей повинен бути розташованим в цих межах, а в разі використання додаткових дисплеїв їх слід розміщувати збоку та старатися обмежити їх використання.

ВИСНОВКИ

У цій роботі було проведено глибокий аналіз публікацій на тему моделювання природної мови, зібрано інформацію про алгоритми та нейромережеві архітектури, які дозволяють це зробити. Розглянуто вимоги до процесу тренування нейронних мереж - від характеру даних до процесу оптимізації обраної моделі, а також інструменти, які дозволяють ці вимоги задовольнити.

Результатом роботи є нейронна модель української мови, що використовує архітектуру GPT-2 та BPE енкодер і дозволяє генерувати текст на основі запропонованого контексту. Для цього було розроблено програмний код, написаний на мові програмування Python з використанням інструментів PyTorch, Hugging Face Transformers, Tokenizers та DVC, що дозволяє проводити експерименти з нейронними мережами в галузі моделювання природних мов, а також генерувати тексти на їх основі.

Отримана модель показує кращі результати, ніж схожі порівнювані публікації, які використовували рекурентні нейронні мережі та аналогічні обчислювальні потужності при тренуванні, що доводить перспективність використання архітектури Трансформер для задач обробки і генерації природної мови. Тим не менше, слід провести подальші експерименти з використанням більшої кількості ресурсів, щоб визначити практичні межі їх застосування, а також проаналізувати їх можливості в інших задачах, таких як класифікація тексту, аналіз настроїв, системи питання-відповідь тощо.

Перспективним напрямком для майбутніх досліджень є також використання розподілених гетерогенних обчислювальних ресурсів для тренування великих нейронних мереж, бо ми зіткнулися з неможливістю

застосування потужностей таких платформ, як Google Colab, що безкоштовно надають в тимчасову оренду відеокарти серверного рівня - основною причиною цьому є складність організації процесу при відсутності гарантій щодо неперервності обчислень. Такі платформи можуть значно прискорити експерименти в галузі машинного навчання.

В розділі “Охорона праці та безпека в надзвичайних ситуаціях” проведено аналіз можливості та доцільності застосування нейронних мереж для створення службових інструкцій з охорони праці для робітників в галузі ІТ, а також описано вимоги до ергономіки їхнього робочого місця згідно до державних стандартів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. R. Kuhn and R. De Mori, “Cache-based natural language model for speech recognition,” Pattern Analysis and Machine Intelligence, IEEE Transactions on, vol. 12, pp. 570–583, Jul. 1990, doi: 10.1109/34.56193.

2. L. Rabiner and B. Juang, “An introduction to hidden Markov models,” IEEE ASSP Magazine, vol. 3, no. 1, pp. 4–16, 1986, doi: 10.1109/MASSP.1986.1165342.

3. Terrence J. Sejnowski, “The unreasonable effectiveness of deep learning in artificial intelligence,” Proceedings of the National Academy of Sciences, vol. 117, no. 48, pp. 30033–30038, 2020, doi: 10.1073/pnas.1907373117.

4. A. Graves and J. Schmidhuber, “Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks.,” in NIPS, 2008, pp. 545–552. [Електронний ресурс]. Режим доступу: http://dblp.uni-trier.de/db/conf/nips/nips2008.html#GravesS08

5. D. C. Ciresan, U. Meier, J. Masci, and J. Schmidhuber, “Multi-column deep neural network for traffic sign classification.,” Neural Networks, vol. 32, pp. 333–338, 2012, [Електронний ресурс]. Режим доступу: http://dblp.uni-trier.de/db/journals/nn/nn32.html#CiresanMMS12

6. D. Cireşan, U. Meier, and J. Schmidhuber, “Multi-column Deep Neural Networks for Image Classification,” 2012, doi: 10.48550/ARXIV.1202.2745.

7. L. Özgür, T. Güngör, and F. Gürgen, “Spam Mail Detection Using Artificial Neural Network and Bayesian Filter,” in Intelligent Data Engineering and Automated Learning – IDEAL 2004, 2004, pp. 505–510.

8. L. Zhang, S. Wang, and B. Liu, “Deep Learning for sentiment analysis : a survey,” arXiv:1801.07883 [cs, stat], Jan. 2018, Accessed: Feb. 02, 2018. [Електронний ресурс]. Режим доступу: http://arxiv.org/abs/1801.07883

9. Y. Bengio, R. Ducharme, and P. Vincent, “A Neural Probabilistic Language Model.,” in NIPS, 2000, pp. 932–938. [Електронний ресурс]. Режим доступу: http://dblp.uni-trier.de/db/conf/nips /nips2000.html#BengioDV00

10. R. Collobert and J. Weston, “A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning,” in Proceedings of the 25th International Conference on Machine Learning, 2008, pp. 160–167. doi: 10.1145/1390156.1390177.

11. A. Karpathy, “The Unreasonable Effectiveness of Recurrent Neural Networks,” May 2015. http://karpathy.github.io/2015/05/21/rnn-effectiveness/

12. L. Tai and M. Liu, Deep-learning in Mobile Robotics - from Perception to Control Systems: A Survey on Why and Why not. 2016.

13. J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. arXiv, 2014. doi: 10.48550/ARXIV.1412.3555.

14. S. Hochreiter and J. Schmidhuber, “Long Short-term Memory,” Neural computation, vol. 9, pp. 1735–80, Dec. 1997, doi: 10.1162/neco.1997.9.8.1735.

15. I. Sutskever, O. Vinyals, and Q. V. Le, Sequence to Sequence Learning with Neural Networks. arXiv, 2014. doi: 10.48550/ARXIV.1409.3215.

16. A. Gulli and A. Kapoor, TensorFlow 1.X deep learning cookbook. Birmingham, England: Packt Publishing, 2017.

17. G. E. Hinton and R. S. Zemel, “Autoencoders, Minimum Description Length and Helmholtz Free Energy,” in Advances in Neural Information Processing Systems 6, 1994, pp. 3–10.

18. D. P. Kingma and M. Welling, Auto-Encoding Variational Bayes. arXiv, 2013. doi: 10.48550/ARXIV.1312.6114.

19. A neural network for machine translation, at production scale. 2016. [Електронний ресурс]. Режим доступу: https://ai.googleblog .com/2016/09/a-neural-network-for-machine.html

20. Y. Wu et al., Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. arXiv, 2016. doi: 10.48550/ARXIV.1609.08144.

21. D. Q. Nguyen, D. Q. Nguyen, C. X. Chu, S. Thater, and M. Pinkal, Sequence to Sequence Learning for Event Prediction. arXiv, 2017. doi: 10.48550/ARXIV.1709.06033.

22. J. Wang, L.-C. Yu, K. R. Lai, and X. Zhang, "Dimensional Sentiment Analysis Using a Regional CNN-LSTM Model," in Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), Aug. 2016, pp. 225–230. doi: 10.18653/v1/P16-2037.

23. A. Conneau, H. Schwenk, L. Barrault, and Y. Lecun, Very Deep Convolutional Networks for Text Classification. arXiv, 2016. doi: 10.48550/ARXIV.1606.01781.

24. N. Kalchbrenner, L. Espeholt, K. Simonyan, A. van den Oord, A. Graves, and K. Kavukcuoglu, Neural Machine Translation in Linear Time. arXiv, 2016. doi: 10.48550/ARXIV.1610.10099.

25. D. Bahdanau, K. Cho, and Y. Bengio, Neural Machine Translation by Jointly Learning to Align and Translate. arXiv, 2014. doi: 10.48550/ARXIV.1409.0473.

26. J. Dey, A past, present, and future of attention. Medium, 2021. [Електронний ресурс]. Режим доступу: https://joshdey .medium.com/a-past-present-and-future-of-attention-f6e269574a5

27. A. Vaswani et al., Attention Is All You Need. arXiv, 2017. doi: 10.48550/ARXIV.1706.03762.

28. A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, “Improving language understanding by generative pre-training.” 2018.

29. A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, “Language Models are Unsupervised Multitask Learners,” 2018, [Електронний ресурс]. Режим доступу: https://d4mucfpksywv .cloudfront.net/better-language-models/language-models.pdf

30. T. B. Brown et al., Language Models are Few-Shot Learners. arXiv, 2020. doi: 10.48550/ARXIV.2005.14165.

31. K. Lacker, Giving GPT-3 a turing test. 2020. [Електронний ресурс]. Режим доступу: https://lacker.io/ai/2020/07/06/giving-gpt-3-a-turing-test.html

32. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv, 2018. doi: 10.48550/ARXIV.1810.04805.

33. M. Lewis et al., BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. arXiv, 2019. doi: 10.48550/ARXIV.1910.13461.

34. Y. Liu et al., RoBERTa: A Robustly Optimized BERT Pretraining Approach. arXiv, 2019. doi: 10.48550/ARXIV.1907.11692.

35. Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. arXiv, 2019. doi: 10.48550/ARXIV.1909.11942.

36. A. Khaburska, Statistical and neural language models for the Ukrainian language. 2020. [Електронний ресурс]. Режим доступу: https://er.ucu.edu.ua/handle/1/2047

37. Моделі lang-uk [Електронний ресурс]. Режим доступу: https:// lang.org.ua /uk/models/

38. V. Radchenko, How to train a new language model for NLP. YouScan, 2020. [Електронний ресурс]. Режим доступу: https://youscan .io/blog/ukrainian-language-model/

39. M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz, “Building a Large Annotated Corpus of English: The Penn Treebank,” Computational Linguistics, vol. 19, no. 2, pp. 313–330, 1993, [Електронний ресурс]. Режим доступу: https://aclanthology.org/J93-2004

40. Common Crawl. [Електронний ресурс]. Режим доступу: https:// commoncrawl .org/

41. 2022. [Електронний ресурс]. Режим доступу: https://dumps. wikimedia.org/ukwiki/20220401/

42. Brown-Uk, Brown-UK/corpus: Браунський корпус української мови. [Електронний ресурс]. Режим доступу: https://github .com/brown-uk/corpus

43. H. Kucera and W. N. Francis, Computational analysis of present-day American English. Providence, RI: Brown University Press, 1967.

44. Корпуси: lang-uk. [Електронний ресурс]. Режим доступу: https://lang.org.ua/uk/corpora/#anchor4

45. Oscar-Corpus/oscar-2109 · datasets at hugging face. [Електронний ресурс]. Режим доступу: https://huggingface.co/datasets/oscar-corpus/OSCAR-2109

46. J. Abadji, P. J. O. Suárez, L. Romary, and B. Sagot, “Ungoliant: An optimized pipeline for the generation of a very large-scale multilingual web corpus,” 2021, pp. 1–9. doi: 10.14618/ids-pub-10468.

47. S. Salman and X. Liu, Overfitting Mechanism and Avoidance in Deep Neural Networks. arXiv, 2019. doi: 10.48550/ARXIV.1901.06566.

48. J. Tan, J. Yang, S. Wu, G. Chen, and J. Zhao, A critical look at the current train/test split in machine learning. arXiv, 2021. doi: 10.48550/ARXIV.2106.04525.

49. S. Raschka, Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning. arXiv, 2018. doi: 10.48550/ARXIV.1811.12808.

50. J. Ramos, Using TF-IDF to Determine Word Relevance in Document Queries. 1999.

51. S. Jabri, A. Dahbi, T. Gadi, and A. Bassir, “Ranking of text documents using TF-IDF weighting and association rules mining,” Apr. 2018, pp. 1–6. doi: 10.1109/ICOA.2018.8370597.

52. P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, “Enriching Word Vectors with Subword Information,” Transactions of the Association for Computational Linguistics, vol. 5, pp. 135–146, 2017, doi: 10.1162/tacl_a_00051.

53. P. Gage, “A New Algorithm for Data Compression,” C Users J., vol. 12, no. 2, pp. 23–38, Feb. 1994.

54. C. Wang, K. Cho, and J. Gu, Neural Machine Translation with Byte-Level Subwords. arXiv, 2019. doi: 10.48550/ARXIV.1909.03341.

55. T. Mikolov, K. Chen, G. Corrado, and J. Dean, Efficient Estimation of Word Representations in Vector Space. arXiv, 2013. doi: 10.48550/ARXIV.1301.3781.

56. R. Kulshrestha, NLP 101: Word2vec - skip-gram and CBOW. Towards Data Science, 2020. [Електронний ресурс]. Режим доступу: https://towardsdatascience.com/nlp-101-word2vec-skip-gram-and-cbow-93512ee24314

57. J. B. Lovins, “Development of a stemming algorithm,” Mech. Transl. Comput. Linguistics, vol. 11, pp. 22–31, 1968.

58. N. A. Smith, D. A. Smith, and R. W. Tromble, “Context-Based Morphological Disambiguation with Random Fields,” in Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing, Oct. 2005, pp. 475–482. [Електронний ресурс]. Режим доступу: https://aclanthology.org/H05-1060

59. R. Sennrich, B. Haddow, and A. Birch, Neural Machine Translation of Rare Words with Subword Units. arXiv, 2015. doi: 10.48550/ARXIV.1508.07909.

60. A. F. Agarap, Deep Learning using Rectified Linear Units (ReLU). arXiv, 2018. doi: 10.48550/ARXIV.1803.08375.

61. K. He, X. Zhang, S. Ren, and J. Sun, Deep Residual Learning for Image Recognition. arXiv, 2015. doi: 10.48550/ARXIV.1512.03385.

62. J. L. Ba, J. R. Kiros, and G. E. Hinton, Layer Normalization. arXiv, 2016. doi: 10.48550/ARXIV.1607.06450.

63. I. Solaiman et al., Release Strategies and the Social Impacts of Language Models. arXiv, 2019. doi: 10.48550/ARXIV.1908.09203.

64. J. Alammar, The illustrated GPT-2 (Visualizing Transformer language models). 2019. [Електронний ресурс]. Режим доступу: https://jalammar.github.io/illustrated-gpt2/

65. N. Qian, “On the momentum term in gradient descent learning algorithms,” Neural Networks, vol. 12, no. 1, pp. 145–151, 1999, doi: https://doi.org/10.1016/S0893-6080(98)00116-6.

66. A. Botev, G. Lever, and D. Barber, Nesterov’s Accelerated Gradient and Momentum as approximations to Regularised Update Descent. arXiv, 2016. doi: 10.48550/ARXIV.1607.01981.

67. Z. Hao, Y. Jiang, H. Yu, and H.-D. Chiang, Adaptive Learning Rate and Momentum for Training Deep Neural Networks. arXiv, 2021. doi: 10.48550/ARXIV.2106.11548.

68. J. C. Duchi, E. Hazan, and Y. Singer, “Adaptive Subgradient Methods for Електронний ресурс Learning and Stochastic Optimization.,” in COLT, 2010, pp. 257–269. [Електронний ресурс]. Режим доступу: https://dl.acm.org/doi/abs/10.5555/1953048.2021068

69. K. Nakamura and B.-W. Hong, Adaptive Weight Decay for Deep Neural Networks. arXiv, 2019. doi: 10.48550/ARXIV.1907.08931.

70. D. P. Kingma and J. Ba, Adam: A Method for Stochastic Optimization. arXiv, 2014. doi: 10.48550/ARXIV.1412.6980.

71. Y. Sasaki, “The truth of the F-measure,” Teach Tutor Mater, Jan. 2007.

72. T. Kuribayashi, Y. Oseki, T. Ito, R. Yoshida, M. Asahara, and K. Inui, Lower Perplexity is Not Always Human-Like. arXiv, 2021. doi: 10.48550/ARXIV.2106.01229.

73. A. Paszke et al., “PyTorch: An Imperative Style, High-Performance Deep Learning Library,” in Advances in Neural Information Processing Systems 32, H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Електронний ресурс]. Режим

доступу: http:// papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

74. Intel MKL [Електронний ресурс]. Режим доступу: https:// www.intel.com/content/www/us/en/develop/documentation/get-started-with-mkl-for-dpcpp/top.html

75. MKL-DNN[Електронний ресурс]. Режим доступу: https://oneapi-src.github.io/oneDNN/v0/index.html

76. OpenMP Home. 2021. [Електронний ресурс]. Режим доступу: https://www.openmp.org/

77. Cuda Toolkit. 2022. [Електронний ресурс]. Режим доступу: https://developer.nvidia.com/cuda-toolkit

78. Nvidia Cudnn. 2021. [Електронний ресурс]. Режим доступу: https://developer.nvidia.com/cudnn

79. T. Developers, TensorFlow. Zenodo, 2022. doi: 10.5281/zenodo.6519082.

80. H. He, “The State of Machine Learning Frameworks in 2019,” The Gradient, 2019, [Електронний ресурс]. Режим доступу: https://thegradient.pub/state-of-ml-frameworks-2019-pytorch-dominates-research-tensorflow-dominates-industry/

81. Torch.onnx. [Електронний ресурс]. Режим доступу: https://pytorch.org/docs/stable/onnx.html

82. Converting a PYTORCH* model. [Електронний ресурс]. Режим доступу: https://docs.openvino.ai/latest/openvino_docs_MO_DG_prepare_model_convert_model_Convert_Model_From_PyTorch.html

83. Triton-Inference-Server, Triton-inference-server/PYTORCH_BACKEND: The Triton backend for the PYTORCH TorchScript models. [Електронний ресурс]. Режим доступу: https://github.com/triton-inference-server/pytorch_backend

84. Mlflow.pytorch. [Електронний ресурс]. Режим доступу: https://www.mlflow.org/docs/latest/python_api/mlflow.pytorch.html

85. Torch.utils.tensorboard. [Електронний ресурс]. Режим доступу: https://pytorch.org/docs/stable/tensorboard.html

86. [Електронний ресурс]. Режим доступу: https://pytorch.org/vision/

87. [Електронний ресурс]. Режим доступу: https://pytorch.org/audio/

88. Torchgeo. [Електронний ресурс]. Режим доступу: https://torchgeo.readthedocs.io/

89. Hugging face – the AI community building the future. [Електронний ресурс]. Режим доступу: https://huggingface.co/

90. T. Wolf et al., HuggingFace’s Transformers: State-of-the-art Natural Language Processing. arXiv, 2019. doi: 10.48550/ARXIV.1910.03771.

91. Tokenizers. [Електронний ресурс]. Режим доступу: https://huggingface.co/docs/tokenizers/index

92. T. Kluyver et al., “Jupyter Notebooks – a publishing format for reproducible computational workflows,” in Positioning and Power in Academic Publishing: Players, Agents and Agendas, 2016, pp. 87–90.

93. Google Colab. Google. [Електронний ресурс]. Режим доступу: https://research.google.com/colaboratory/

94. Data Version control · DVC. [Електронний ресурс]. Режим доступу: https://dvc.org/

95. Верховна Рада України, “Про затвердження Положення про розробку інструкцій з охорони праці (ДНАОП 0.00-4.15-98),” Офіційний вебпортал парламенту України. [Електронний ресурс]. Режим доступу: https://zakon.rada.gov.ua/laws/show/z0226-98.

96. Інструкція з охорони праці для програміста. [Електронний ресурс]. Режим доступу: https://www.victorija.ua/blanki-ta-formi-dokumentiv/instruktsiya-z-ohorony-pratsi-dlya-prohramista-aktualizovano-na-08-12-2017r.html

97. L. Mou and O. Vechtomova, “Stylized Text Generation: Approaches and Applications,” in Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: Tutorial Abstracts, Jul. 2020, pp. 19–22. doi: 10.18653/v1/2020.acl-tutorials.5.

98. L. Reynolds and K. McDonell, Prompt Programming for Large Language Models: Beyond the Few-Shot Paradigm. arXiv, 2021. doi: 10.48550/ARXIV.2102.07350.

99. S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee, Generative Adversarial Text to Image Synthesis. arXiv, 2016. doi: 10.48550/ARXIV.1605.05396.

100. H. Zhang et al., StackGAN++: Realistic Image Synthesis with Stacked Generative Adversarial Networks. arXiv, 2017. doi: 10.48550/ARXIV.1710.10916.

101. A. Ramesh et al., Zero-Shot Text-to-Image Generation. arXiv, 2021. doi: 10.48550/ARXIV.2102.12092.

102. A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen, Hierarchical Text-Conditional Image Generation with CLIP Latents. arXiv, 2022. doi: 10.48550/ARXIV.2204.06125.

103. C. Wei, S. M. Xie, and T. Ma, Why Do Pretrained Language Models Help in Downstream Tasks? An Analysis of Head and Prompt Tuning. arXiv, 2021. doi: 10.48550/ARXIV.2106.09226.

104. G. Betz, C. Voigt, and K. Richardson, Critical Thinking for Language Models. arXiv, 2020. doi: 10.48550/ARXIV.2009.07185.

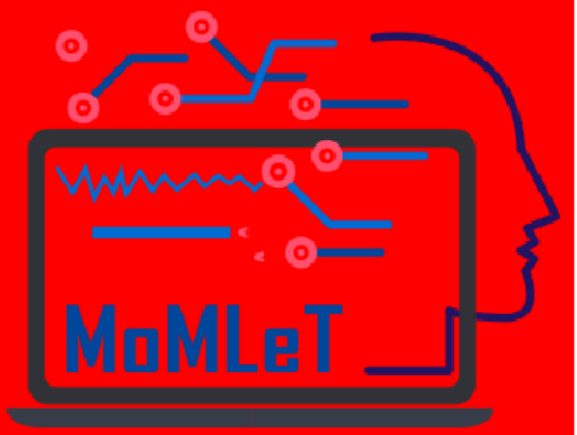
105. M. Saeed, N. Ahmadi, P. Nakov, and P. Papotti, "RuleBert: Teaching Soft Rules to Pre-trained Language Models," 2021, doi: 10.48550/ARXIV.2109.13006.

106. A. Talmor, O. Tafjord, P. Clark, Y. Goldberg, and J. Berant, Leap-Of-Thought: Teaching Pre-Trained Models to Systematically Reason Over Implicit Knowledge. arXiv, 2020. doi: 10.48550/ARXIV.2006.06609.

ДОДАТКИ

Додаток А

Vasyl Lytvyn
Michael Emmerich
Victoria Vysotska
Vitor Basto-Fernandes
Volodymyr Lytvynenko
(Eds.)



Modern Machine Learning Technologies and Data Science Workshop

Workshop Proceedings of the 10th International Conference on "Mathematics. Information Technologies. Education", MoMLeT&DS Workshop 2021

Lviv-Shatsk, Ukraine
June, 2021

Usage of Artificial Intelligence Systems and Working with the Neural Network in Assessing the Condition of Plants in Smart Greenhouses

Olha Danyltsiv, Andrii Khomiak, Oleg Nazarevych

Ternopil Ivan Puluj National Technical University, Ruska Street, 56, Ternopil, Ternopil region, 46001, Ukraine

Abstract

The paper proposes an approach to solving the problem of recognizing the state of plants using a neural network for use in information systems for monitoring and managing the microclimate for smart growing boxes and smart greenhouses. The developed smart greenhouse prototype is presented, which includes information technology for monitoring temperature, humidity, soil and recording photos of plants and further analysis of plant condition using a neural network (deeplearning). In addition, a telegram-bot interface is built for easier control light, temperature and other factors that create a favourable microclimate for a particular type of plants and manage through smartphone.

The main purpose of the work is to adapt the algorithm for training an artificial neural network using a GPU and the ability to assess the condition of plants based on photos for use in information technology control of smart greenhouse prototype.

Keywords 1

Artificial intelligence, smart greenhouse, deeplearning, plant physiology

1. Introduction

In recent decades, there has been an increased interest in neural networks in connection with the implementation of the SMART concept in various spheres of human life. Neural networks started to be explored and introduced into everyday life not only by specialists in technology, physiology and psychology but also by ordinary users. This is logical, because the artificial neural network, in fact, is a model of the natural nervous system of a living organism, so the creation and study of such networks allows us to learn more about the functioning of natural systems [1].

An important factor and a unique feature of the process is that the neural network draws certain generalized conclusions automatically. Due to its structure, it is able to analyze, compare, process information without requiring any specific programs. Another property of neural networks is reliability. Even when some elements of the network work incorrectly or simply fail, the network is still able to give the correct results, albeit with less accuracy.

There are special types of neural networks, they are able to generate an abstract image, which is based on input signals. As an example, a network is a sequence of distorted images of a symbol, letter, or punctuation mark. After training, the network will be able to generate this input without distortions.

This indicates that the network is able to generate unique conclusions based on the input information obtained during training using the finished dataset.

Thus, today the study of neural network training methods is a really relevant topic that can be applied to the concept of SMART greenhouse management.

MoMLeT+DS 2021: 3rd International Workshop on Modern Machine Learning Technologies and Data Science, June 5, 2021, Lviv-Shatsk, Ukraine

EMAIL: olhadanyltsiv@gmail.com (O. Danyltsiv); andrewhamster@hotmail.com (A. Khomiak); taltek.te@gmail.com (O. Nazarevych)

ORCID: 0000-0002-2014-0997 (O. Danyltsiv); 0000-0002-1763-868X (A. Khomiak); 0000-0002-8883-6157 (O. Nazarevych)

Додаток Б

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ПУЛЮЯ

М А Т Е Р І А Л И

ІХ НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

8–9 грудня 2021 року

ТЕРНОПІЛЬ
2021

УДК 621.326
О. Данильців – ст. гр. СНним-61, А. Хом'як – ст. гр. СНним-61, Т. Назаревич – ст. гр. ТР-204, О.Б. Назаревич – науковий керівник: к.т.н., доцент кафедри комп'ютерних наук
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)
(Технічний коледж Тернопільського національного технічного університету імені Івана Пулюя, Україна)

ВИКОРИСТАННЯ НЕЙРОННИХ МЕРЕЖ ДЛЯ ДОСЛІДЖЕННЯ СТАНУ РОСЛИН В РОЗУМНИХ ТЕПЛИЦЯХ

UDC 621.326
O. Danyltsiv, A. Khomiak, T. Nazarevych

THE USE OF NEURAL NETWORKS FOR STUDY THE CONDITION OF PLANTS IN SMART GREENHOUSES

Ключові слова: Нейронна мережа, Матриця «плутанини», Глибинне навчання.
Key words: Neural network, Confusion matrix, Deep learning.

Останні десятиліття цікавість людей до штучних нейронних мереж значно збільшилась. Це відбувається у зв'язку із впровадженням концепції SMART у різні сфери сучасного життя. Одною з унікальних можливостей таких мереж являється здатність узагальнювати інформацію та автоматично представляти висновки роботи.

Ключовим завданням даного дослідження є адаптація алгоритму тренування штучної нейронної мережі, з використанням GPU та з можливістю оцінки стану рослин на основі їх фото для застосування в інформаційній технології керування розумними міні-теплицями.

Прототип для роботи включає датчики температури, систему провітрювання, обігрів, крапельний полив рослин. Об'єкт оснащено веб-камерами для попереднього збору інформації та графічних зображень, на основі яких проводиться навчання нейронної мережі, а в подальшому формуються чіткі рекомендації стосовно догляду тої чи іншої культури.

Після проведення процесу тренування моделі отримуємо таблицю з результатами. Вона представлена на рисунку 1.

epoch	train_loss	valid_loss	accuracy	time
0	0.577004	0.066567	0.963415	00:56
1	0.281548	0.100566	0.975610	00:54
2	0.178278	0.013852	1.000000	00:54
3	0.131060	0.010825	1.000000	00:55

```
Saved weights in clf_light_0
```

Рисунок 1. Вихідна таблиця результатів тренування моделі

УДК 004.81
А. Хом'як ст. гр. СНнм-61
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

АНАЛІЗ НЕЙРОМЕРЕЖЕВИХ МОДЕЛЕЙ ПРИРОДНОЇ МОВИ

UDC 004.81
Andrii Khomiak

ANALYSIS OF NEURAL LANGUAGE MODELS

Мовна модель – статистична модель, яка дозволяє для будь-якої послідовності токенів мови визначити ймовірність того, що ця послідовність зустрінеться в природньому застосуванні цієї мови. Корисною властивістю такої моделі є можливість визначити яке продовження послідовності є найімовірнішим, що дозволяє використовувати таку модель в багатьох галузях, пов'язаних з обробкою природної мови - зокрема генерування текстів [1].

Нейронні мережі - це сімейство алгоритмів машинного навчання, які використовують комбінацію з лінійних та нелінійних перетворень даних та стохастичну оптимізацію для апроксимації довільних функцій. Через розвиток апаратних прискорювачів векторних обчислень та значне збільшення доступних обсягів даних, такий підхід дозволив значно покращити якість результатів в багатьох галузях, повязаних з когнітивно-складними задачами (машинний зір, класифікація і регресія багатовимірних даних, обробка природної мови тощо).

Найбільшого застосування в обробці природної мови (і мовному моделюванні зокрема) набули архітектури рекурентних нейронних мереж (Recurrent Neural Networks, RNN) та мереж з довго-короткотривалою пам'яттю (Long-Short Term Memory), проте з 2017 року їх витісняють підходи на базі трансформерів (Transformers) [2].

Якість мовних моделей можна оцінити за допомогою перплексії (perplexity), а оскільки ця міра легко обчислюється і не залежить від імплементації мовної моделі, то вона є хорошим інструментом для порівняння різних підходів. Експерименти з великими мовними моделями показують, що використання великих репрезентативних обсягів тексту покращують результати [3], тому тренувальні дані є важливим критерієм аналізу.

Станом на 2021 рік, поширена відсутність вільних даних та сирцевого коду в публікаціях призвела до галузевої кризи відтворюваності [4], тож наявність та доступність даних та коду тренування моделі є важливим критерієм при оцінці підходів до моделювання мови.

Література.

1. A. K. Yadav. Sentence generation from a bag of words using N-gram model [Електронний ресурс] / A. K. Yadav, S. K. Borgohain. - 2014. - Режим доступу до ресурсу: https://ieeexplore.ieee.org/document/7019414.
2. A. Vaswani. Attention Is All You Need [Електронний ресурс] / A. Vaswani et al. - 2017. - Режим доступу до ресурсу: https://arxiv.org/pdf/1706.03762.pdf.
3. A. Radford. Language Models are Unsupervised Multitask Learners [Електронний ресурс] / A. Radford et al. - 2017. - Режим доступу до ресурсу: https://cdn.openai.com /better-language-models/language_models_are_unsupervised_multitask_learners.pdf.
4. M. Hutson. Artificial intelligence faces reproducibility crisis. / M. Hutson. - 2018. - Режим доступу до ресурсу: https://www.science.org/doi/10.1126/science.359.6377.725.

Додаток В

Міністерство освіти і науки України,
Тернопільський національний технічний університет
імені Івана Пулюя
Маріборський університет (Словенія)
Технічний університет в Кошице (Словаччина)
Каунаський технологічний університет (Литва)
Львівський національний університет
імені Івана Франка,
Гірничо-металургійна академія ім. Станіслава Сташиця
(Польща)
Луцький національний технічний університет,
Чернівецький національний університет
імені Юрія Федьковича,
Вроцлавський економічний університет (Польща)
Донбаська державна машинобудівна академія

Студентське наукове товариство

IV МІЖНАРОДНА

студентська науково - технічна конференція

"ПРИРОДНИЧІ ТА ГУМАНІТАРНІ НАУКИ.

АКТУАЛЬНІ ПИТАННЯ"

28-29 квітня 2021 р.

(збірник тез конференції)

Тернопіль 2021

З М І С Т

Секція: **Інформаційні технології**

IV Міжнародна студентська науково - технічна конференція
"ПРИРОДНИЧІ ТА ГУМАНІТАРНІ НАУКИ. АКТУАЛЬНІ ПИТАННЯ"

УДК 621.326
Данильців О.–ст. гр. СНнм-51
Тернопільський національного технічного університет імені Івана Пулюя
Хом'як А.–ст. гр. СНнм-51
Тернопільський національного технічного університет імені Івана Пулюя
Назаревич Т.–ст. гр. ТР-104
Технічний коледж Тернопільського національного технічного університету імені Івана Пулюя

ВИКОРИСТАННЯ СИСТЕМИ ШТУЧНОГО ІНТЕЛЕКТУ ПРИ ОЦІНЮВАННІ СТАНУ РОСЛИН В РОЗУМНИХ ТЕПЛИЦЯХ

Науковий керівник: к.т.н., доцент Назаревич О.Б.

Danyltsiv O.
Ternopil Ivan Puluj National Technical University
Khomiak A.
Ternopil Ivan Puluj National Technical University
Nazarevich T.
Technical College of Ternopil Ivan Puluj National Technical University

USE OF THE ARTIFICIAL INTELLIGENCE SYSTEM IN ASSESSING THE CONDITION OF PLANTS IN REASONABLE GREENHOUSES

Supervisor: Ph.D., Associate Professor of Computer Science,
Nazarevich O.

Ключові слова: Розумна теплиця, штучний інтелект, нейронна мережа.
Keywords: Smart greenhouse, Artificial Intelligence, Neural network.

Розумна теплиця, що використовує засади штучного інтелекту як ключового принципу функціонування являє собою конструкцію, основні процеси в якій автоматизовані за допомогою моделі нейронних зв'язків, комп'ютерного бачення та відповідних датчиків росту рослин.

В основі StrikhaAI лежить концепція пристрою для вирощування різного виду рослин за допомогою підтримки автоматичного поливу, регулювання освітленості, температурних показників та інших чинників, які створюють сприятливий мікроклімат для насіння з подальшим його регулюванням на мобільному пристрої або ж через Інтернет.

Сам прототип містить температурні датчики, вентилятор, обігрів, полив рослин який налаштовано на автоматичну роботу, базуючись на зібраних попередньо даних. Зібрані показники формують чотири основні графіки, які подано нижче на рис. 1.

Додаток Г

Лістинг dvc.yaml

```
stages:
  download_dataset:
     cmd: python scripts/download_dataset.py data/src/
     deps:
     - scripts/download_dataset.py
     params:
     - download_dataset
     outs:
     - data/src/
  train_tokenizer:
     cmd: python scripts/train_tokenizer.py data/src/
data/tokenizer/
     deps:
     - scripts/train_tokenizer.py
     - data/src/
     outs:
     - data/tokenizer/
  prepare_dataset:
     cmd: python scripts/prepare_dataset.py data/src/
data/tokenizer/tokenizer.json data/prepared/ --temp-folder
data/cache/
     deps:
     - scripts/prepare_dataset.py
     - data/tokenizer/
     - data/src/
     params:
     - prepare_dataset
     outs:
     - data/prepared/
  train:
     cmd: python scripts/train.py data/prepared/
data/tokenizer/tokenizer.json data/model/ --checkpoint_root
data/checkpoints/
     deps:
     - scripts/train.py
     - scripts/utils/
```

```
- data/tokenizer/
- data/prepared/
params:
- train
outs:
- data/model/
metrics:
- metrics.json:
     cache: False
```

Додаток Д

Лістинг params.yaml

```
download_dataset:
  dataset:
    path: oscar
    name: "unshuffled_deduplicated_uk"

prepare_dataset:
  batch_size: 64
  block_size: 64

train:
  datasets:
     train_dataset_name: "train"
     eval_dataset_name: "validation"
  model_config: {}
  trainer_config:
     num_train_epochs: 1
     evaluation_strategy: "no"
     learning_rate: 2.e-5
     weight_decay: 0.01
     save_strategy: steps
     per_device_train_batch_size: 10
     save_total_limit: 5
     save_steps: 1000
     do_eval: False
  resume_from_checkpoint: True
```

Додаток Е

Лістинг scripts/download_dataset.py

```
     from argparse import ArgumentParser
     from pathlib import Path
     from typing import Dict, Optional

     import yaml
     from datasets import DatasetDict
     from datasets.load import load_dataset
     from loguru import logger


     def get_params(params_file: Optional[Path] =
Path("params.yaml"), key: Optional[str] = "download_dataset") ->
Dict:
          return
yaml.safe_load(params_file.read_text(encoding="utf-8"))[key]


     def describe_datasets(datasets: DatasetDict):
          logger.info(f"Loaded datasets: {datasets}")
          for key in datasets:
          logger.info(f"Dataset[{key}] examples:
{datasets[key][:1]}")

     def download_dataset(target_dir: Path):
          params = get_params()
          datasets: DatasetDict =
load_dataset(**params["dataset"])
          describe_datasets(datasets)
          datasets.save_to_disk(target_dir)


     if __name__ == "__main__":
          parser = ArgumentParser()
          parser.add_argument("dir", type=Path)
          args = parser.parse_args()

          download_dataset(target_dir=args.dir)
```

Додаток Є

Лістинг scripts/train_tokenizer.py

```
     from argparse import ArgumentParser
     from pathlib import Path
     from typing import Iterable, Iterator, Optional

     from tokenizers import Tokenizer
     import tokenizers
     from tokenizers.models import BPE
     from tokenizers.trainers import BpeTrainer, Trainer
     from tokenizers.pre_tokenizers import Whitespace
     from tokenizers.processors import PostProcessor
     from datasets.arrow_dataset import Dataset
     import datasets


     def get_tokenizer() -> Tokenizer:
          tokenizer = Tokenizer(BPE(unk_token="[UNK]"))
          tokenizer.pre_tokenizer = Whitespace()
          tokenizer.enable_padding()
          return tokenizer


     def get_trainer() -> Trainer:
          return BpeTrainer(special_tokens=['[UNK]', '[SEP]',
'[PAD]', '[MASK]'])


     def get_dataset(input_folder: Path) -> Dataset:
          return
datasets.load.load_from_disk(input_folder)["train"]


     def get_dataset_iterator(input_folder: Path,
                              batch_size: Optional[int] = 1024)
-> Iterator[Iterable[str]]:
          dataset = get_dataset(input_folder)
          for i in range(0, len(dataset), batch_size):
          yield dataset[i: i + batch_size]["text"]
```

```
     def train_tokenizer(input_folder: Path, output_folder:
Path):
          tokenizer = get_tokenizer()
          trainer = get_trainer()
          dataset_iterator = get_dataset_iterator(input_folder)
          tokenizer.train_from_iterator(dataset_iterator,
trainer=trainer)
          output_folder.mkdir(exist_ok=True, parents=True)
          tokenizer.save(str(output_folder/"tokenizer.json"))


     if __name__ == "__main__":
          parser = ArgumentParser()
          parser.add_argument("input_folder", type=Path)
          parser.add_argument("output_folder", type=Path)
          args = parser.parse_args()

          train_tokenizer(input_folder=args.input_folder,
                          output_folder=args.output_folder)
```

Додаток Ж

Лістинг scripts/prepare_dataset.py

```
import shutil
from argparse import ArgumentParser
from multiprocessing import cpu_count
from pathlib import Path
from pprint import pformat
from tempfile import gettempdir
from typing import Dict, Optional

import datasets
import yaml
from datasets import DatasetDict
from loguru import logger
from tokenizers import Tokenizer
from transformers import PreTrainedTokenizerFast


def get_params(params_file: Optional[Path] =
Path("params.yaml"),
               stage_name: Optional[str] =
"prepare_dataset") -> Dict:
    return
yaml.safe_load(params_file.read_text(encoding="utf-
8"))[stage_name]


def get_tokenizer(tokenizer_file: Path) -> Tokenizer:
    tokenizer =
PreTrainedTokenizerFast(tokenizer_file=str(tokenizer_file))
    logger.info(f"Loaded {tokenizer}")
    return tokenizer


def get_dataset(input_folder: Path) -> DatasetDict:
    dataset = datasets.load.load_from_disk(input_folder)
    logger.info(f"Loaded {dataset}")
    return dataset
```

```
     def get_cache_file_names(temp_folder: Path,
clear_cache_folder: bool) -> Dict:
          if clear_cache_folder and temp_folder.exists():
          shutil.rmtree(temp_folder, ignore_errors=True)

          temp_folder.mkdir(exist_ok=True, parents=True)

          # TODO: add evaluation set
          return {name: (temp_folder/f"{name}.arrow").as_posix()
for name in ("train", "test", "validation")}


     def describe_datasets(datasets: DatasetDict):
          logger.info(f"Got datasets: {datasets}")
          for key in datasets:
          logger.info(f"Dataset[{key}] examples:
{datasets[key][:1]}")


     def get_lm_datasets(input_folder: Path, tokenizer_file:
Path, temp_folder: Path, params: Dict, clear_cache_folder:
Optional[bool] = True) -> DatasetDict:
          batch_size = params["batch_size"]
          block_size = params["block_size"]
          tokenizer = get_tokenizer(tokenizer_file)

          def tokenize_function(examples):
          return tokenizer(examples["text"])
          dataset = get_dataset(input_folder=input_folder)
          logger.info(f"Mapping {dataset} with
{tokenize_function}")
          cache_file_names =
get_cache_file_names(temp_folder=temp_folder,

     clear_cache_folder=clear_cache_folder)
          current_columns = dataset["train"].features
          tokenized_datasets = dataset.map(tokenize_function,
                                           batched=True,
                                           batch_size=batch_size,
                                           num_proc=cpu_count(),
```

```
     cache_file_names=cache_file_names,

     remove_columns=current_columns)

          describe_datasets(tokenized_datasets)

          def group_texts(examples):
          # Concatenate all texts.
          try:
               concatenated_examples = {
                    k: sum(examples[k], []) for k in
examples.keys()}
          except:
               logger.error(f"{pformat(examples,
compact=True)}")
               raise
          total_length =
len(concatenated_examples[list(examples.keys())[0]])
          # We drop the small remainder, we could add padding if
the model supported it instead of this drop, you can
          # customize this part to your needs.
          total_length = (total_length // block_size) *
block_size
          # Split by chunks of max_len.
          result = {
               k: [t[i: i + block_size]
                    for i in range(0, total_length, block_size)]
               for k, t in concatenated_examples.items()
          }
          result["labels"] = result["input_ids"].copy()
          return result

          logger.info(f"Mapping {tokenized_datasets} with
{group_texts}")
          lm_datasets = tokenized_datasets.map(
          group_texts,
          batched=True,
          batch_size=batch_size,
          num_proc=cpu_count(),
          )
```

```
          describe_datasets(lm_datasets)
          return lm_datasets


     def prepare_dataset(dataset_folder: Path, tokenizer_file:
Path, output_folder: Path, temp_folder: Path):
          output_folder.mkdir(exist_ok=True, parents=True)
          params = get_params()
          lm_datasets =
get_lm_datasets(input_folder=dataset_folder,

     tokenizer_file=tokenizer_file,
                                        temp_folder=temp_folder,
                                        params=params)
          lm_datasets.save_to_disk(str(output_folder))
          shutil.rmtree(temp_folder, ignore_errors=True)


     if __name__ == "__main__":
          parser = ArgumentParser()
          parser.add_argument("dataset_folder", type=Path)
          parser.add_argument("tokenizer_file", type=Path)
          parser.add_argument("output_folder", type=Path)
          parser.add_argument("--temp-folder", type=Path,
                              required=False,
default=gettempdir())
          args = parser.parse_args()

          prepare_dataset(dataset_folder=args.dataset_folder,
                          tokenizer_file=args.tokenizer_file,
                          output_folder=args.output_folder,
                          temp_folder=args.temp_folder)
```

Додаток З

Лістинг scripts/train.py

```
    import json
    import math
    from argparse import ArgumentParser
    from pathlib import Path
    from typing import Dict, Optional, Tuple

    import datasets
    import yaml
    from checksumdir import dirhash
    from hashlib import md5
    from datasets import DatasetDict
    from loguru import logger
    from tokenizers import Tokenizer
    from transformers import (DataCollatorForLanguageModeling,
GPT2Config,
                              GPT2LMHeadModel,
PreTrainedTokenizerFast, Trainer,
                              TrainingArguments)
    from transformers.tokenization_utils_base import
PreTrainedTokenizerBase
    from utils import colab


    def get_params(params_file: Optional[Path] =
Path("params.yaml"),
                   stage_name: Optional[str] = "train") ->
Dict:
        return
yaml.safe_load(params_file.read_text(encoding="utf-
8"))[stage_name]


    def get_tokenizer(tokenizer_file: Path) ->
PreTrainedTokenizerBase:
        tokenizer_object: Tokenizer =
Tokenizer.from_file(str(tokenizer_file))
        logger.info(f"Tokenizer padding:
{tokenizer_object.padding}")
```

```
          tokenizer: PreTrainedTokenizerBase =
PreTrainedTokenizerFast(
          tokenizer_object=tokenizer_object)
          tokenizer.pad_token =
tokenizer_object.padding["pad_token"]
          logger.info(f"TokenizerFast: {tokenizer}")
          return tokenizer


     def get_trainer(model: GPT2LMHeadModel,
                     lm_datasets: DatasetDict,
                     data_collator:
DataCollatorForLanguageModeling,
                     params: Dict,
                     checkpoint_folder: Path) -> Trainer:
          config =
TrainingArguments(output_dir=str(checkpoint_folder),
                                    overwrite_output_dir=True,
                                    **params["trainer_config"])
          trainer = Trainer(model=model,
                            args=config,
                            data_collator=data_collator,

     train_dataset=lm_datasets[params["datasets"]

     ["train_dataset_name"]],

     eval_dataset=lm_datasets[params["datasets"]

     ["eval_dataset_name"]],
                           )
          logger.info(f"Trainer: {trainer}, with config:
{config}")
          logger.info(
          f"train_dataset: {trainer.train_dataset}
({len(trainer.train_dataset)})")
          return trainer


     def get_collator(tokenizer: PreTrainedTokenizerBase) ->
DataCollatorForLanguageModeling:
```

```
          collator =
DataCollatorForLanguageModeling(tokenizer=tokenizer, mlm=False)
          logger.info(f"Got collator: {collator}")
          return collator


     def get_model(params: Dict) -> GPT2LMHeadModel:
          config = GPT2Config(**params["model_config"])
          return GPT2LMHeadModel(config)


     def get_lm_datasets(input_folder: Path) -> DatasetDict:
          lm_datasets =
datasets.load.load_from_disk(input_folder)
          logger.info(f"LM datasets: {lm_datasets},
{len(lm_datasets)}")
          return lm_datasets


     def save_metrics(metrics: Dict,
                      metrics_file: Optional[Path] =
Path("metrics.json")):
          metrics_file.parent.mkdir(exist_ok=True, parents=True)
          metrics_file.write_text(json.dumps(metrics))


     def get_dataset_hash(dataset_folder: Path,
                          params_file: Optional[Path] =
Path("params.yaml")) -> str:
          params_content_hash =
md5(params_file.read_bytes()).digest()
          logger.info(f"Computing hash for {dataset_folder}")
          dir_hash = dirhash(dataset_folder).encode("utf-8")
          result_hash = md5(params_content_hash +
dir_hash).hexdigest()
          return result_hash


     def get_checkpoint_config(dataset_folder: Path,
                               checkpoint_root: Path, params:
Dict) -> Tuple[Path, bool]:
```

```
          dataset_hash =
get_dataset_hash(dataset_folder=dataset_folder)
          checkpoint_folder = checkpoint_root/dataset_hash
          checkpoints_exist = checkpoint_folder.exists() and
any(
          checkpoint_folder.glob("checkpoint-*"))
          resume_enabled = params["resume_from_checkpoint"]
          if resume_enabled and not checkpoints_exist:
          checkpoint_folder.mkdir(parents=True, exist_ok=True)
          resume = resume_enabled and checkpoints_exist
          return checkpoint_folder, resume


     def train(dataset_folder: Path,
               tokenizer_file: Path,
               output_folder: Path,
               checkpoint_root: Path):
          output_folder.mkdir(exist_ok=True, parents=True)
          params = get_params()
          tokenizer = get_tokenizer(tokenizer_file)
          collator = get_collator(tokenizer)
          lm_datasets =
get_lm_datasets(input_folder=dataset_folder)
          model = get_model(params)
          checkpoint_folder, resume_from_checkpoint =
get_checkpoint_config(dataset_folder=dataset_folder,

     checkpoint_root=checkpoint_root,

     params=params)
          trainer = get_trainer(model=model,
                                lm_datasets=lm_datasets,
                                params=params,
                                data_collator=collator,

     checkpoint_folder=checkpoint_folder)
          colab.ensure_colab_compatibility()
          trainer.train(resume_from_checkpoint=resume_from_check
point)
          trainer.save_model(output_folder)
          eval_results = trainer.evaluate()
```

```
        metrics = {
        "perplexity": 2 ** eval_results['eval_loss']
        }
        save_metrics(metrics)


    if __name__ == "__main__":
        parser = ArgumentParser()
        parser.add_argument("dataset_folder", type=Path)
        parser.add_argument("tokenizer_file", type=Path)
        parser.add_argument("output_folder", type=Path)
        parser.add_argument("--checkpoint_root", type=Path,
required=True)
        args = parser.parse_args()

        train(dataset_folder=args.dataset_folder,
            tokenizer_file=args.tokenizer_file,
            output_folder=args.output_folder,
            checkpoint_root=args.checkpoint_root)
```

Додаток І

Лістинг scripts/utils/text_generation.py

```
     from argparse import ArgumentParser
     from pathlib import Path
     from typing import List
     from loguru import logger
     from torch.nn.modules.module import Module

     from transformers import GPT2LMHeadModel,
PreTrainedTokenizerFast, PreTrainedTokenizerBase
     from tokenizers import Tokenizer


     def get_tokenizer(tokenizer_file: Path) ->
PreTrainedTokenizerBase:
          tokenizer_object: Tokenizer =
Tokenizer.from_file(str(tokenizer_file))
          tokenizer: PreTrainedTokenizerBase =
PreTrainedTokenizerFast(
          tokenizer_object=tokenizer_object)
          tokenizer.pad_token =
tokenizer_object.padding["pad_token"]
          return tokenizer

     def count_params(model: Module) -> int:
          return sum(p.numel() for p in model.parameters())

     def get_model(model_folder: Path) -> GPT2LMHeadModel:
          model = GPT2LMHeadModel.from_pretrained(model_folder)
          logger.info(f"Loaded {model} with
{count_params(model)} parameters")
          return model


     def get_generated_sequences(prompt: str, tokenizer:
PreTrainedTokenizerFast, model: GPT2LMHeadModel) -> List[str]:
          inputs = tokenizer(prompt, return_tensors="pt")
          output = model.generate(
          **inputs,
          max_length=50,
```

```
          num_beams=5,
          early_stopping=True,
          num_return_sequences=5,
          no_repeat_ngram_size=2,)
          return [
          tokenizer.decode(sequence, skip_special_tokens=True)
          for sequence in output
          ]


     def generate_text(prompt: str, tokenizer_file: Path,
model_folder: Path):
          tokenizer =
get_tokenizer(tokenizer_file=tokenizer_file)
          model = get_model(model_folder)
          sequences = get_generated_sequences(prompt=prompt,

     tokenizer=tokenizer,
                                              model=model)
          print("\n".join(sequences))


     if __name__ == "__main__":
          parser = ArgumentParser()
          parser.add_argument("prompt", type=str)
          parser.add_argument("--tokenizer", type=Path,
required=False,

     default=Path("data/tokenizer/tokenizer.json"))
          parser.add_argument("--model", type=Path,
required=False,
                              default=Path("data/model/"))
          args = parser.parse_args()

          generate_text(prompt=args.prompt,
                    tokenizer_file=args.tokenizer,
model_folder=args.model)
```