НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
Факультет прикладної математики
Кафедра прикладної математики

«На правах рукопису»

УДК 004.62:510.22:004.023

«До захисту допущено»
Завідувач кафедри
_____ Олег ЧЕРТОВ
«___» _____ 2024 р.

Магістерська дисертація

на здобуття ступеня магістра
за освітньо-професійною програмою «Наука про дані та математичне моделювання»
зі спеціальності 113 «Прикладна математика»
на тему: «Математичне та програмне забезпечення системи семантичного аналізу відгуків на заклади харчування»

Виконав: студент ІІ курсу, групи КМ-11мпв
Бевзюк Костянтин Андрійович _____

Керівник:
Старший викладач, канд. техн. наук
Ліскін В’ячеслав Олегович _____

Консультант з нормоконтролю:
старший викладач,
Мальчиков Володимир Вікторович _____

Рецензент:
Доцент, канд. техн. наук, доцент каф. СПіСКС
Тарасенко-Клятченко Оксана Володимирівна _____

Засвідчую, що в цій магістерській дисертації немає запозичень із праць інших авторів без відповідних посилань.
Студент _____

Київ — 2024

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра прикладної математики

Рівень вищої освіти — другий (магістерський)

Спеціальність – 113 «Прикладна математика»

Освітньо-професійна програма «Наука про дані та математичне моделювання»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олег ЧЕРТОВ
«___» _____ 2024 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Бевзюку Костянтину Андрійовичу

1. Тема дисертації: «Математичне та програмне забезпечення системи семантичного аналізу відгуків на заклади харчування», керівник дисертації, Ліскін Вячеслав Олегович, старший викладач, канд. техн. наук, затверджені наказом по університету від «28» грудня 2023 р. № 5822 – С.
2. Термін подання студентом дисертації: «09» січня 2024 р
3. Об’єкт дослідження: Методи обробки великих обсягів текстових даних, методи побудови глибинних нейронних на базі архітектури рекурентних нейронних зв’язків та “трансформерів”, розробка, підбір параметрів та навчання моделі, яка зможе класифікувати текстові дані.
4. Предмет дослідження: математичне та програмне забезпечення системи класифікації відгуків на заклади харчування за допомогою нейронних мереж на базі “трансформерів”, це дозволить дати кількісну оцінку тексту і може бути

використано для покращення якості обслуговування та задоволення потреб клієнтів у сфері гастрономії та гостинності.

5. Перелік завдань, які потрібно розробити: огляд, аналіз, вибір оптимального методу вирішення поставленої задачі, розробити програмне забезпечення та протестувати його.

6. Перелік ілюстративного матеріалу: рисунки, схеми, спрощена схема взаємодії модулів системи, знімки екранних форм.

7. Орієнтовний перелік публікації: Ліскін В.О. та Бевзюк К.А. тези конференції на тему: «Математичне та програмне забезпечення системи семантичного аналізу відгуків на заклади харчування».

8. Дата видачі завдання: «1» вересня 2023

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Ґрунтовне ознайомлення з предметною областю	15.09.2022	
2	Визначення структури магістерської дисертації; вивчення літератури, пошук додаткової літератури	01.10.2022	
3	Робота над першим розділом магістерської дисертації	15.12.2022	
4	Проведення наукового дослідження; робота над другим розділом магістерської дисертації	15.01.2023	
5	Проведення наукового дослідження; робота над статтею за результатами наукового дослідження	15.04.2023	

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
6	Робота над третім розділом магістерської дисертації; підготовка статті за результатами наукового дослідження; розроблення програмного забезпечення	01.07.2023	
7	Завершення роботи над основною частиною магістерської дисертації	15.11.2023	
8	Оформлення текстової і графічної частин магістерської дисертації	25.12.2023	

Студент _____ Костянтин БЕВЗЮК

Керівник роботи _____ Вячеслав ЛІСКІН

РЕФЕРАТ

Дисертацію виконано на 81 аркушах, вона містить 2 додатки та перелік посилань на використані джерела з 24 найменувань. У роботі наведено 51 рисунків та 4 таблиць.

Актуальність теми. У світі де інформація змінюється щосекунди дуже важко услідкувати за всім разом, тому чисельна оцінка відгуків закладів харчування спрощує наше повсякденне життя, це тим самим прискорює процеси аналізу інформації

Зв'язок роботи з науковими програмами, планами, темами. Дисертаційна робота виконувалась згідно з планом науково-дослідних робіт кафедри прикладної математики Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

Мета і задачі дослідження. Метою дослідження є семантичний аналіз відгуків на заклади харчування, в якості прикладу буде взято кав'ярню «Starbucks».

Для досягнення вказаної мети було розв'язано такі задачі:

- проаналізувати існуючі рішення та знайти оптимальне для вирішення семантичного аналізу поставленої задачі;
- створити математичну модель рішення для розв'язку семантичного аналізу;
- імплементувати програмне забезпечення для обранної математичної моделі;
- створити інтерфейс користувача для роботи з програмним засобом;

Об'єктом дослідження є розробка, підбір параметрів та навчання моделі, яка зможе класифікувати великі обсяги текстових даних.

Предметом дослідження є математичне та програмне забезпечення системи класифікації відгуків на заклади харчування за допомогою нейронних мереж на базі «трансформерів», це дозволить дати кількісну оцінку тексту і може бути

використано для покращення якості обслуговування та задоволення потреб клієнтів у сфері гастрономії та гостинност

Методи дослідження. Для розв’язання поставленої задачі використовувалися такі методи: нейронні мережі на базі рекурентних нейронних зв’язків, нейронні мережі на базі «трансформерів».

Наукова новизна одержаних результатів. Було запропоновано безкоштовний веб-застосунок для семантичного аналізу коментарів на заклади харчування. Запроваджено новий метод розподілу вхідного датасету на тествоий, валідаційний та тренувальні частини.

Практичне значення одержаних результатів. В майбутньому це дає поштовх розширювати моделі для обробки людської мови, а саме даний приклад легко масштабувати на базу української мови або будь-якої іншої.

Апробація результатів дисертації. Основні положення й результати роботи представлено на XVI науково-практичній конференції магістрантів та аспірантів «Прикладна математика та комп’ютинг – ПМК-2023» (Київ, 28-30 листопада 2023 р.) та опубліковані у збірнику тез за результатами конференції.

Публікації. Результати дисертації викладено в 1 науковій праці:

- 1 публікація у тезах конференцій.

Ключові слова: Семантичний аналіз, нейронні мережі, RNN, трансформери

ABSTRACT

The dissertation consists of 81 pages, including 2 appendices and a list of references with 25 entries. The work features 51 figures and 4 tables.

Topic relevance. In a world where information changes every second, keeping track of everything is challenging. Therefore, numerical evaluation of feedback on food establishments simplifies our daily lives and accelerates information analysis processes.

Thesis connection to scientific programs, plans, and topics. The dissertation was carried out according to the plan of scientific research of the Department of Applied Mathematics at the National Technical University of Ukraine "Kyiv Polytechnic Institute named after Igor Sikorsky."

Research goal and objectives. The research aims to conduct semantic analysis of feedback on food establishments, using the example of the "Starbucks" cafe. To accomplish this goal, the following objectives were reached:

- Analyze existing solutions and find the optimal one for semantic analysis.
- Create a mathematical model for solving semantic analysis.
- Implement software for the chosen mathematical model.
- Create a user interface for working with the software.

Object of research is the development, parameter selection, and training of a model capable of classifying large volumes of textual data.

Subject of research is the mathematical and software components of the feedback classification system using neural networks based on "transformers," which can provide a quantitative assessment of the text and improve service quality in the gastronomy and hospitality sector.

Methods of research. The following methods were used to solve the research problem: neural networks based on recurrent connections, neural networks based on "transformers."

Scientific contribution. A free web application for semantic analysis of comments on food establishments was proposed. A new method of dividing the input dataset into test, validation, and training parts was introduced.

Practical value of obtained results. In the future, this opens up opportunities to expand models for natural language processing, making it easily scalable to the Ukrainian language or any other language.

Approbation of thesis results. The main findings and results of the work were presented at the XVI Scientific-Practical Conference of Masters and PhD Students "Applied Mathematics and Computing – AMC-2023" (Kyiv, November 28-30, 2023) and published in the conference proceedings.

Publications. The results of the dissertation are presented in 1 scientific work:

- 1 publication in conference proceedingsю

Keywords: Semantic analysis, neural networks, RNN, transformers.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

GRU (Gated Recurrent Unit) – це інший тип рекурентних нейронних мереж, подібний до LSTM, який також розроблений для вирішення проблем зникаючого градієнту та ефективного моделювання послідовностей.

LSTM (Long Short-Term Memory) – це рекурентна нейронна мережа спеціально призначена для роботи з послідовностями даних та вирішення проблем зникаючого градієнту, які можуть виникати у звичайних рекурентних нейронних мереж.

RNN (Recurrent Neural Network) – рекурентна нейронна мережа, яка побудована на основі LSTM, GRU модулів.

UML (Unified Modeling Language) – уніфікована мова моделювання, що використовується у парадигмі об'єктно-орієнтованого програмування.

URL – уніфіко́ваний лока́тор ресу́рсів або адре́са ресу́рсу (англ. Uniform Resource Locator — єдиний вказівник на ресурс, URL) – стандартизована адреса певного ресурсу

ВСТУП

У світі, насиченому гастрономічними враженнями та технологічними нововведеннями, взаємодія споживачів із закладами харчування набуває нового виміру. Однак зі зростанням кількості відгуків у Інтернеті суттєво зростає і необхідність в розробці систем семантичного аналізу, які не лише фіксують кількість та якість відгуків, а й здатні розуміти їхню сутність.

Системи семантичного аналізу відгуків на заклади харчування стають невід'ємною частиною стратегій управління репутацією та покращення обслуговування. Ці системи використовують в собі поєднання математичних алгоритмів та програмного забезпечення для виявлення семантичних відтінків у текстових відгуках, що дає змогу зрозуміти реальні враження клієнтів. Вони стають важливим інструментом для власників гастрономічних закладів, дозволяючи адаптувати своє обслуговування та реагувати на потреби клієнтів, що забезпечує не лише задоволення смакових пристрастей, але й неперевершені враження від обслуговування. У цьому контексті, розвиток та вдосконалення систем семантичного аналізу стає критичним елементом для підтримки конкурентоспроможності гастрономічних закладів в епоху активної цифрової взаємодії. Шляхом аналізу коментарів також можна визначити, які страви користуються попитом, а які можуть потребувати удосконалення.

1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

Семантичний аналіз – це завдання обробки природної мови, спрямоване на визначення настроїв або емоцій, виражених у фрагменті тексту. Це завдання передбачає аналіз письмової чи усної мови, щоб класифікувати її як позитивну, негативну чи нейтральну, надаючи розуміння суб’єктивних почуттів чи думок, які передає автор.

З розвитком соціальних медіа, відгуків клієнтів та інших онлайн-платформ аналіз настроїв стає все більш важливим для компаній і організацій. Це дозволяє їм оцінювати громадську думку, розуміти відгуки клієнтів і приймати рішення на основі даних. Моделі аналізу настроїв використовують технології машинного навчання та обробки природної мови для автоматичної класифікації та інтерпретації тексту.

Алгоритми керованого машинного навчання, включно з опорними векторними машинами та нейронними мережами, зазвичай використовуються для аналізу настроїв. Ці моделі навчаються на позначених наборах даних, що містять текстові зразки з пов’язаними мітками настроїв. Крім того, лексикони настроїв і підходи на основі правил можуть підвищити точність аналізу настроїв, особливо коли мова йде про певні домени чи мови.

Застосування сематичного аналізу різноманітні, починаючи від моніторингу брендів і аналізу відгуків про продукт і закінчуючи відстеженням політичних настроїв. Витягуючи настрої з текстових даних, компанії можуть адаптувати свої стратегії, підвищити рівень задоволеності клієнтів і ефективно реагувати на тенденції настроїв суспільства.

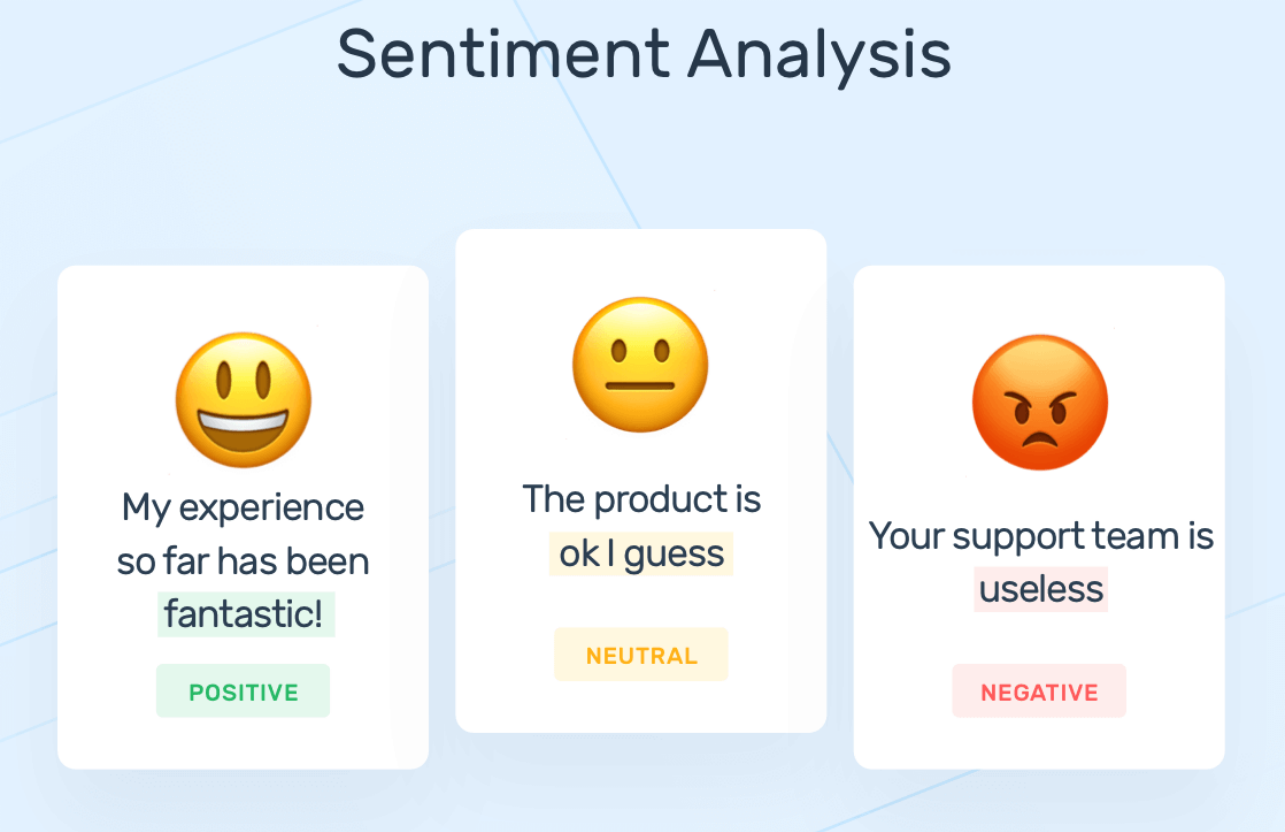


Рисунок 1.1 – Приклад семантичного аналізу тексту [1]

Сьогодні технології та рішення у сфері семантичного аналізу тексту розвиваються над швидкими темпами і кількість таких рішень зростає щохвилини, одними з провідних рішень для цієї задачі являються нейронні мережі, особливо архітектури на базі рекурентних зв’язків (RNN) та на базі трансформерів (Transformer). Надалі будемо глибше поринати у ці дві архітектури та оцінювати існуючі рішення.

1.1 Нейронні мережі на базі рекурентних нейронних зв’язків

Рекурентні нейронні мережі (RNN) — це клас нейронних мереж, призначених для обробки послідовних даних, що робить їх особливо придатними для завдань, що включають шаблони в часі та аналіз природньої мови. На відміну

від традиційних нейронних мереж прямого зв’язку, RNN мають з’єднання, які утворюють спрямовані цикли, що дозволяє їм підтримувати прихований стан, який фіксує інформацію про попередні вхідні дані в послідовності.

Ключовою особливістю RNN є їхня здатність зберігати пам’ять про минулі вхідні дані, дозволяючи їм враховувати контекст і залежності в послідовних даних. Це робить їх ефективними для таких завдань, як обробка природної мови, розпізнавання мовлення та прогнозування часових рядів. Однак стандартні мережі RNN стикаються з проблемами, пов’язаними з фіксацією довгострокових залежностей через проблеми зі зникаючим і зростаючим градієнтом, що обмежує їхню ефективність у певних сценаріях.

Щоб вирішити ці проблеми, були розроблені варіації RNN, такі як мережі довготривалої короткочасної пам’яті (LSTM) і Gated Recurrent Units (GRU). Ці архітектури включають спеціалізовані механізми, такі як шлюзи, щоб контролювати потік інформації та пом’якшувати проблему зникнення градієнта, дозволяючи їм більш ефективно фіксувати та використовувати довготривалі залежності.

1.1.2 Рекурентні нейронні мережі (RNN)

Традиційні нейронні мережі, розглядають кожне спостереження як незалежне, оскільки мережі не здатні зберігати минулу або історичну інформацію. Саме це призвело до появи рекурентних нейронних мереж (RNN), які вводять концепцію пам’яті в нейронні мережі, включаючи залежність між точками даних. Завдяки цьому RNN можна навчити запам’ятовувати концепції на основі контексту, тобто вивчати повторювані шаблони. RNN «запам’ятовують» через цикл зворотного зв'язку в комірці. І це головна відмінність RNN від традиційної нейронної мережі. Цикл зворотного зв’язку дозволяє передавати інформацію в

межах рівня на відміну від нейронних мереж прямого зв’язку, в яких інформація передається лише між шарами. – [2]

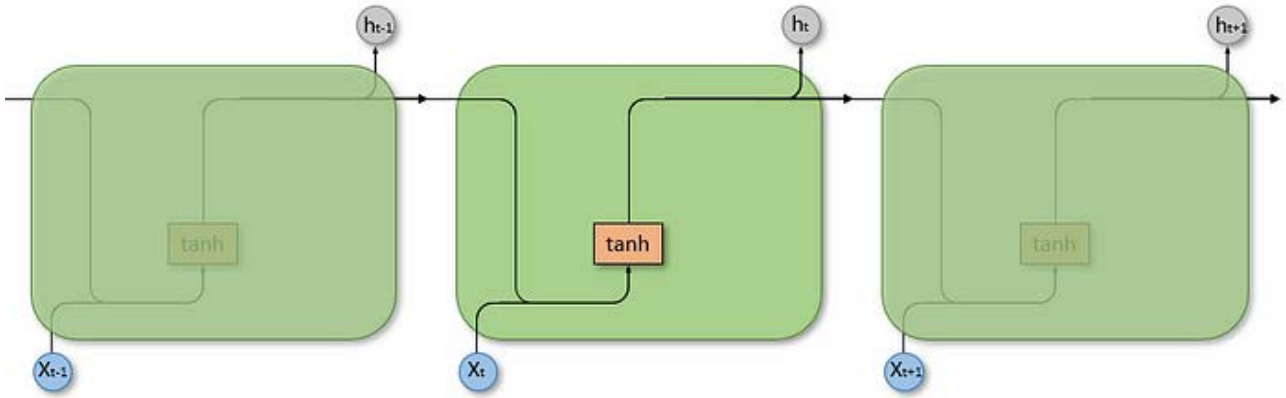


Рисунок 1.2 - Цикл зворотного зв’язку [2]

Через зворотний зв'язок вихід однієї клітини рекурентної нейронної мережі (RNN) також використовується як вхід для цієї ж самої клітини. Отже, кожна клітина має два входи: минуле та теперішнє. Використання інформації про минуле призводить до короткотермінової пам'яті.

Для кращого розуміння ми розгортаємо/розкриваємо зворотний зв'язок клітини RNN. Довжина розгорнутої клітини дорівнює кількості кроків часу вхідної послідовності. Ми можемо спостерігати, як минулі спостереження передаються через розгорнуту мережу у вигляді прихованого стану. В кожній клітині вхід поточного кроку часу x (поточне значення), прихований стан h з попереднього кроку часу (минуле значення) і зміщення об'єднуються, а потім обмежуються функцією активації для визначення прихованого стану поточного кроку часу.

$$h_t = tanh(W_x x_t + W_h h_{t-1} + b) \quad (1.1)$$

Тут маленькі літери представляють вектори, тоді як великі літери - матриці.

Ваги W у RNN оновлюються за допомогою алгоритму зворотного поширення помилок у часі. (backpropagation)

Рекурентні нейронні мережі можуть використовуватися для прогнозування відносно одного-до-одного, одного-до-багатьох, багатьох-до-одного та багатьох-до-багатьох.

Переваги RNN:

- Обробка послідовних даних – RNN можуть ефективно вирішувати завдання з послідовними даними та визначати закономірності в історичних даних завдяки своїй короткочасній пам'яті.
- Робота з різною довжиною вхідних даних: RNN можуть успішно опрацьовувати вхідні дані різної довжини, що дозволяє їм працювати з різними форматами даних.

Недоліки RNN:

- Проблема зникаючого грададієнту: RNN стикаються із проблемою зникання градієнта, де градієнти, які використовуються для оновлення вагів під час зворотнього розповсюдження помилок, стають дуже малими. Помноження ваг на градієнт, який близький до нуля, унеможливлює мережі вивчати нові ваги. Це призводить до зупинки навчання, і, в результаті, RNN забуває, що бачило в довших послідовностях. Проблема зникання градієнта поглиблюється зі збільшенням кількості шарів у мережі.
- Обмежений розгляд минулих спостережень: Оскільки RNN зберігає лише останню інформацію, модель має проблеми при розгляді спостережень, які віддалені в минулому. RNN тенденційно втрачає інформацію протягом довгих послідовностей, оскільки вона зберігає лише найновішу інформацію. Таким чином, у RNN є тільки короткочасна, а не довгочасна пам'ять.
- Зворотня проблема до зникаючих градієнтів – «вибухаючі» градієнти. Оскільки RNN використовує зворотнє розповсюдження помилок в часі для оновлення коефіцієнтів, мережа також стикається із проблемою вибухання градієнтів і, при використанні активаційних функцій ReLU, із "мертвих" одиниць ReLU - це нейрони в нейронних мережах, які втратили свою

здатність активуватися і завжди повертають нульове значення на вході. Це становище може виникнути під час навчання, коли ваги або зсуви (biases) в одиницях ReLU налаштовуються так, що вихід завжди буде від'ємним.. Перше може призводити до проблем із збіжністю, а друге - до зупинки навчання моделі.

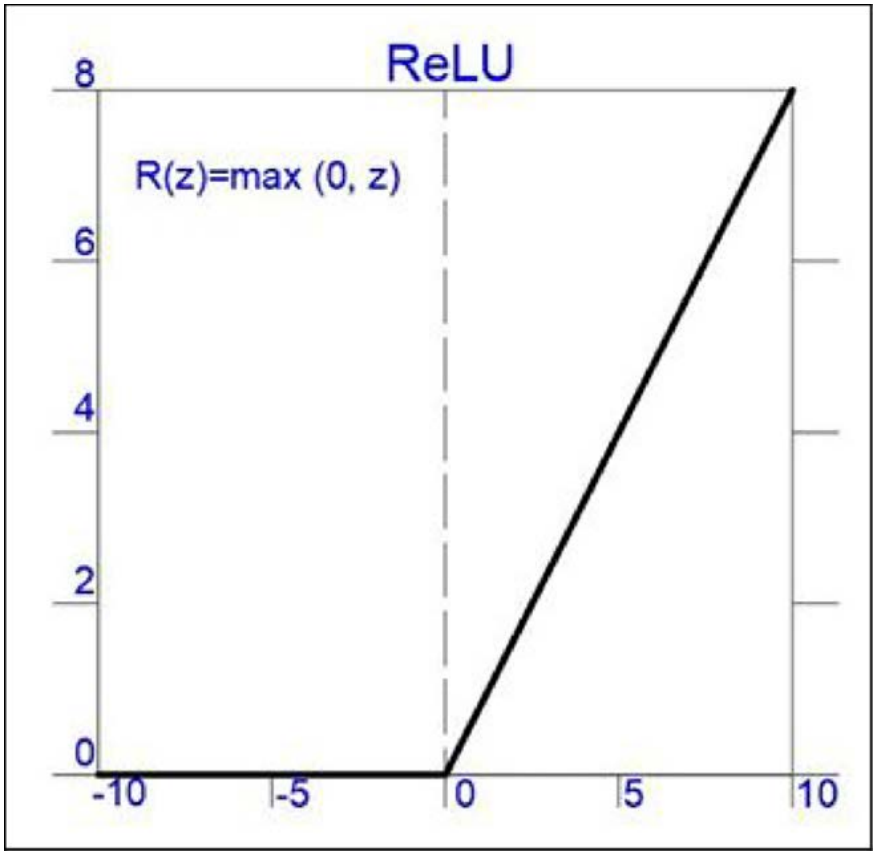


Рисунок 1.3 – Графік функції ReLU [3]

Доволі часто ReLU використовують саме для боротьби з «зникаючими» градієнтами, оскільки як видно на рисунку 1.3 ця функція є гладкою усюди крім точки 0, не зважаючи на це функція є доволі популярним вибором серед інших функцій активації у нейронних мережах.

1.1.3 Рекурентні нейронні мережі на основі LSTM шарів

На відміну від звичайних рекурентних нейронних мереж, мережі на основі LSTM шарів це особливий тип рекурентних нейронних мереж, який вирішує основну проблему простих RNN, а саме проблему «зникаючих» градієнтів, тобто проблему втрати інформації, яка знаходиться далеко в минулому.

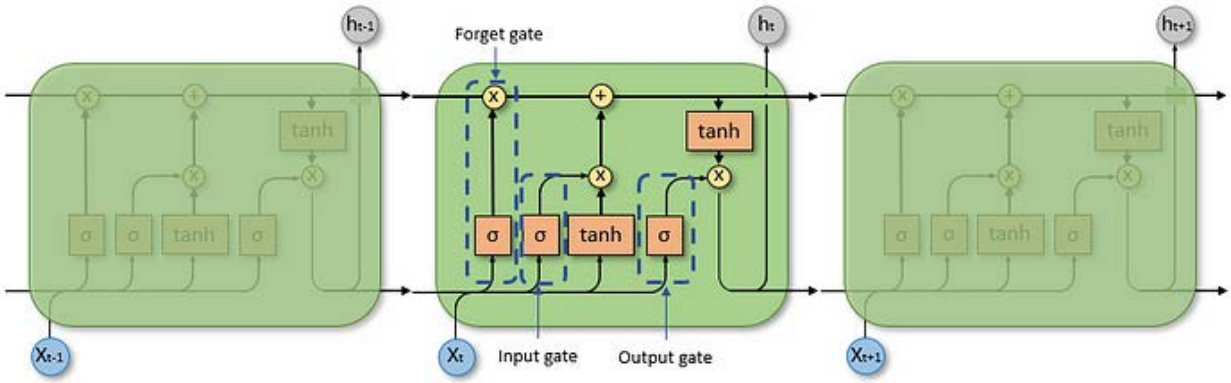


Рисунок 1.4 – Рекурентна нейронна мережа на основі LSTM шару [2]

Ключем до LSTMs є стан «клітини», який передається від входу до виходу іншої «клітини». Таким чином, стан клітини дозволяє інформації рухатися вздовж всього ланцюжка з мінімальними лінійними діями через три відділи. Таким чином, стан «клітини» представляє собою довгострокову пам'ять LSTM. Кожен відділ має свою назву: ворота забуття, ворота для входу та ворота для виходу. Кажен з цих відділів(воріт) діють як фільтри і контролюють потік інформації, визначаючи, яка інформація залишається в пам’яті або ж навпаки відкидається.

Ворота забуття визначає, скільки довгострокової пам'яті слід зберігати. Для цього використовується сигмоїдна функція активації, яка вказує на важливість стану «клітини». Вихід варіюється між 0 і 1 та показує, скільки інформації зберігається, якщо 0, то інформація не зберігається, якщо 1, то навпаки зберегти всю інформацію стану «клітини». Вихід визначається за допомогою поєднання

поточного входу x, прихованого стану h з попереднього кроку часу та зміщенням (bias) - b.

Формула 1.2 - перетворює інформацію у відділі забуття:

$$f_t = \sigma\left(W_{f,x}x_t + W_{i,h}h_{t-1} + b_f\right), \text{де} \tag{1.2}$$

$$\sigma = \frac{1}{1 + e^{-x}} \tag{1.3}$$

На рисунку 1.5 можемо побачити графік функції сигмоїди, як бачимо функція обмежена знизу нулем а зверху одиницею, саме тому вихід з усього відділу забуття варіюється між нулем та одиницею.

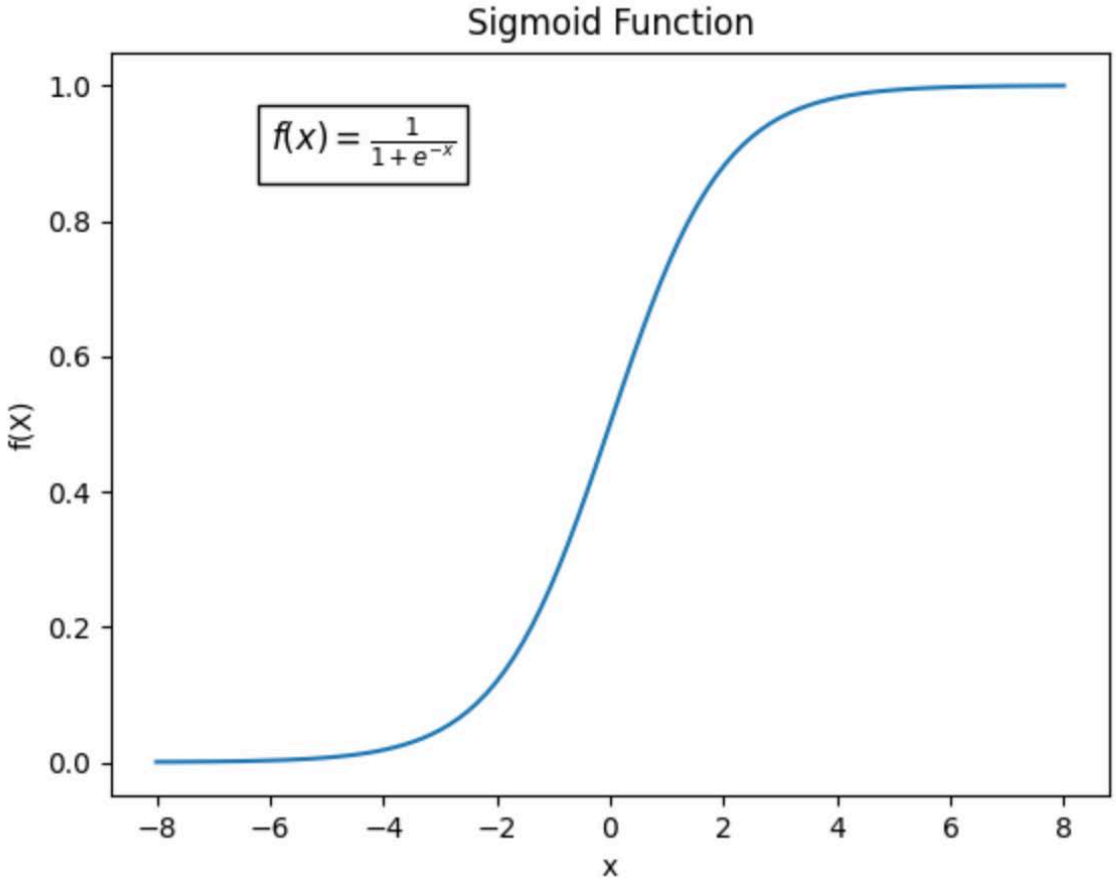


Рисунок 1.5 – Сигмоїдальна функція активації

Ворота для входу або ж відділ входу вирішує, яка інформація буде додана до стану «клітини» і, отже, також до довгострокової пам'яті. Тут шар сигмоїду визначає, які значення будуть оновлені.

$$i_t = \sigma\left(W_{i,x}x_t + W_{i,h}h_{t-1}\right) \tag{1.4}$$

Формула 1.4 – перетворює інформацію у відділі входу, використовуючи у формулі ту ж саму сигмоїдальну функцію представлену у формулі 1.3 або ж на рисунку 1.5.

Ворота для виходу або ж відділ виходу вирішує, яка інформація буде додана до короткострокової пам'яті. Формула 1.5 показує як обчислюється інформація у цьому відділі:

$$o_t = \sigma\left(W_{o,x}x_t + W_{o,h}h_{t-1} + b_0\right) \tag{1.5}$$

Як можна побачити всі три відділи представлені однією і тією ж самою функцією. Але ваги та байаси у всіх різні, стан «клітини» оновлюється за домогою вхідних та вихідних ворот за наступною формулою:

$$c_t = \left(f_t \circ c_{t-1} + i_t \circ \tanh(h_{t-1})\right) \tag{1.6}$$

Перший член у вищезазначеному рівнянні визначає, скільки з довгострокової пам'яті зберігається, тоді як другий член додає нову інформацію до коміркового стану.

Прихований стан поточного кроку часу потім визначається виходовим воротом та функцією тангенсу, яка обмежує стан «клітини» в межах від -1 до 1.

Переваги LSTM архітектури:

- LSTMs відзначаються здатністю ефективно захоплювати патерни як у довгостроковій, так і у короткостроковій перспективі послідовності. Це робить їх дуже ефективними для різноманітних завдань, особливо при роботі з залежностями протягом тривалого часу.
- Архітектура перевірена часом та великим попитом серед багатьох напрацювань у сфері обробки природньої мови

Недоліки LSTM архітектури:

- Складніша структура LSTMs призводить до вищої обчислювальної складності, роблячи їх більш витратними з точки зору обчислень. Ця складність призводить до подовжених часів навчання, що може бути практичним недоліком.
- Важке навчання нейронної мережі, особливо з «нуля», часто потерпає від зникаючих та вибухових градієнтів під час алгоритму зворотного розповсюдження помилки.

1.2 Нейронні мережі на базі трансформерів

Трансформерна нейронна мережа є новаторською архітектурою, яка спрямована на вирішення завдань відображення послідовностей, легко враховуючи довгострокові залежності. Наразі є передовою технікою в галузі обробки природної мови.

1.2.1 Механізм уваги

Для того щоб зрозуміти як працює нейронні мережі на основі трансформерів необхідно більш широко зрозуміти принцип роботи механізму уваги, для цього найкраще підійде приклад із реального життя.

Припустимо, що хтось надав нам книгу з машинного навчання та попросив надати інформацію щодо категоріальної крос-ентропії. Існують два шляхи досягнення цієї мети: перше, прочитати всю книгу і повернутися з відповіддю; друге, звернутися до змісту, знайти розділ "втрати", перейти до частини про крос-ентропію та ознайомитися із розділом про категоріальну крос-ентропію.

Наприклад, у першому методі може знадобитися тиждень на прочитання всієї книги. У другому випадку це займе всього 5 хвилин. Крім того, інформація, отримана від першого методу, може бути більш розпливчастою та різноманітною, оскільки вона базується на занадто багато інформації. З іншого боку, інформація, отримана від другого методу, буде точною та відповідатиме вимозі. – [7]

У першому випадку ми не сконцентрувалися на конкретній частині книги, тоді як у другому випадку ми зосередили свою увагу на розділі "втрати", а потім ще більше узагальнили увагу на частині, присвяченій крос-ентропії, де пояснюється концепція категоріальної крос-ентропії. Фактично, це той спосіб, яким більшість нас, людей, вчинили б подібно. Увага в нейронних мережах в певному сенсі подібна до того, що ми спостерігаємо у людей. Вони зосереджуються на високій роздільній здатності в певних частинах входів, тоді як інший вхід знаходиться у низькій роздільній здатності [8].

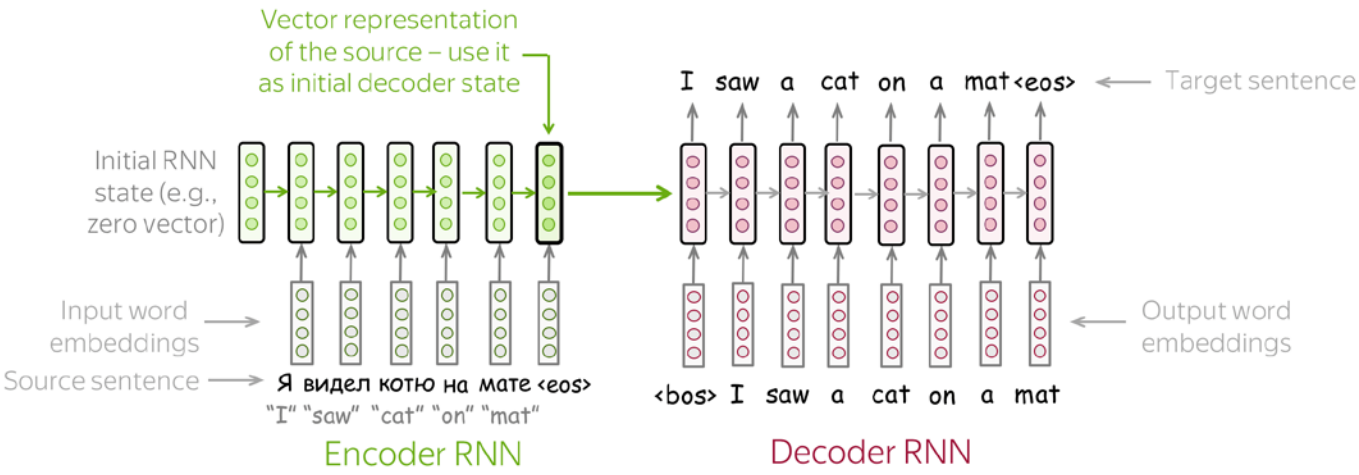


Рисунок 1.6 – Приклад моделі Seq-to-Seq [7]

На рисунку 1.6 ми бачимо, що прихований стан насправді є вектором контексту, який ми передаємо декодеру. Виявляється, що вектор контексту виявився проблематичним для цих типів моделей. Моделі мають проблему при роботі з довгими реченнями. Можна сказати, що вони стикалися з проблемою зниклих градієнтів при роботі з довгими реченнями. Таким чином, з'явилось рішення в роботі, введено «Механізм уваги». Це великою мірою покращило якість машинного перекладу, оскільки це дозволяє моделі фокусуватися на відповідній частині вхідної послідовності за необхідності.

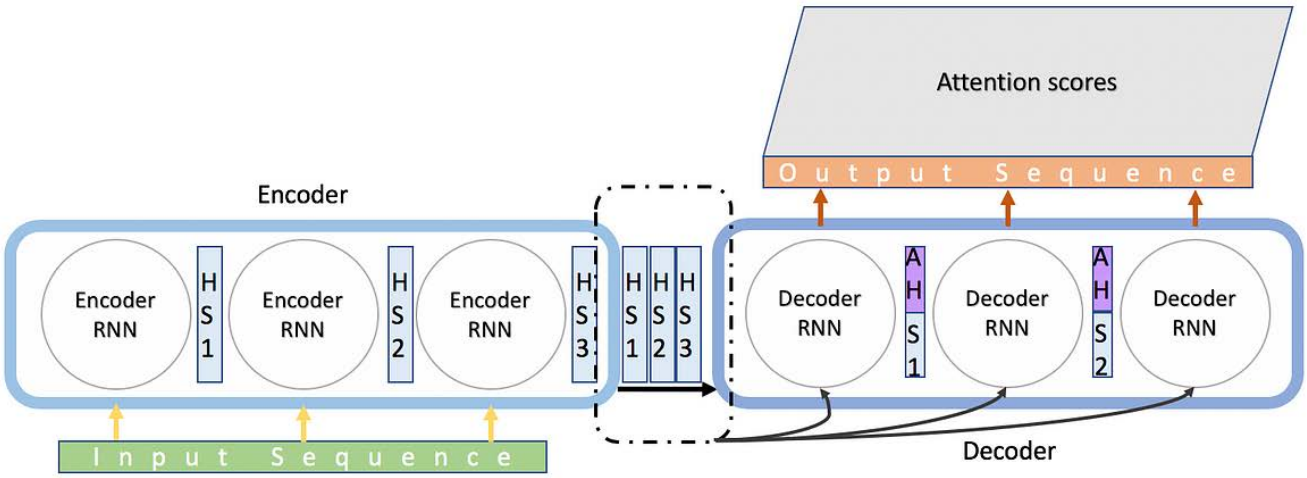


Рисунок 1.7 – Приклад моделі Seq-to-Seq, з використанням attention [7]

Ця модель уваги відрізняється від класичної моделі seq-to-seq двома способами, а саме: порівняно з простою моделлю seq-to-seq тут енкодер передає декодеру набагато більше даних. Раніше до декодера відправлявся лише останній, кінцевий прихований стан частини кодування, але тепер кодер передає всі приховані стани (навіть проміжні) декодеру. Частина декодера робить додатковий крок перед тим, як виробити вивід, а саме останній крок декодера виконується так:

1) він перевіряє кожний прихований стан, який він отримав, оскільки кожен прихований стан кодера в основному пов'язаний з певним словом вхідного речення.
2) Він потім надає кожному прихованому стану оцінку. Потім кожну оцінку множиться на відповідний softmax-бал, тим самим підсилюючи приховані стани з високими балами та заглушуючи приховані стани з низькими балами.

   Візуалізація цього процесу показано на рис. 1.8

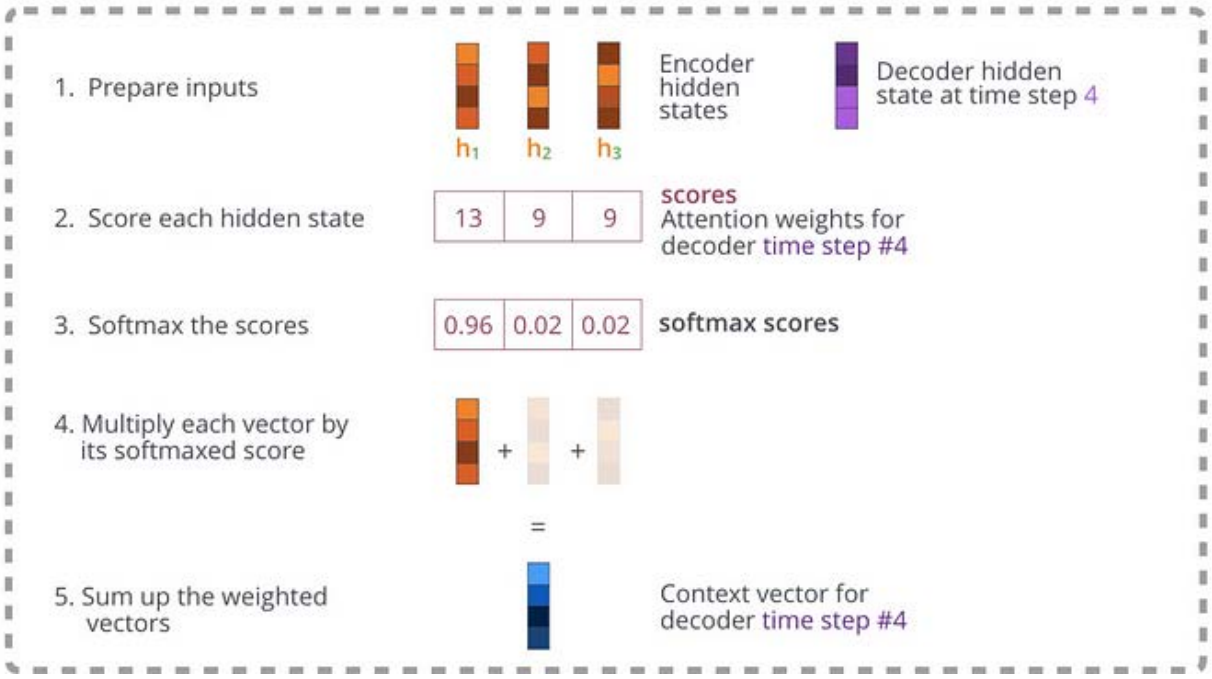


Рисунок 1.8 – Алогоритм роботи механізму уваги [7]

На рисунку 1.8 можна побачити у 3-ому кроці операцію softmax – це означає що усі тензори пропускаються через формулу 1.7

$$s(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{1.7}$$

Де факто ця функція перетворює усі вхідні значення у проміжок [0; 1], і нормує. Тепер, коли ми об'єднуємо усе разом:

Шар декодера уваги приймає вектор вбудовування токена та початковий прихований стан декодера. RNN обробляє його введення та генерує вивід і новий вектор прихованого стану (h4). Тепер ми використовуємо приховані стани кодера та вектор h4 для обчислення вектора контексту C4 для цього кроку часу. Саме тут застосовується концепція уваги, тому це називається кроком уваги. Ми конкатенуємо (h4) та C4 в один вектор. Тепер цей вектор передається у нейронну мережу прямого поширення. Вивід нейронної мережі прямого поширення вказує слово виводу на цьому кроці часу. Ці кроки повторюються для наступних кроків часу. – [7]

1.2.2 Архітектура “трансформеру”

Вперше поняття «трансформер» було запроваджено в роботі «Attention Is All You Need», опублікованій у 2017 році, де факто це архітектура енкодер-декодер на основі шарів уваги(механізму уваги), детальний опис можна побачити у наступних сторінках роботи,

У трансформерах вхідну послідовність можна передавати паралельно, щоб ефективно використовувати GPU, і також збільшити швидкість навчання. Це базується на шарах “багатоголовкової” уваги, що досить вагомо допомагає подолати проблему зниклих градієнтів. У цій статті робота ґрунтується на застосуванні трансформера до задачі машинного перекладу, основним концептом стало те що у трансоформері ми можемо передавати всі слова речення паралельно

і визначати вектор слова одночасно. На рис. 1.9 представлена архітектура трансформера

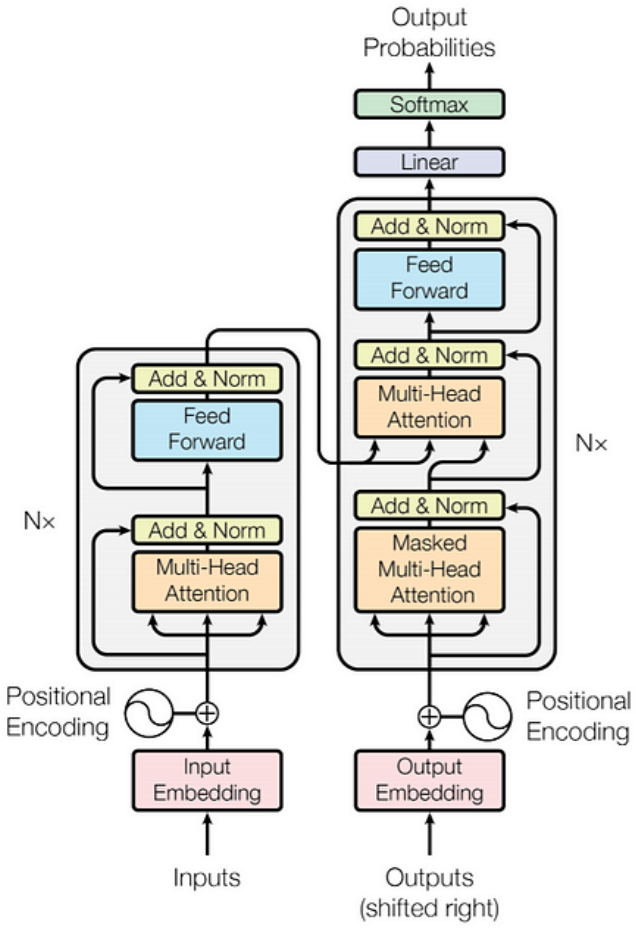


Figure 1: The Transformer - model architecture.

Рисунок 1.9 – Архітектура трансоформера [6]

Цей трансформер блок можна представити як окрему нейронну мережі і для того, щоб в ньому розібратися почнемо його їсти поступово, а саме з першої частини – «encoder block». На рис. 1.10 показано що собою являє «encoder block».

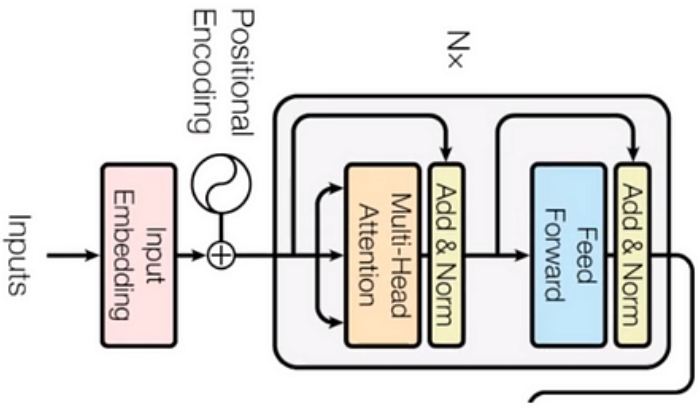


Рисунок 1.10 – Encoder block [6]

Як бачимо з рисунку вище, спочатку ми трансформуємо вхідне речення у вектор, бо нейроні мережі не вміють працювати з текстом, це відбувається першим шаром – «Input embeding», далі постає проблема що кожне слово в різних реченнях може мати різні значення. Щоб вирішити це питання, ми звертаємося до «Positional Encoders» що є другим шаром в нашій кодувальній частині. Де факто це є вектори, які надають контекст в залежності від позиції слова у реченні. Процес самого кодування речення у вектор буде описаний у наступній частині.

Далі ми переходимо до шару «багатоголовкової» уваги, що представлений у помаранчевому кольорі на рис. 1.10. Він акцентує увагу на тому, наскільки конкретне слово є значущим відносно інших слів у даному реченні. Його представлено у вигляді вектора уваги. Для кожного слова може бути згенерований вектор уваги, який відображає контекстуальні відносини між словами в цьому реченні. – [7]

Єдина проблема, з якою він стикається, полягає в тому, що кожне слово має набагато більшу цінність у реченні, якщо ми схильні до його взаємодії з іншими словами цього речення. Отже, ми визначаємо кілька векторів уваги на слово та беремо зважене середнє, щоб обчислити кінцевий вектор уваги кожного слова.

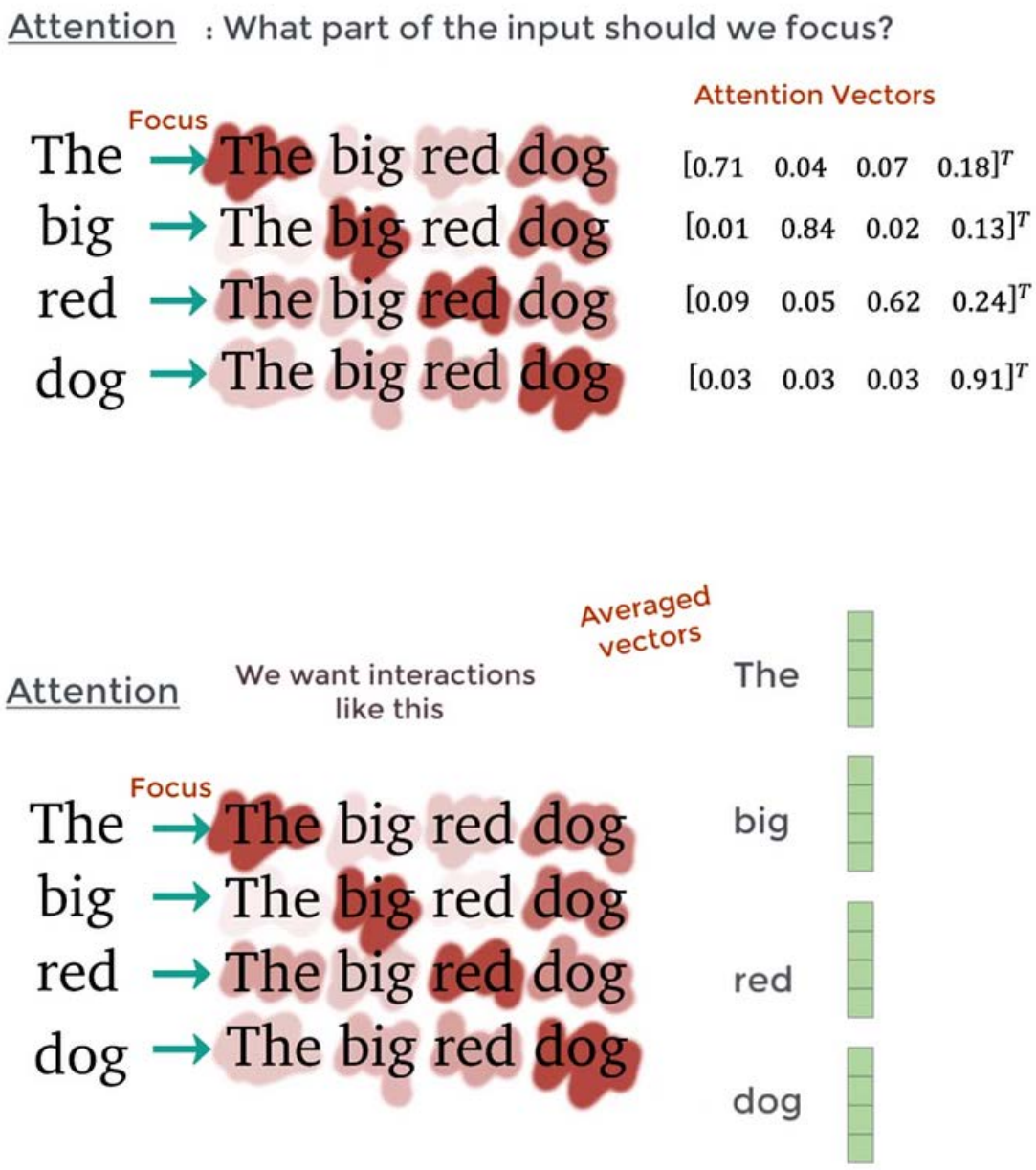


Рисунок 1.10 – 1.11 – Приклад роботи механізму багатоголовкової уваги [7]

Наступним іде блок звичайної нейронної мережі прямого поширення яка застосовується до кожного вектора уваги; її основна мета – перетворити вектори уваги в форму, прийнятну для наступного шару кодера чи декодера. Мережа прямого поширення приймає вектори уваги «по одному». І найбільшим проривом у галузі являється те що тут можна застосувати паралелізацію, що пришвидшує час навчання суттєво. А це означає, що ми можемо передати все речення загалом(всі слова в реченні) одночасно у наш encoder block.

Взагалі ми можемо високорівнево ототожнити encoder block як представлення чисельних векторів слів з відповідними вагами у n-вимірному просторі, з якого потім вже безпосередньо декодер буде зчитувати і перетворювати n-вимірний результат на тензорне представлення векторів слів. Подивимось що собою являє декодер докладніше.

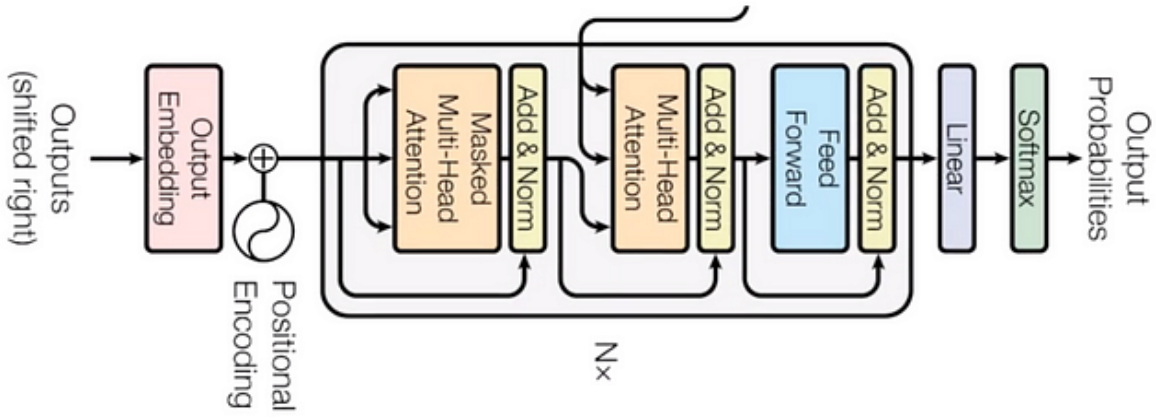


Рисунок 1.12 – Декодер блок [6]

З початку як можна побачити з рис. 1.12 іде такий самий блок кодування, що перетворює вихідну послідовність у тензорне представлення, так само як і в енкодінг частині цієї архітектури. Далі іде блок маскованої багатоголовкової уваги, процес включає в себе ступінчате створення вагових коефіцієнтів для кожного слова з урахуванням його взаємозв’язку з іншими словами у реченні. Маскування використовується для обмеження доступу до інформації, яка ще не повинна бути врахована під час генерації уваги.

Застосування багатоголовковості дозволяє паралельно враховувати різні аспекти контексту, а вагова увага визначає значущість кожного слова у контексті його відносин з іншими. Отримані ваговані вектори уваги об’єднуються, утворюючи кінцевий вектор уваги для кожного слова у реченні.

Цей підхід дозволяє моделі ефективно враховувати контекстуальні відносини між словами під час генерації перекладу, покращуючи точність та розмаїття уваги.

На рис. 1.13 можна побачити більш детальне пояснення цього процессу.

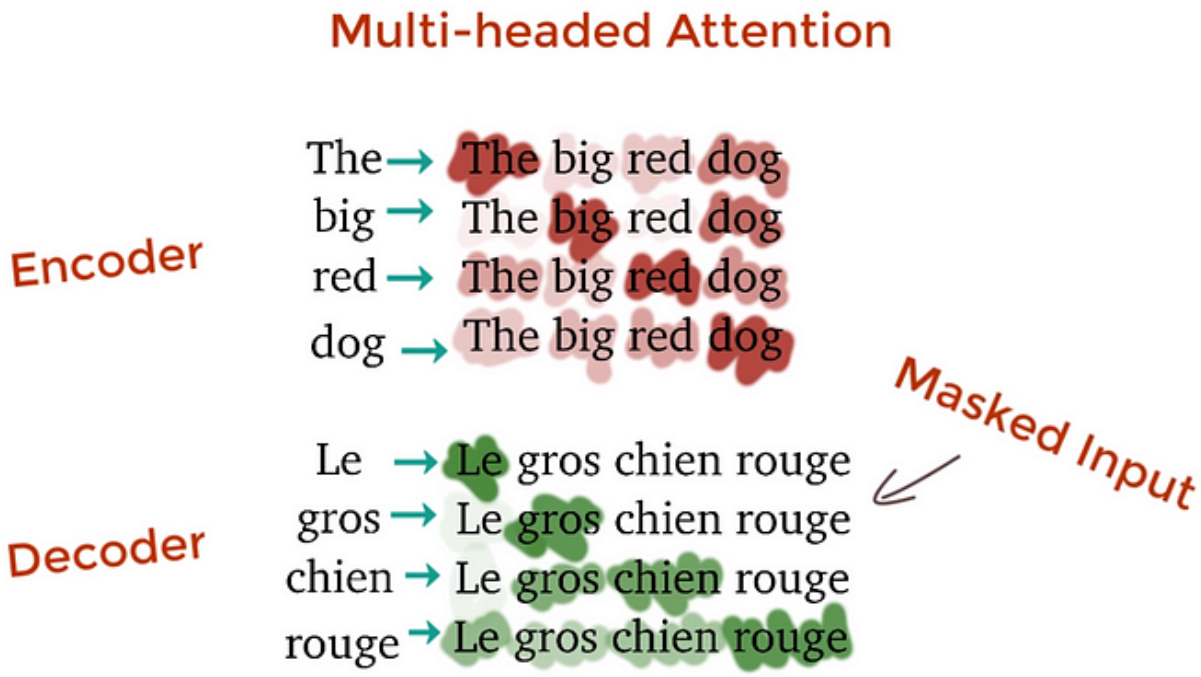


Рисунок 1.13 – Приклад багатоголовкової уваги з маскуванням [6]

Отримані вектори уваги з попереднього шару та вектори з блоку кодера подаються в інший блок багатоголовкової уваги. Цей етап критичний, оскільки включає в себе інтеграцію результатів з блоку кодера в поточний шар. На діаграмі відзначено, що результати з блоків кодера подаються сюди. Саме це обумовлює назву цього блоку – «Encoder-Decoder Attention Block». Тепер ми подаємо кожен вектор уваги в блок feed-forward, який перетворює вихідні вектори в щось, що легко прийнятно іншим блоком декодера або лінійним шаром. Лінійний шар – це ще один блок feed-forward. Він використовується для розширення розмірності до кількості слів у французькій мові після перекладу. Тепер це проходить через шар Softmax докладніше у формулі 1.7, який перетворює вхід в ймовірнісний розподіл, який може бути інтерпретований людиною. Результуюче слово виробляється з найвищою ймовірністю після перекладу.

1.3 Порівняльна характеристика існуючих рішень

Для порівняння цих архітектур було взято датасет IMDb – [8], містить інформацію про фільми, їх жанри та оцінки, які користувачі надають цим фільмам. Кожен рядок у таблиці відповідає конкретній оцінці, включаючи ідентифікатор фільму, його назву, жанри, ідентифікатор користувача, оцінку та мітку часу, має табличний формат даних, далі представимо його короткий опис у вигляді таблиці:

Таблиця 1.1 – Таблиця опису данних у IMDb dataset:

Атрибут	Тип данних	Опис
movieId	int	Унікальний ідентифікатор фільму
title	str	Назва фільму
genres	str	Жанри фільму (розділені комами)
userId	int	Унікальний ідентифікатор користувача, який залишив оцінку
Rating	float	Оцінка фільму, надана користувачем
timestamp	datetime	Часова мітка оцінки (в секундах або іншому форматі)

Далі ми продемонструємо експерементальні результати досліджень:

Таблиця 1.2 - Таблиця порівняння архітектур

Methods	Accuracy	Precision	Recall	F1 score
RNN	0.86	0.85	0.75	0.7969
LSTM	0.87	0.83	0.88	0.854
Transfoemer	0.93	0.904	0.91	0.907

Для того щоб краще зрозуміти результати таблиці 1.2, введемо метрики які були використані у цій таблиці.

Щоб було простіше описувати метрики наведені нижче введемо поняття матриці невідповідностей.

Матриця невідповідностей (confusion matrix) є інструментом в оцінці результатів моделі класифікації в галузі машинного навчання. Ця матриця відображає кількість правильних та неправильних класифікацій для кожного класу в задачі класифікації. У науковому стилі, матриця невідповідностей слугує інструментом для визначення ефективності моделі та виявлення схильностей чи помилок в її прогнозах. – [10]

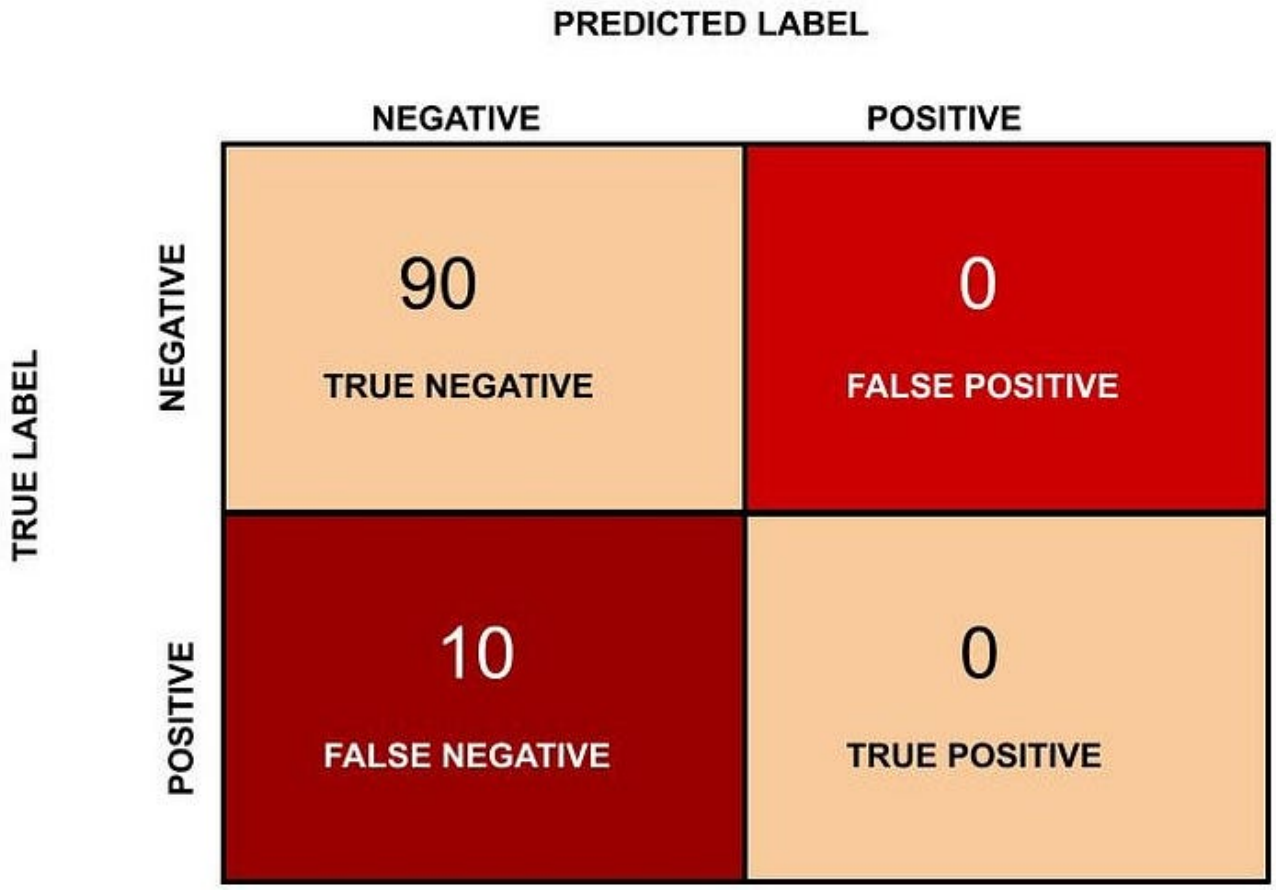


Рисунок 1.14 – Приклад матриці невідповідності [11]

- True Positive (TP): Кількість спостережень, які належать певному класу і були правильно класифіковані як цей клас.
- False Negative (FN): Кількість спостережень, які належать певному класу, але були помилково класифіковані як інший клас.
- False Positive (FP): Кількість спостережень, які не належать певному класу, але були помилково класифіковані як цей клас.
- True Negative (TN): Кількість спостережень, які не належать певному класу і були правильно визначені як інший клас. – [10]

Далі введемо поняття точності (accuracy) - метрика оцінки ефективності моделі машинного навчання, яка вимірює відсоток правильних прогнозів моделі серед усіх прогнозів, які вона здійснила. Формально визначається як відношення

кількості правильних прогнозів до загальної кількості спостережень у наборі даних. – [10]

Формула для обчислення :

$$Accuracy = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}} \quad (1.8)$$

Ця метрика часто використовується для оцінки загальної ефективності моделі, особливо у випадках, коли класи в задачі класифікації мають приблизно однакову важливість. Однак у випадках з незбалансованими класами варто також звернути увагу на інші метрики, такі як precision, recall або F1-score.

Наступним введемо поняття precision - метрика в галузі оцінки ефективності моделей класифікації Вона вимірює, як часто модель правильно ідентифікує позитивні екземпляри серед усіх ідентифікованих позитивних прогнозів. Точність вказує на те, наскільки модель уникає неправильних позитивних класифікацій. – [10]

Розраховується за наступною формулою:

$$Precision = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad (1.9)$$

У формулі скорочення TP – відповідає дійсно позитивним значенням, FP – неправильно позитивне значенняВисока точність свідчить про те, що модель майже виключно робить позитивні прогнози лише для тих екземплярів, які насправді є позитивними.

Наступним введемо поняття чутливості або повноти (recall) – це метрика в області оцінки ефективності моделей класифікації, вона вимірює, як часто модель

правильно ідентифікує всі дійсні позитивні екземпляри серед усіх наявних дійсних позитивних екземплярів. Ця метрика ставить свою увагу на здатність моделі виявляти всі існуючі позитивні класи. – [10]

$$Recall = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{1.10}$$

Ця метрика може бути корисною в ситуаціях, коли важливо максимізувати виявлення позитивних прикладів, навіть за ціною деяких помилок (наприклад, у випадку виявлення хвороби, де пропуск одного позитивного випадку може мати серйозні наслідки).

Далі введемо поняття останньої метрики f1-score - це метрика в галузі оцінки ефективності моделей класифікації, яка представляє собою гармонічний середній між точністю (precision) і чутливістю (recall). F1-score дозволяє оцінити баланс між точністю та повнотою моделі. – [10]

$$F1_{score} = \frac{2 * precision * recall}{recall + precision} \tag{1.11}$$

F1-score – вважається заслуженно однією з найкращих метрик для оцінки якості моделей класифікації, оскільки вона є універсальною для сбалансованих та несбалансованих даних, очевидно що різні задачі потребують іноді більш комплексних підходів для оцінки моделей, але в нашому випадку цих метрик достатньо для розуміння дійсності

1.4 Висновки до розділу

В цьому розділі були розглянуті новітні підходи для розв’язання задачі семантичного аналізу тексту, а саме нейронні мережі на базі рекурентних нейронних зв’язків, на основі lstm блоків та трансформер блокі, результати точності приведені в таблиці 1.2, подальше дослідження буде сконцетровано навколо нейронних мереж на основі трансформер блоків, оскільки вони на практиці показали найкращі результати за метрикою f1-score.

2 ПРОЕКТУВАННЯ МАТЕМАТИЧНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Опис даних

Для роботи з даними було обрано заклад «Starbacks» і безпосередньо на відгуках цього закладу буде проводитись тренування та валідація математичної моделі. Для початку подивимось початковий «сирий» вигляд данних, який представлений у вигляді таблиці 2.1

Таблиця 2.1 Опис даних

Атрибут	Тип даних	Опис
Name	str	Ім'я рецензента, якщо присутнє
state	str	Локація(місто) яке відноситься до відгуку, якщо присутнє
Date	datetime	Дата коли відгук був створений
Rating	int	Оцінка, варіюється від 1 до 5
Review	str	Відгук на заклад

Як бачимо основними для нас являються поля date, location, rating, review, які будуть використовуватись в майбутньому, поле name, в подальшому використовуватись не буде. Далі поглянемо на данні трохи детальніше оцінивши, з якою вибіркою ми працюємо.

Для початку оцінемо кількість пропущених значень:

```
df.isna().sum()
```

```
name        0
Rating    145
Review      0
state       0
city        0
year        0
month       0
dtype: int64
```

Рисунок 2.1 – Кількість пропущених даних

Як видно з рис. 2.1 бачимо 145 рядків коментарів з пропущеною оцінкою, що для нас доволі критично оскільки оцінка потрібна для кінечної перевірки як працює математична модель, а також для навчання, оскільки наша задача класифікації відноситься до задачі навчання математичної моделі з учителем. Тому доведеться викинути ці спотворені рядки, адже ні один з методів заповнення пропущених даних тут не стане у нагоді, ні заповнення медіанним значенням ні середнім.

Далі поглянемо на розподіл даних відносно оцінки, це покаже наскільки збалансований наш датасет. На рис. 2.2 можна побачити, що є змішення у сторону негативних відгуків з оцінкою в 1, інші оцінки мають приблизно однакове розподілення, хоча відгуків з оцінкою в 3 досить мала кількість. Також можна зробити висновок, що датасет не збалансований і це потрібно брати до уваги при тренуванні та під час валідації математичної моделі. Для цього доповнимо датасет даними які в меншості, щоб він став більш збалбансований і не було перекосу під час тренування, також у якості метрики оцінки моделей будемо використовувати f1-score, див. розділ 1.3, формулу 1.11

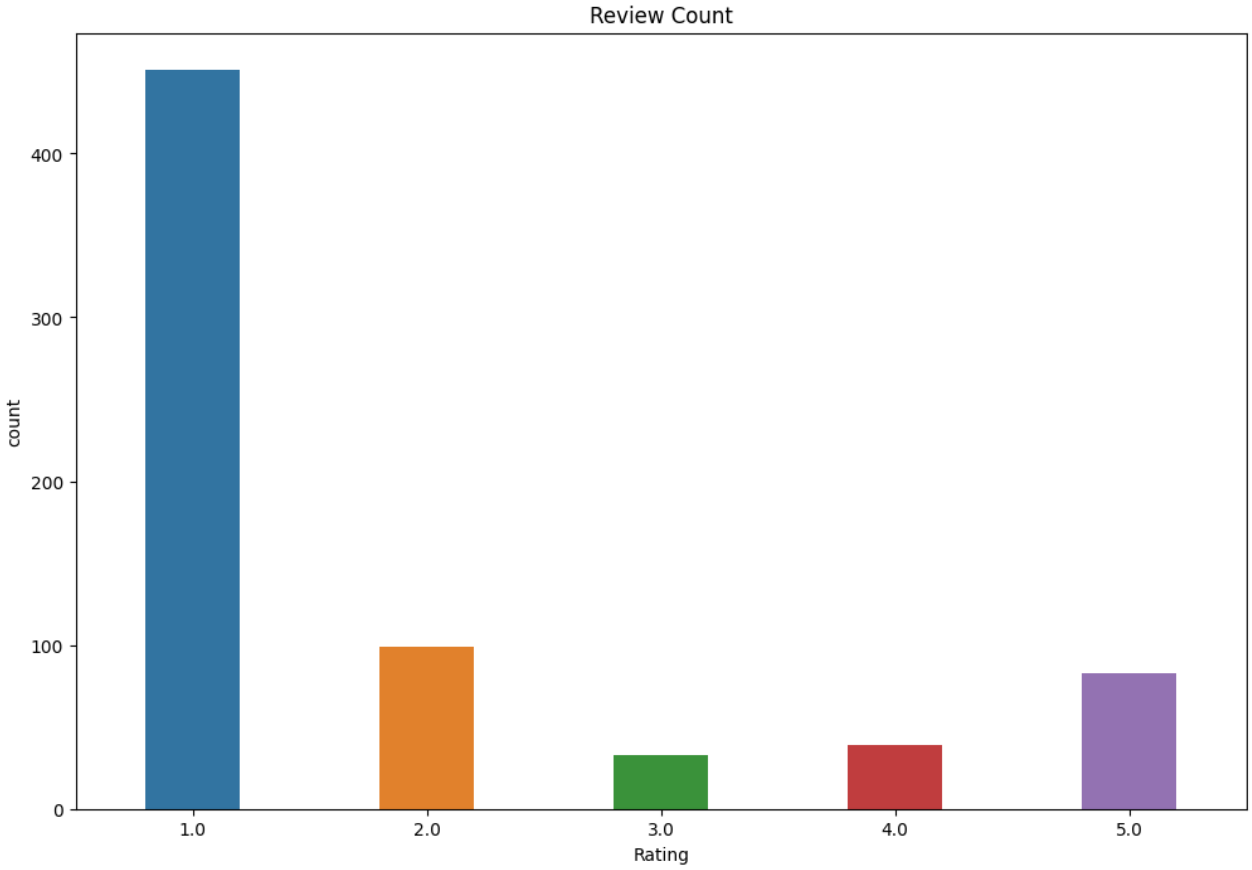


Рисунок 2.2 – Розподіл оцінок у датасеті

Також подивимось на розподіл оцінок у розрізі локації в якій був клієнт. Рис. 2.3 демонструє нам такий розподіл.

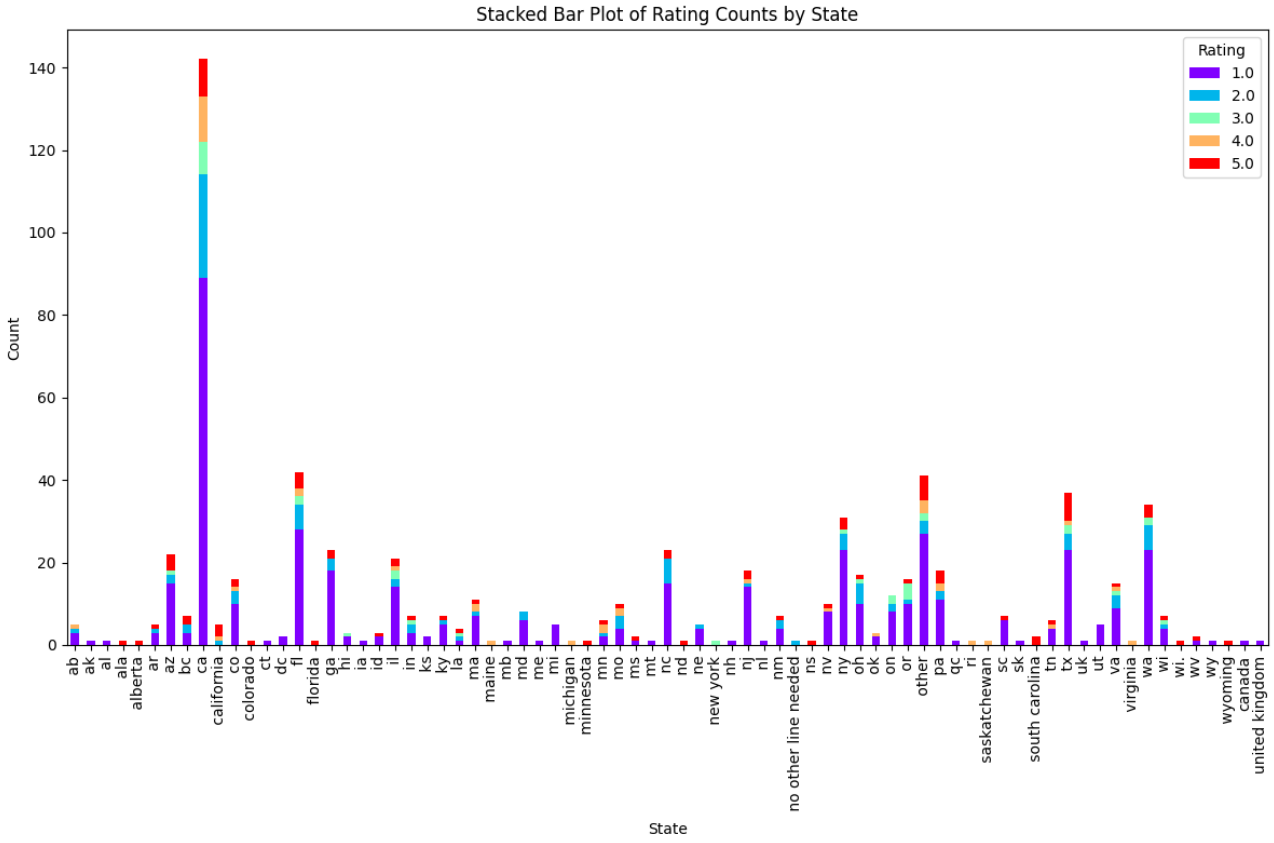


Рисунок 2.3 – Розподіл оцінок за штатом

З рисунку 2.3 можна побачити, що найбільше даних в нас прийшло зі штату Каліфорнія, що є одним з найбагатших штатів саме тому має найбільшу кількість відгуків і скоріше за все і замовлень. Також доволі багато відгуків маємо із Нью-Йорку та Флориди, що є багатими містами, де кількість замовлень у «Старбакс» може сягати сотні тисяч у день, що говорить нам про те, що чим більше замовлень було зроблено у закладі тим більше він відгуків буде мати, навіть якщо тільки 5% відсотків від усіх клієнтів залишають відгук, то при масштабуванні картина очевидна.

Також очевидно зрозуміло чому негативних відгуків більше за позитивних, бо якщо людям щось сподобалось вони менше бажають витрачати час на коменування, що саме їм сподобалось, і навпаки якщо щось не так, то бажання зробити закладу якнайгірше з чого випливає більша кількість негативних коментарів.

Далі поглянемо на розподіл відгуків за роками та взагалі з наскільки актуальними даними ми працюємо.

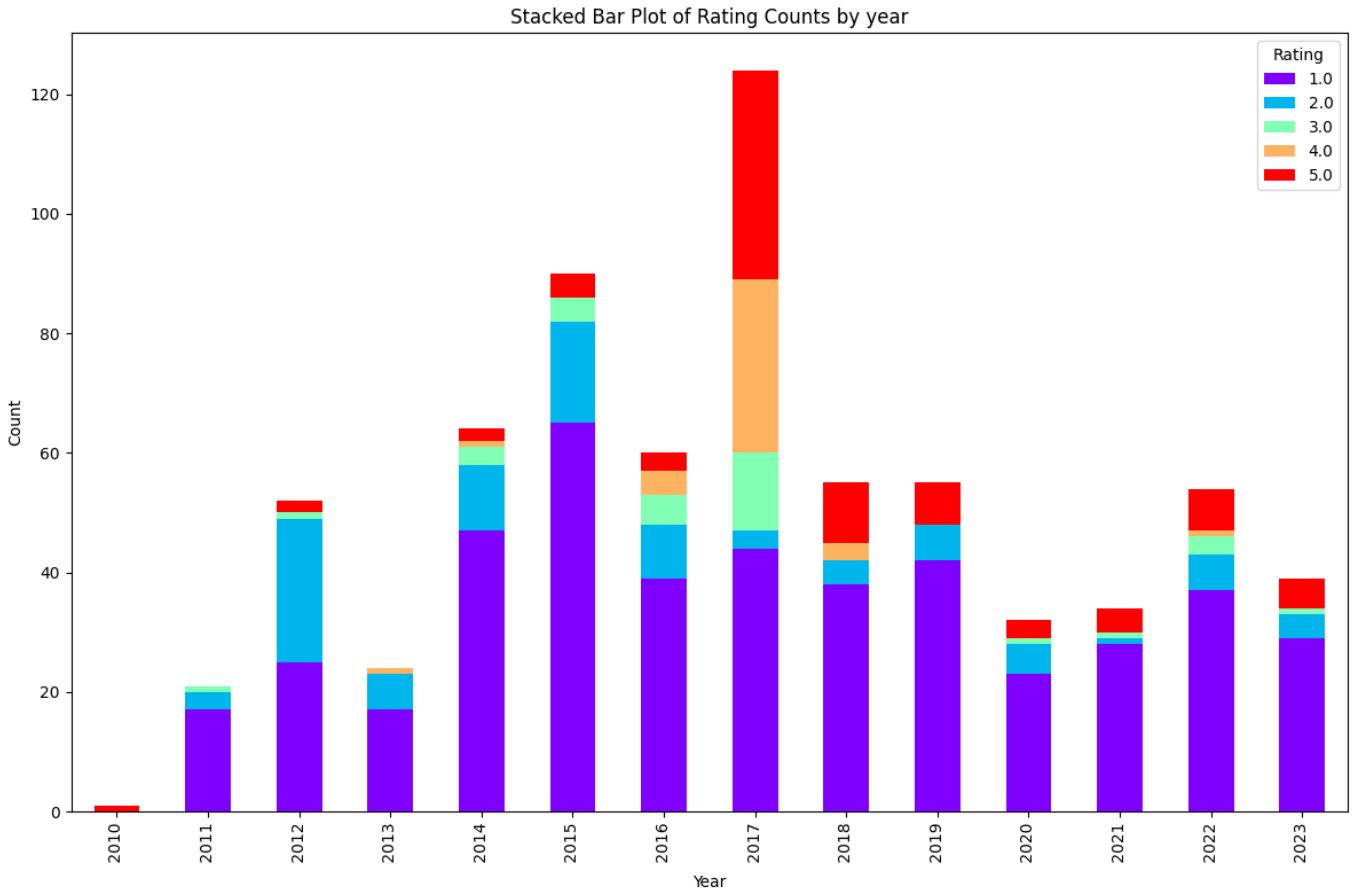


Рисунок 2.4 – Розподіл оцінок за роками

Як бачимо з рис. 2.4 маємо досить великий розрив між мінімальною датою відгуку та максимальною, 12 років, але для нашого випадку це не проблема, оскільки колонка дати буде використовуватись лише для поділу даних на тренувальну, валідаційну та тестову вибірку. До якої повернемось трохи згодом, також бачимо, що майже вся популяція відгуків з оцінкою 4 припадає на 2017 рік, що не дуже добре і нам треба буде доповнити данні інших років дублями із 2017.

Ця необхідність потребується для нашого методу розділення даних на тренувальні та валідаційні.

2.2.1 Поділ данних

Для навчання моделей машинного та глибинного поняття часто вводиться поняття тренувальної, валідаційної та тестувальної вибірки. Кожна з яких має свій практичний зміст на різних єтапах життєвого циклу моделей машинного навчання. Введемо формальні поняття кожної з них.

Тренувальна вибірка – є підмножиною вихідного датасету, яка використовується для навчання моделі. Ці дані використовуються для визначення параметрів моделі та оптимізації її ваг, щоб максимально точно адаптуватися до характеристик вихідних даних. Тренувальна вибірка є основою для формування моделі та визначення її здатності до узагальнення на нових даних.

Валідаційна вибірка є окремою частиною датасету, яка не використовується під час тренування моделі, але використовується для налаштування гіперпараметрів моделі та оцінки її загальної ефективності. Вона дозволяє уникнути перенавчання, а також визначити оптимальні параметри для досягнення кращої узагальненої продуктивності моделі.

Тестова вибірка є окремим набором даних, який залишається поза впливом як тренувальної, так і валідаційної вибірок під час всього процесу навчання. Ці дані використовуються для фінальної оцінки продуктивності моделі після завершення тренування та налаштування. Тестова вибірка надає незалежну перевірку того, наскільки добре модель може узагальнювати свої знання на нових даних, яких модель ще не бачила.

Розбиття датасету на тренувальну, валідаційну та тестову вибірки важливе для уникнення перенавчання, об'єктивної оцінки ефективності моделі та визначення оптимальних параметрів. Такий підхід забезпечує, що модель ефективно навчається, узагальнює на нові дані та готова до використання в реальних умовах.

Проблема звичайного підходу до поділу даних по індексу в тому що цей метод недоцільний у багатьох реальних ситуаціях. Причина полягає в тому, що рядки рідко бувають незалежними. Наприклад, якщо ви тренуєте модель для передбачення затримок у рейсах, затримки прибуття рейсів у той самий день будуть високо корелювати одне з одним. Це називається «утіканням інформації» (leakage), і це важлива проблема, яку слід уникати при виконанні завдань машинного навчання.

У нашому випадку ми використаємо більш цікавий підхід до поділу даних. Спочкатку для цього нам потрібно обрати ознаку в датасеті яка має в собі властивість, що рядки в ній між собою мають кореляцію, у нашому випадку це колонка з датою, після цього використаємо хеш функцію яка перетворить нашу колонку дат у 64-ох бітний числовий формат, для цього буде використано FARM_FINGERPRINT алгоритм хешування.

Даний алгоритм із сім'ї FarmHash алгоритмів, розроблених Google, його переваги у тому що він детермінований та має добрий розподіл значень, а також імплементований багатьма мовами програмування, що дає велику гнучкість у розробці.

Після виконання хешування обраного атрибуту, порахуємо залишок від ділення отриманого 64-ох бітного цілого числа і візьмемо значення строго менше 8 до тренувальної вибірки, щоб сформувати валідаційний набір візьмемо значення залишку від ділення, яке дорівнює 8, для тестувального набору значення буде дорівнювати 9.

Перевагою такого підходу є те що він легко відтворює ті самі тренувальні, валідаційні та тестові датасети при різних тренуваннях та експериментах.

Чому ми обрали саме дату для розмежування даного датасету – бо вона повинна відповідати деяким критеріям, а саме:

- Рядки, що відповідають одній та тій же даті, повинні бути взаємозалежними — це, знову ж таки, ключова причина, чому ми хочемо переконатися, що всі рядки на одній та тій же даті належать до одного й того ж розділу.

- Дата не є вхідними атрибутом для нашої математичної моделі (ознаки, витягнуті з дати, такі як день тижня чи година доби, можуть бути введеннями, але ми не можемо використовувати фактичний вхід, оскільки навчена модель не побачить 20% можливих значень входу).
- Має бути достатньо значень дат. Оскільки ми обчислюємо хеш та визначаємо залишок відносно 10, нам потрібно мати щонайменше 10 унікальних значень хешу. Звісно, чим більше унікальних значень у нас є, тим краще.
- Мітки (labels) повинні бути рівномірно розподілені серед дати. Якщо виявиться, що всі оцінки в 4 відбулися в одну дату, і за рештою року таких оцінок не було, це не працюватиме, оскільки розділені набори даних будуть перекошені.

Саме для останнього пункту і ми доповнювали наш датасет штучними даними, щоб не було перекосу у наших новоутворених тренувальних, валідаційних та тестувальних вибірках.

2.2 Передобробка даних

Ефективна передобробка тексту дозволяє очистити дані від зайвої інформації, провести лематизацію та токенізацію для виокремлення значущих елементів, а також вирішити проблеми із збалансуванням та обробкою різноманітності мовленнєвих особливостей.

На цьому етапі ми виконаємо фундаментальні дії з очищення тексту. Ці дії включають перетворення всього тексту до нижнього регістру, вилучення символів, які не відповідають словам чи пропускам, а також видалення всіх числових цифр, які присутні у тексті. – [15]

Лістинг відповідного фрагменту програмного коду приведений нижче:

```
df = df.applymap(lambda x: x.lower() if isinstance(x, str) else x)
```

Результат такої обробки зображено на рис. 2.5

Рисунок 2.5 – Результат приведення тексту до нижнього регістру

Наступним етапом іде очистка тексту від URL, оскільки дана інформація не несе в собі вагомості, наступним кроком необхідно вилучити будь-які символи, які не вважаються словами чи пропусками з текстового набору даних.

Ці символи, які не є словами чи пропусками, можуть включати розділові знаки, символи та інші спеціальні символи, які не надають жодної значущої інформації для нашого аналізу. – [15]

Лістинг відповідного фрагменту програмного коду приведений нижче:

```
df = df.replace(to_replace=r'[^\w\s]', value='', regex=True)
```

Результат такого перетворення продемонстровано на рис 2.5:

Рисунок 2.6 – Результат очистки тексту від зайвих символів

Наступним кроком важливо вилучити всі числові цифри з текстового набору даних. Це тому, що у більшості випадків числові значення не надають значущого змісту для процесу аналізу тексту. Крім того, вони можуть втручатися у алгоритми

обробки природної мови, які спроектовані для розуміння та обробки текстової інформації. – [16]

2.2.1 Очищення зайвих слів

З точки зору аналізу текстових даних та побудови моделей обробки природної мови варто проводити очистку тектсу від так званих стоп-слів, це найбільш поширені слова в будь-якій природній мові, стоп-слова можуть не додавати багато цінності до значення документа. Тому видалення стоп-слів може допомогти нам зосередитися на найважливішій інформації в тексті та покращити точність нашого аналізу.

Однією з переваг видалення стоп-слів є зменшення розміру набору даних, що в свою чергу зменшує час навчання моделей обробки природної мови. Приклад видалення стоп-слів продемонстровано на рис. 2.9

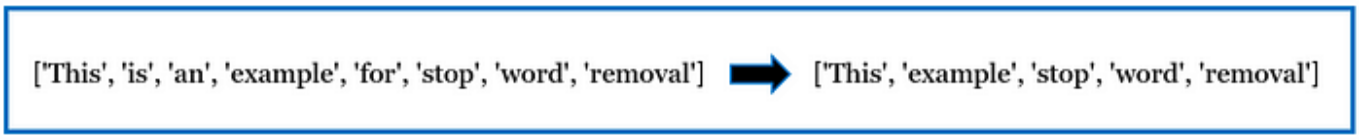


Рисунок 2.9 – Приклад видалення стоп-слів [17]

Далі введемо поняття токенізації слів, на простому прикладі, а потім масштабуємо на наші дані

2.2.2 Токенізація слів

Введемо означення терміну токенізація. Токенізація - це процес розбиття великих блоків тексту, таких як абзаци і речення, на менші, більш керовані одиниці, а потім перетворення у векторні представлення, для того щоб мовні моделі, зазвичай трансформери, могли зрозуміти значення та семантику різних слів у реченні або ж тексті. Наведемо приклад схожої семантики слів в тексті, вектори для слів «King» і «Man» можуть бути схожими, так само як вектори для «Queen» і «Woman». Ці вектори також мають певні властивості, які можуть бути корисними для навчання мовних моделей. Наприклад, ви можете відняти вектор для «Man» від вектора для «King» і додати вектор для «Woman», щоб отримати вектор для «Queen». Ці властивості дозволяють моделі розуміти різні значення слів. – [21].

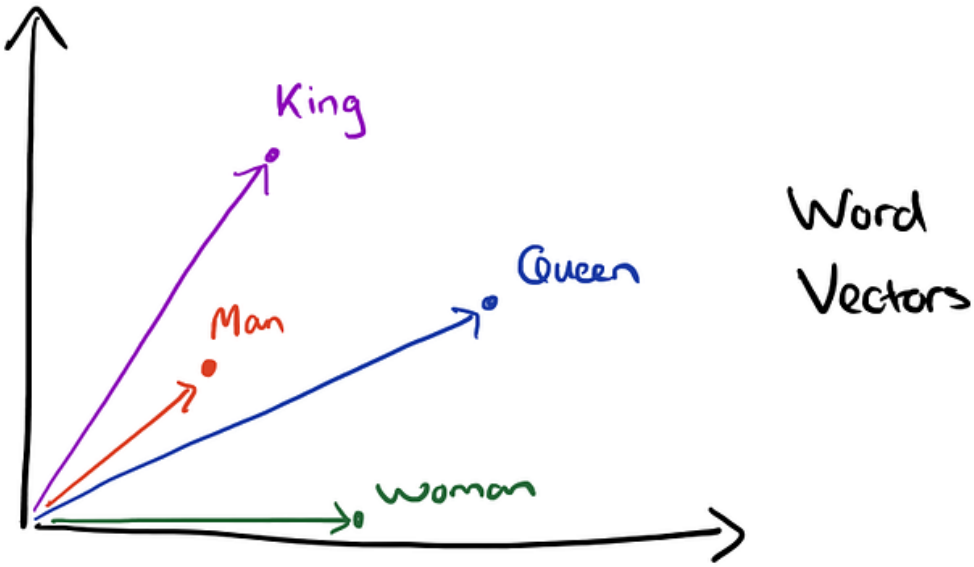


Рисунок 2.7 – Приклад векторів слів [21]

Рисунок 2.8 – Приклад композиції векторів [21]

Вектор слова можна уявити як точку у багатовимірному просторі, де кожна вимірювана вісь представляє конкретний аспект чи характеристику слова. Наприклад, вектор слова для "queen" може мати високі значення для вимірювань, які представляють "жіночність" та "королівство", і низькі значення для вимірювань, які представляють "чоловічість". Комбінуючи всі ці виміри, утворюється вектор для "queen", що дозволяє нашій моделі розуміти значення слова та його взаємозв'язок з іншими словами. Кількість вимірів у векторі слова часто називається "розмірність". Високорозмірний вектор слова матиме багато вимірів, що дозволяє захопити широкий спектр характеристик і нюансів слова.

Однак це також означає, що для навчання та використання йому знадобиться більше даних і обчислень. З іншого боку, низькорозмірний вектор слова матиме менше вимірів, що робить його простішим та ефективнішим для роботи, але можливо, втратжуючи деяку багатозначність та деталі значення слова. – [21]

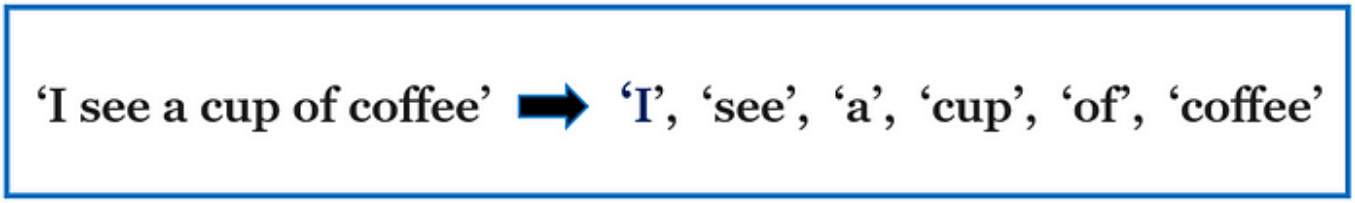


Рисунок 2.9 – Приклад токенізації речення по словам [15]

Методи токенізації варіюються залежно від розгортання тексту та конкретних вимог завдання. Ці методи можуть включати розбиття тексту на окремі слова, на окремі символи або навіть на менші одиниці. Давайте докладніше розглянемо різні типи:

- Токенізація за словами: Цей метод розбиває текст на окремі слова. Це найпоширеніший підхід і особливо ефективний для мов з чіткими межами слів, таких як англійська. Як він працює наведено на рис. 2.9

- Токенізація за символами: Текст розбивається на окремі символи. Цей метод корисний для мов, де відсутні чіткі межі слів або для завдань, які вимагають детального аналізу, такого як виправлення помилок у написанні. Приклад наведено на рис. 2.10:

Рисунок 2.10 – Приклад токенізації за символом [15]

- Токенізація за підсловами: Ставлячи на збалансований підхід між токенізацією за словами і за символами, цей метод розбиває текст на одиниці, які можуть бути більшими за один символ, але меншими за повне слово. Наприклад, "Chatbots" може бути токенізовано як "Chat" і "bots". Цей підхід особливо корисний для мов, де значення формуються шляхом поєднання менших одиниць або при роботі з невідомими словами в завданнях обробки

природної мови (NLP). Даний метод буде використовуватись для підготовки наших даних, більш детальний приклад наведено на рис. 2.11:

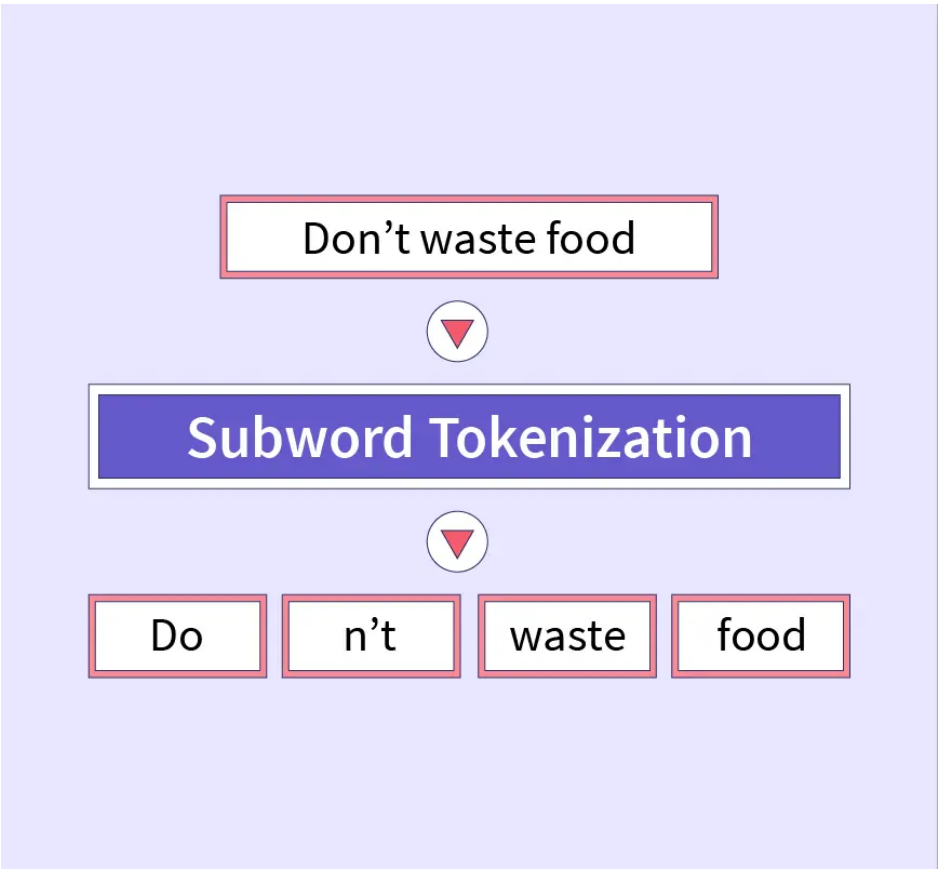


Рисунок 2.11 – Токенізація за півсловом [16]

Розглянувши різні види токенізації слів, варто відмітити, що найчастіше використовують саме токенізацію за словами, проте це все ж таки залежить від конкретної задачі.

2.2.3 Вкладання слів (Word embeddings)

Як було вже описано в пункті 2.2.1, після процесу токенцізації тексту ми переходимо до процессу створення словників, які можуть представляти токен у

багатовимірному просторі у вигляді числового вектору. Ці вектори - це числові представлення значення токену та його взаємозв'язку з іншими токенами у нашому словнику. Однак модель повинна спочатку вивчити ці представлення. Початково вектори ініціалізуються випадковим чином, але під час навчання вони адаптуються для включення певного значення.

Під час навчання ці числові представлення, схожі на ваги в нейромережі, коригуються моделлю для більш точного відображення значення токену та його взаємозв'язку з іншими токенами у нашому словнику.

У обробці природної мови термін "embedding" вказує на процес відображення данних, які не є векторизованими, такими як токени, в векторний простір, який має значення для моделі машинного навчання чи нейромережі. Роблячи це, модель може вивчити взаємозв'язки між словами та їх значенням автоматично, а не ручно вказувати їх. Коли ви візуалізуєте вбудовування слів, ви можете уявляти результат як карту словника, яка показує, як один токен пов'язаний з іншими токенами з точки зору значення. Кожен токен розташований поруч із іншими токенами зі схожим значенням.

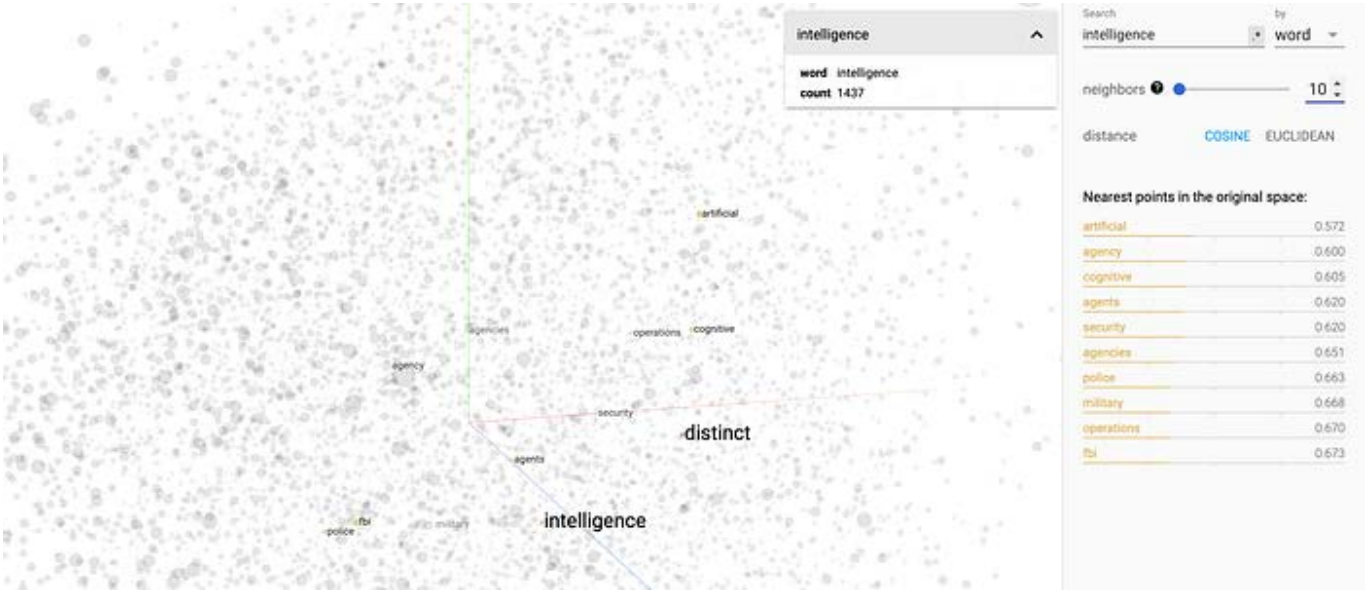
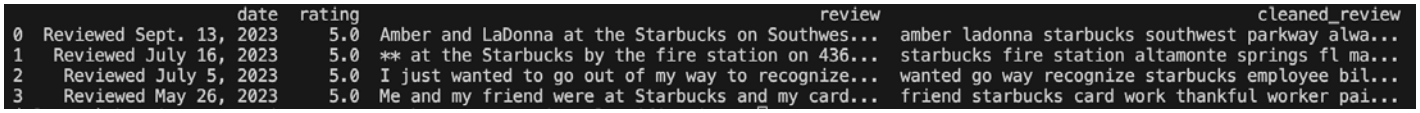


Рисунок 2.12 – Візуалізація вкладання слів (word embeddings) [16]

Тепер перейдемо до наших данних та як це працює в призмі нашого випадку.

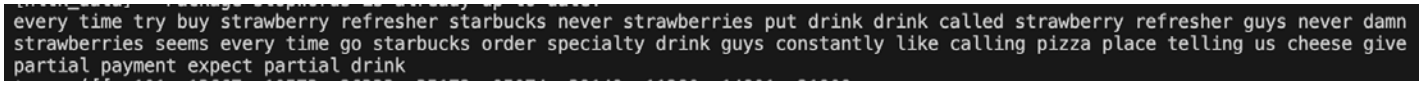
Спочатку покажемо перетворення з роділів 2.2-2.2.1, як вони перетворюють наші данні, на рис. 2.13 показано перші 4 записи:



	date	rating	review	cleaned_review
0	Reviewed Sept. 13, 2023	5.0	Amber and LaDonna at the Starbucks on Southwes...	amber ladonna starbucks southwest parkway alwa...
1	Reviewed July 16, 2023	5.0	** at the Starbucks by the fire station on 436...	starbucks fire station altamonte springs fl ma...
2	Reviewed July 5, 2023	5.0	I just wanted to go out of my way to recognize...	wanted go way recognize starbucks employee bil...
3	Reviewed May 26, 2023	5.0	Me and my friend were at Starbucks and my card...	friend starbucks card work thankful worker pai...

Рисунок 2.13 – Фрагмент датасету після обробки

Наступним кроком стане перетворення токенів у чисельні векори, оскільки математичні моделі не вміють працювати із тектсом напряму. Для цього ми будемо використовувати BertTokenizer, який на виході видає словник, кожен елемент якого це тензор, який в майбутньому буде передаватися в нашу математичну модель. Розглянемо як він працює на прикладі, для демонстрації всіх наступних операцій будемо використовувати наступний шматок датасету, який пройшов всі попередні етапи обробки:



every time try buy strawberry refresher starbucks never strawberries put drink drink called strawberry refresher guys never damn strawberries seems every time go starbucks order specialty drink guys constantly like calling pizza place telling us cheese give partial payment expect partial drink

Рисунок 2.14 – Демонстраційний фрагмент тексту для наступних перетворень

```
from utils import sentiment_analyser

data = pd.read_csv("data/final/processed.csv")


tokenizer = BertTokenizer.from_pretrained("models/tokenizer")

inputs = tokenizer(data.cleaned_review.loc[11], return_tensors="pt")


print(data.cleaned_review.loc[11])
print(inputs["input_ids"])
print(inputs["token_type_ids"])
print(inputs["attention_mask"])
```

Рисунок 2.15 – Імплементація BertTokenizer

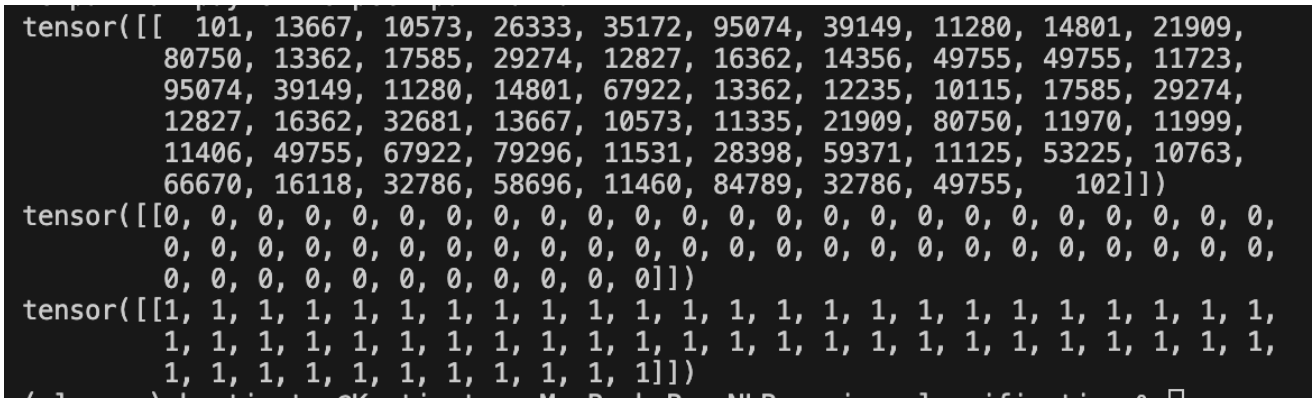


Рисунок 2.16 – Результат виконання

На рис 2.15 – представлено код імплементації Bert токенізатора, розберемо детальніше, що повертає нам цей токенізатор.

Перший тензор представляє собою *input_ids*, які є ідентифікаторною репрезентацією кожного токена.

Другий тензор – це *token_type_ids*, який є бінарною маскою, що визначає, до якої послідовності належить токен. Якщо у нас є лише одна послідовність, то всі ідентифікатори типів токенів будуть дорівнювати 0.

Третій рядок – це *attention_mask*, яка є бінарною маскою, що визначає, чи є токен реальним словом чи просто заповненням. Якщо токен містить [CLS], [SEP] або будь-яке реальне слово, то маска буде дорівнювати 1. У той час, якщо токен є лише заповненням або [PAD], то маска буде дорівнювати 0. – [16]

Фактично ми можемо розкодувати перший тензор в реальні токени наступним фрагментом коду:

example_text = tokenizer.decode(bert_input.input_ids[0])

І в результаті отримаємо наступне представлення:

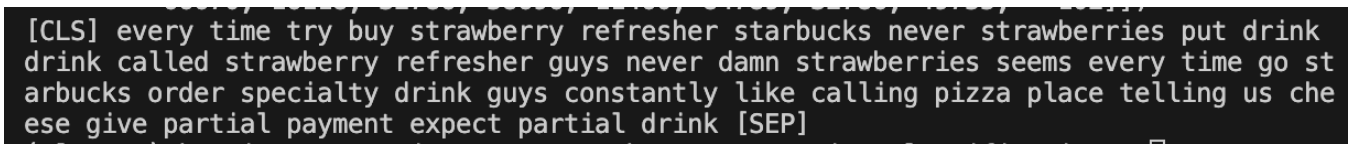


Рисунок 2.17 - *input_ids,* ідентифікаторна репрезентація кожного токена

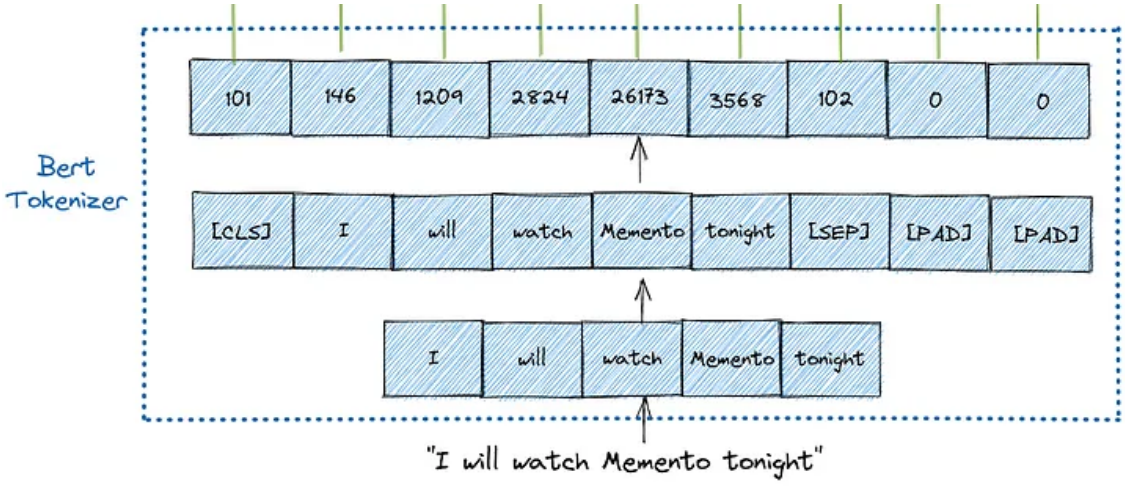


Рисунок 2.18 – Візуалізація роботи BertTokenizer [20]

Щоб перейти від векторного представлення до вкладань слів, використаємо наступний фрагмент коду:

```
model = BertModel.from_pretrained("local_path_to_model")
example_token_id =
tokenizer.convert_tokens_to_ids(data.cleaned_review.loc[11])[0]
example_embedding =
model.embeddings.word_embeddings(torch.tensor([example_token_id]))
print(example_embedding.shape)
# torch.Size([1, 768])
```

Як бачимо в результаті отримаємо тензор з розмірністю (1, 768) – це і буде наше багавмірне представлення вектора слова у вкладенні слів, цей тензор готовий до передачі у математичну модель та подальшому тренуванні.

Для перевірки семантики слів(перевірки на схожість) часто використовують косинус подібності, тож введемо це поняття. Косинусна подібність - вимірює подібність між двома векторами, розраховуючи косинус кута між цими двома векторами. Для розрахунку косинусної схожості ми дивимося на кут між двома векторами.

Якщо вектори вказують в одному напрямку, вони більш схожі, і якщо вони вказують в протилежних напрямках, вони менш схожі. Результатом є число від -1 до 1, де 1 означає, що вектори ідентичні, а -1 означає, що вони абсолютно різні. – [22]

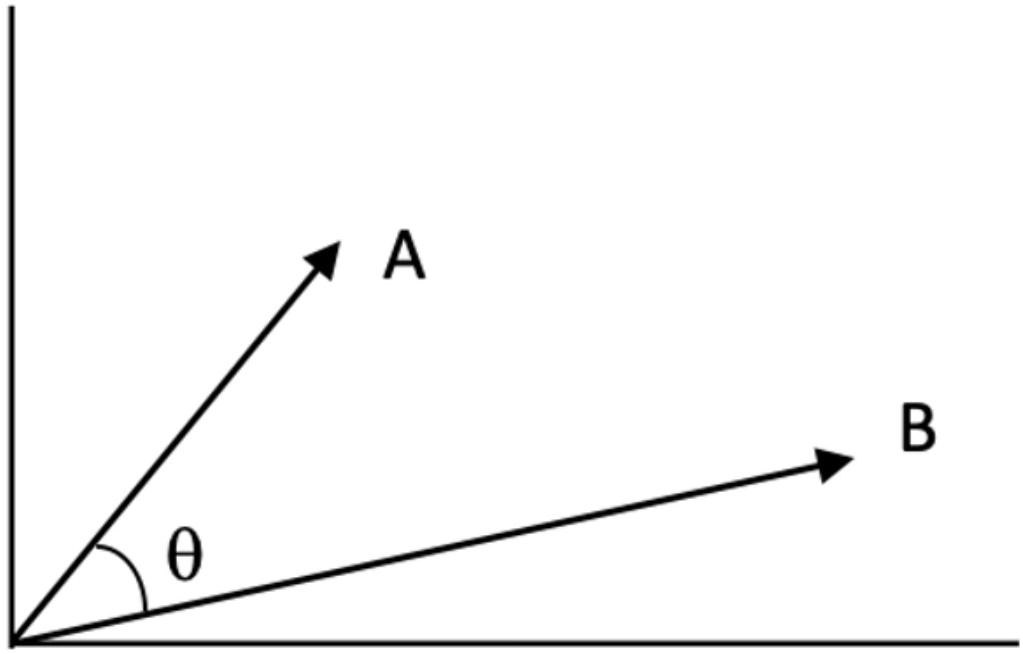


Рисунок 2.18 – Косинусна подібність між векторами A та B

І розраховується за формулою:

$$\cos\theta = \frac{A * B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \quad (2.1)$$

Для прикладу поріняємо два слова та перевіримо як працює наш вкладення слів(word embedinng) за допомогою наступного фрагменту коду:

king_token_id = tokenizer.convert_tokens_to_ids(["king"])[0]

```
    king_embedding =
model.embeddings.word_embeddings(torch.tensor([king_token_id]))
    man_token_id = tokenizer.convert_tokens_to_ids(["man"])[0]
    mann_embedding =
model.embeddings.word_embeddings(torch.tensor([man_token_id]))

    cos = torch.nn.CosineSimilarity(dim=1)
    similarity = cos(king_embedding, man_embedding)
    print(similarity[0])
```

З представленого фрагменту коду бачимо, що слова король та чоловік мають косинусну подібність у 0.8469, отже, мають однаковий напрямок у n-вимірному просторі і знаходяться досить близько, що цілком логічно.

2.2 Модель нейронної мережі BERT

Виходячи з висновків із розділу 1, де ми порівняли різні архітектури і прийшли до того, що найкраще для розв’язання поставленої задачі підійдуть нейронні мережі на базі трансформерів і одним з найпопулярнішим і водночас ефективним рішенням є архітектура нейронної мережі BERT, далі поглянемо на цю архітектуру більш детально.

BERT – це акронім від Bidirectional Encoder Representations from Transformers. Саме назва дає нам кілька підказок щодо того, що таке BERT. Архітектура BERT складається з кількох величезних багатоярусних трансформерних кодерів. Кожен трансформерний кодер включає два підшари: шар самоуваги та шар передачі зворотнього зв'язку.

Ключовим технічним досягненням BERT є застосування двохстороннього тренування трансформера, популярної моделі уваги, для мовного моделювання. Це відмінно від попередніх спроб, які розглядали текстову послідовність або зліва направо, або об'єднували тренування зліва направо та справа наліво. Результати статті показують, що мовна модель, яка тренується бідирекційно, може мати глибший розуміння мовного контексту та логіки, ніж моделі з одною напрямленістю. – [23].

Як ми вже 1.1.1 пояснювали трансформер блоки – блоки що складаються з енкодер та декодер частин, більш наглядно показано на рис. 2.19

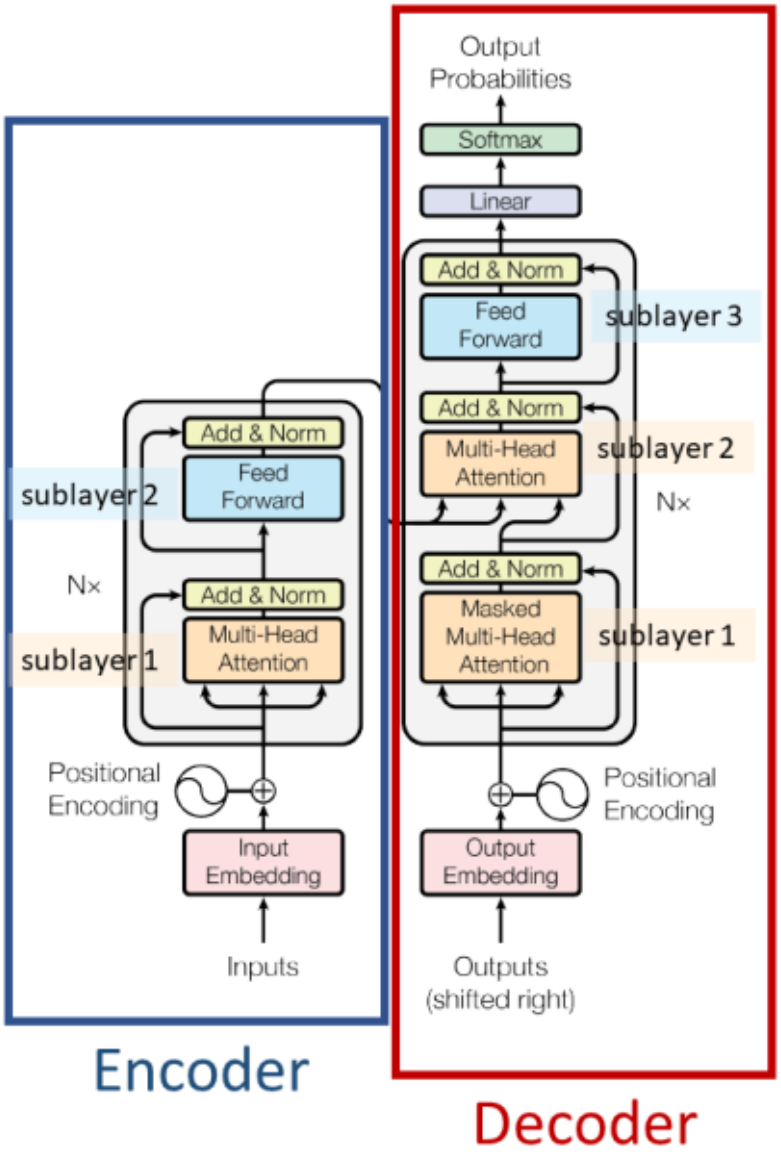


Рисунок 2.19 – Енкодер – декодер частини трансоффрмеру [23]

BERT використовує лише частину енкодера з архітектури трансформера, скасовуючи декодер. Так само, як трансформер, BERT також використовує позиційне кодування, самоувагу, багатоголовкову увагу та залишкове з'єднання. Він використовує точно таку ж архітектуру енкодера, яка використовується в трансформері. Узагальнюючи, BERT – це, в основному, просте об’єднання енкодерів, які саме були взяті з трансформер архітектури, ліва частина рис. 2.19

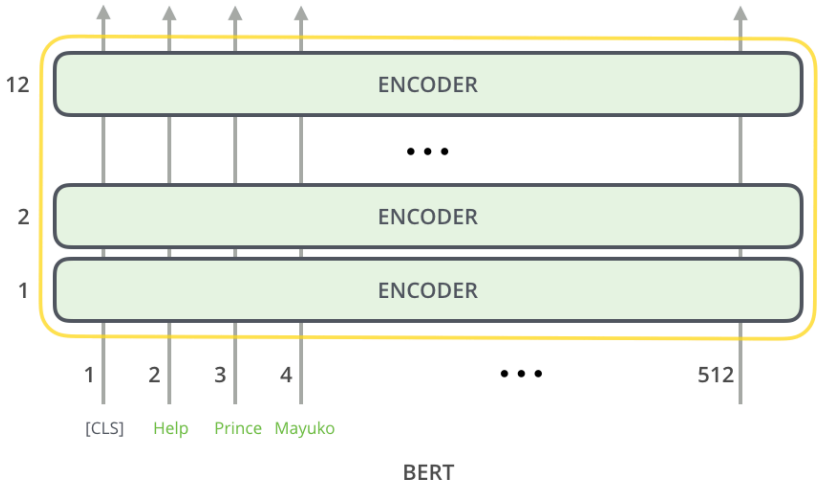


Рисунок 2.20 – Високорівневий вид трансформеру - [23]

Щоб було простіше пояснити як працює BERT, розіб’ємо цю архітектуру на 4 основні частини і пояснимо кожну з них окремо:

- Частина перетворення слова на вектор
- Енкодери з архітектури трансформера
- Масковане моделювання мови (MLM)
- Прогнозування наступного речення (NSP)

Першу та другу частину ми вже покрили у розділах 2.2.2 та 1.1.2 відповідно, тепер розглянемо компоненту масковане моделювання мови.

2.2.1 Масковане моделювання мови MLM

Маскована мовна модель випадковим чином маскує деякі токени вхідного тексту, і мета полягає в тому, щоб передбачити оригінальний ідентифікатор словника маскованого слова лише на підставі його контексту. На відміну від оригінального мовного моделювання зліва направо, завдання MLM дозволяє представленню об'єднувати контексти з обох сторін, що дозволяє нам попередньо тренувати глибокий бідирекційний трансформер – [23]. Перед тим як подавати послідовності слів на вхід до BERT, 15% слів в кожній послідовності замінюються токеном [MASK]. Потім модель намагається передбачити оригінальне значення замаскованих слів на основі контексту, який надають інші, не замасковані, слова в послідовності. З технічної точки зору передбачення вихідних слів передбачає:

1. Додавання класифікаційного шару поверх виходу енкодера.
2. Помноження векторів виходу на embedding матрицю, перетворюючи їх в розмір словника.
3. Розрахунок ймовірності кожного слова в словнику за допомогою функції softmax, можна побачити у формулі 1.7

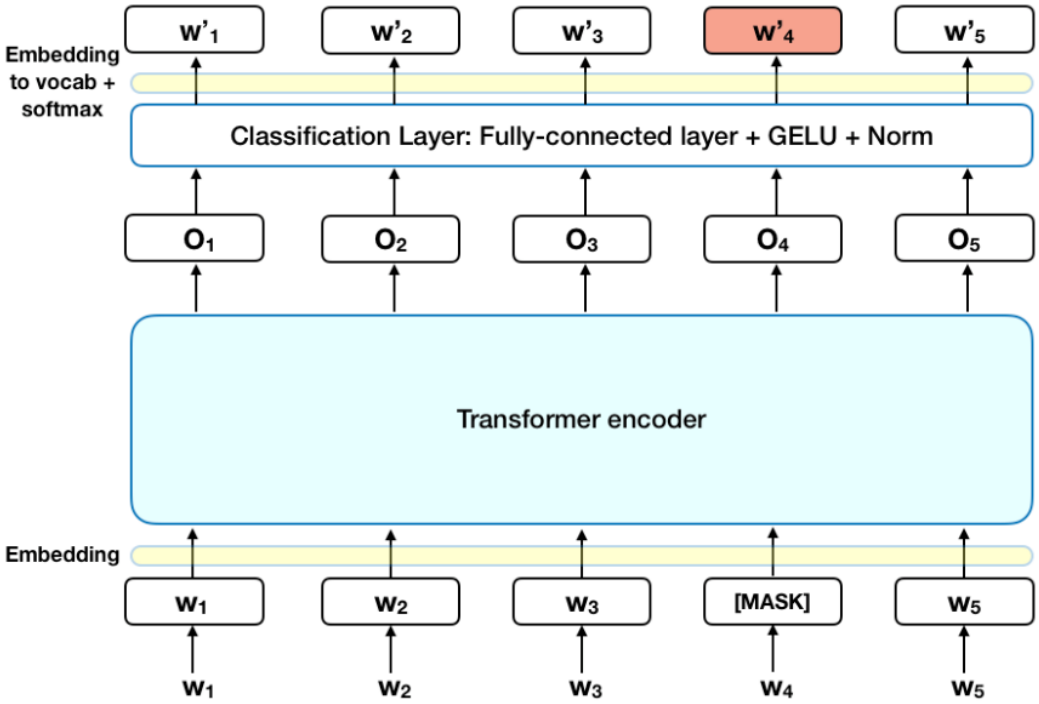


Рисунок 2.20 – Візуалізація Bert на рівні виходів [23]

Тут w$_1$, ..., w$_5$ це вектори word embedding, а O$_1$, ..., O$_5$ виходи від BERT.

BERT використовує активацію GELU (Gaussian Error Linear Unit).

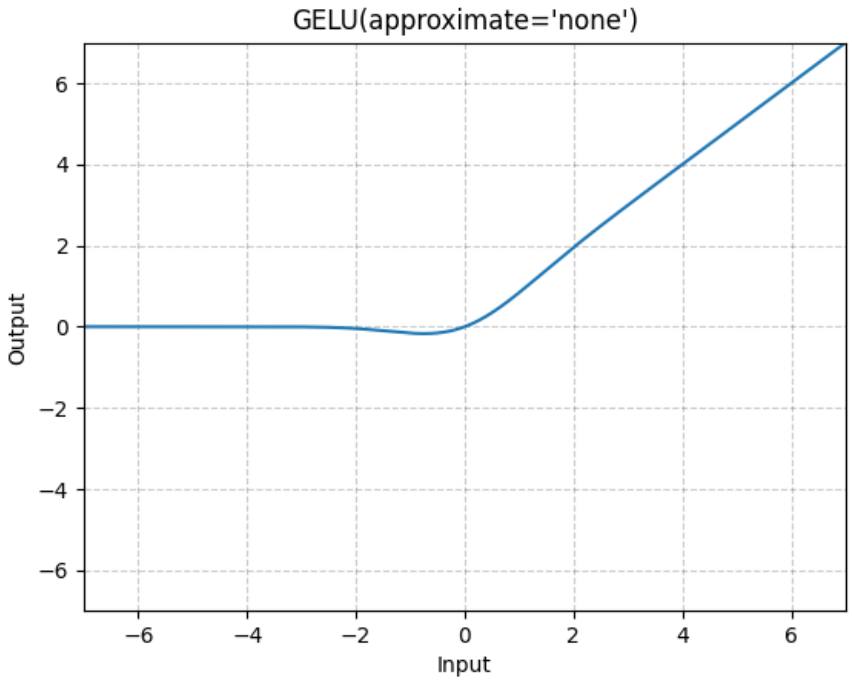


Рисунок 2.21 – Графік функції GELU

Формула для розрахунку GELU:

$$\text{GELU(x)} = 0.5x\left(1 + \tan\left(\sqrt{\frac{2}{\pi}} * \ (x + 0.044715 * x^3)\right)\right) \tag{2.2}$$

Як бачимо дургий множник у формулі 2.2 це кумулятивна функція розподілу (CDF) для гауссівого (нормального) розподілу відображає ймовірність того, що випадкова величина приймає значення менше або рівне заданій точці.

Функція втрат BERT враховує лише передбачення маскованих значень і ігнорує передбачення не-маскованих слів. Внаслідок цього модель збігається повільніше, ніж напрямлені моделі, ця характеристика компенсується її підвищеною освідомленістю контексту. Огляд методу маскованого мовлення (MLM) в BERT виглядає наступним чином:

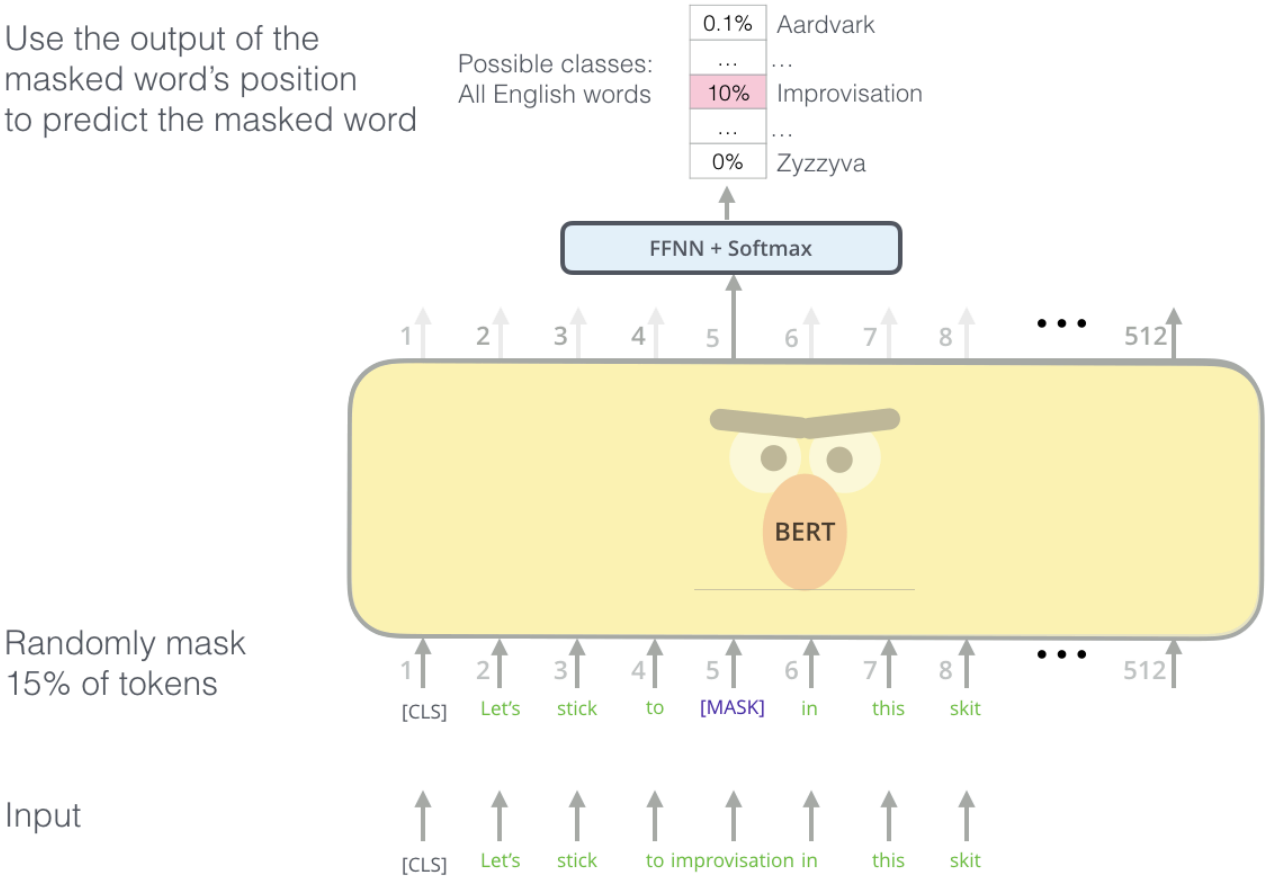


Рисунок 2.22 – Візуалізація роботи MLM компоненти [23]

2.2.2 Прогнозування наступного речення (NSP)

Щоб поліпшити здатність BERT до обробки відносин між кількома реченнями, процес попереднього навчання включає додаткове завдання: дано два речення (A і B), чи ймовірно, що B є реченням, що слідує за A, чи ні?

Під час процесу навчання BERT модель отримує пари речень на вході і вивчає передбачення, чи є друге речення в парі наступним реченням в оригінальному документі. Під час навчання 50% вхідних даних - це пара, в якій друге речення є наступним в оригінальному документі, тоді як у інших 50% випадків випадкове речення з корпусу вибирається як друге речення. Припущення полягає в тому, що випадкове речення буде відокремлене від першого речення. Щоб допомогти моделі відрізняти ці два речення під час навчання, вхід обробляється наступним чином, перш ніж потрапити в модель:

Токен [CLS] вставляється на початок першого речення, а токен [SEP] додається в кінець кожного речення. До кожного токена додається вбудовування речення, яке вказує на речення A або речення B. Вбудовування речення схоже за концепцією на токенові вбудовування з словником розміром 2.

До кожного токена додається позиційне вбудовування, яке вказує на його позицію в послідовності. Концепцію та реалізацію позиційного вбудовування представлено на рисунку 2.23.

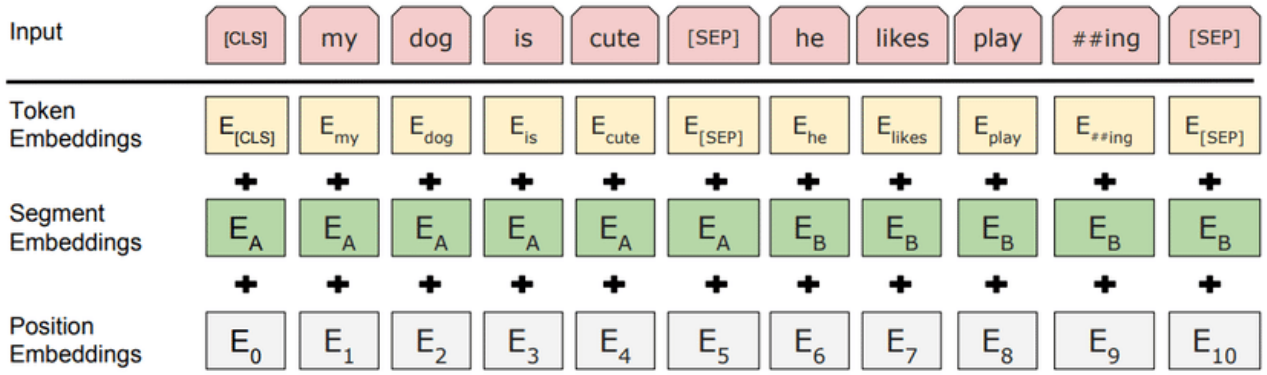


Рисунок 2.23 – Позиційне вбудовування [23]

Щоб передбачити, чи дійсно друге речення пов'язане з першим, виконуються наступні кроки:

1. Вся вхідна послідовність проходить через модель Трансформатора. (розмір вектора вводу - 768 для кожного слова в BERT base)
2. Кожна позиція виводить вектор розміром hidden_size (768 в BERT Base).
3. Вивід токена [CLS] трансформується в вектор форми 2x1, використовуючи простий шар класифікації (навчені матриці ваг і зсувів).
4. Розрахунок ймовірності IsNextSequence (мітки IsNext і NotNext) з використанням softmax.

Під час тренування моделі BERT Masked LM та Next Sentence Prediction тренуються разом з метою мінімізації комбінованої функції втрат обох стратегій. Тепер загальна модель BERT у цьому виглядає наступним чином:

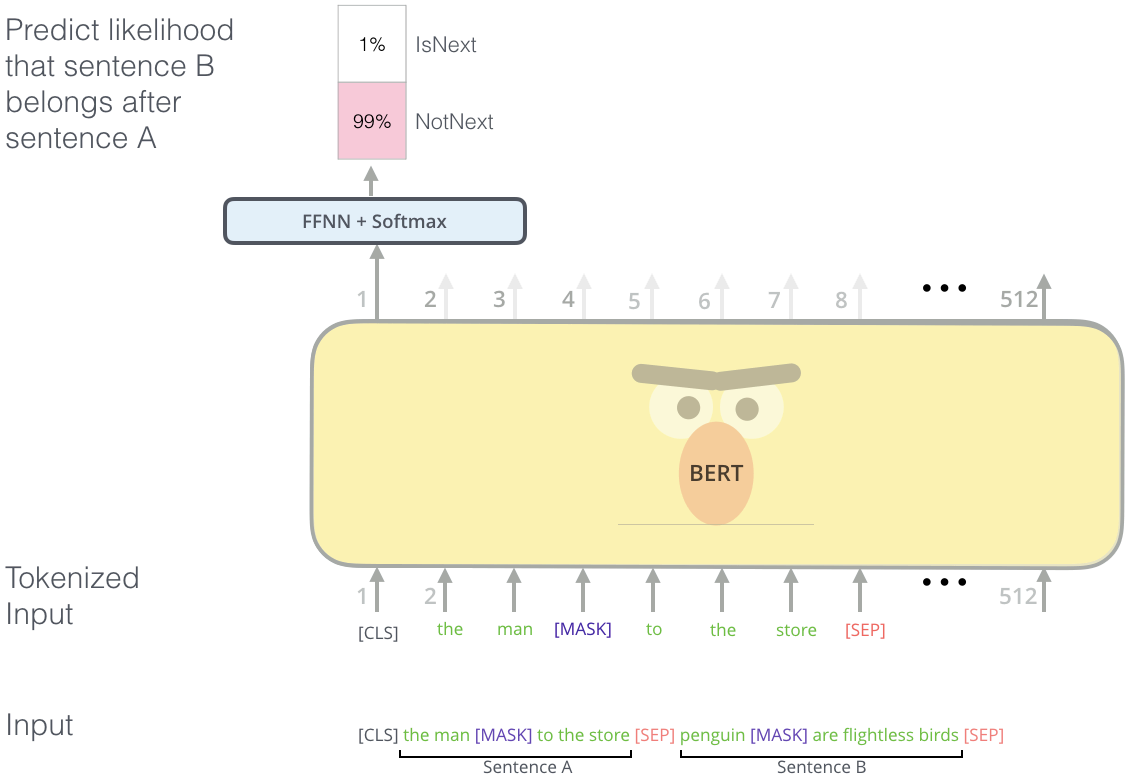


Рисунок 2.24 - Всі компоненти загалом [23]

2.3 Висновки до розділу

В цьому розділі було докладно описано математичні підходи до обробки текстових даних, також зроблено аналітичний аналіз початкового датасету, були продемонстровані розподіли міток в різних розрізах, показано об'єм даних та методи додавання даних для збільшення та сбалансування датасету.

Було запроваджено новий алгоритм до поділу данних на тренувальний, валідаційний та тестовий датасет, де використовується ідея хешування.

Було описано алгоритм передобробки даних, а саме методи очистки тексту і методи перетворення тексту у векторні представлення для майбутньої подачі у нероннy мережу.

Зроблено огляд на архітектуру обранної математичної моделі, прниципи її взаємодії з вхідними даними, внутрішні нюанси імплементації та методи побудови великих мовних моделей.

3 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Принципи взаємодії системи

Для того щоб краще описати як взаємодіють програмні компоненти у цілому, одне з одним представимо діаграму у UML нотації на рисунку 3.1

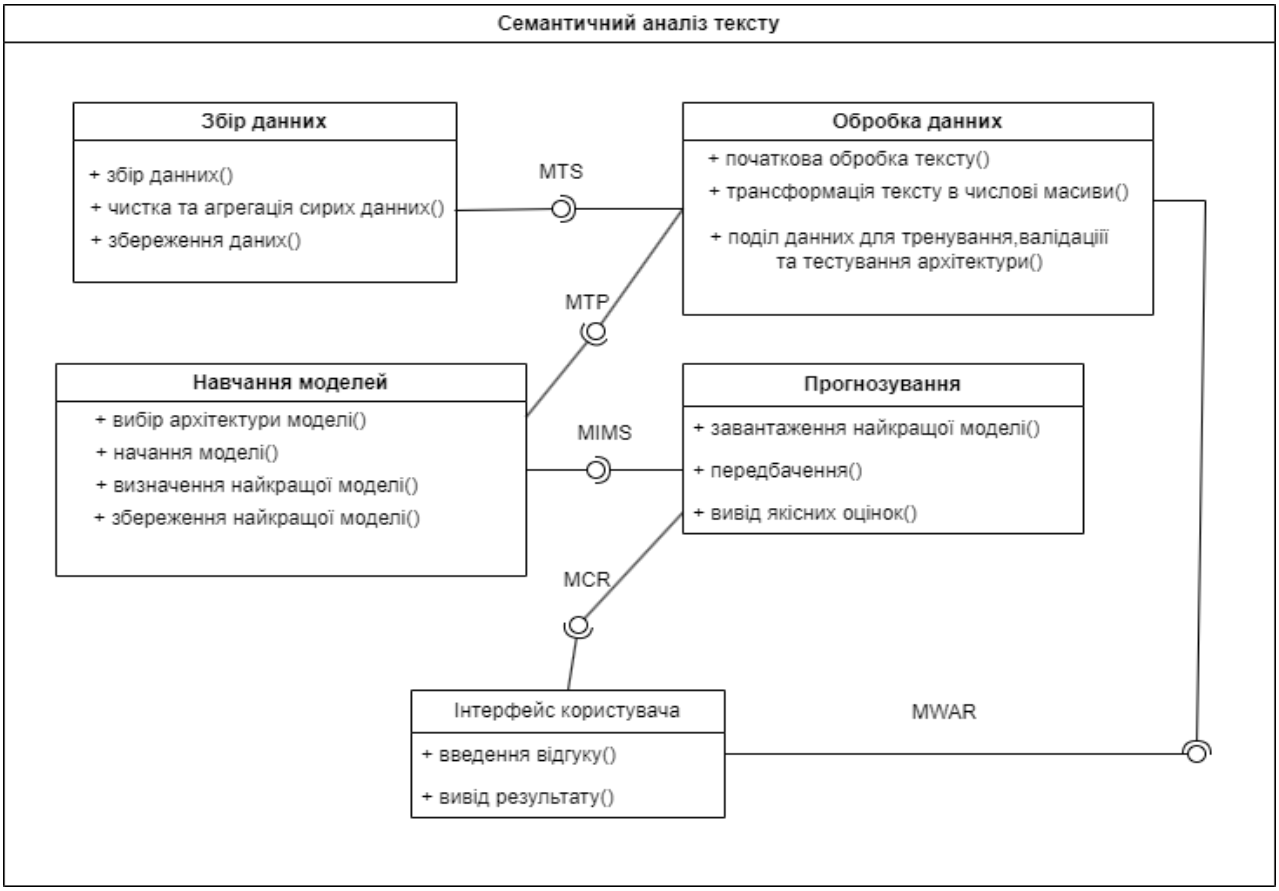


Рисунок 3.1 – UML діаграма програмних компонентів

Докладніше розглянемо кожну з компонентів, а саме компоненти архітектури системи забезпечує виконання таких основних функцій:

- збір, завантаження та збереження даних – MTS (Module Text Save);
- попередня обробка тексту – MTP (Module Text Processing);
- класифікація тексту – MIMS (Module Identification Model Save);

- тренування, ініціювання, валідацію та збереження нейронної мережі – MWAR (Module Web Application Raw);
- надання інтерфейсу для користувача – MCR (Module Classification Result).

Компонентна модель системи семантичного аналізу, що реалізує вказані функції складається з наступних частин : збір відгуків, обробка тексту, навчання моделей, передбачення моделі та web-застосунок. – [24]

Деякі з цих компонент вже описані у попередніх розділах, а саме MTS, MTP та MIMS описані в розділі 3. Розглянемо тільки деякі детал імпелементації цих компонентів, краще розглянемо більш детально компоненти MCR та MWAR.

3.1.1 MWAR компонента

Далі поговримо про API яка виористовується у дизайні математичної моделі нашої нейронної мережі. Для побудови таких великих та комплексних архітектур нейронних мереж використовується один з провідних фреймворків для дизайну, імплементаціїї нейронних мереж – pytorch. PyTorch — це оптимізована бібліотека тензорів для глибокого навчання з використанням графічних та центральних процесорів. Дуже потужний фреймворк у розумінні якості написання великих шматків прогрмних компонентів для роботи з глибинними нейронними мережами, його переваги над іншими фреймворками такими як Tensorflow та Keras, це саме можливість управління усіма математичними компонентами на дуже низько рівневому рівні, даючи можливість робити майже будь-які маніпуляції з данними та побудувою моделей, що стає дуже вагомою перевагою серед інших фреймворків. Порівняльна таблиця інших фреймворків представлено у таблиці 3.1

Таблиця 3.1 – Порівняння популярних фреймворків для роботи з нейронними мережами

Критерій	TensorFlow	PyTorch	Keras
Популярність	Найбільш популярний	Другий за популярністю	Третій за популярністю
Зручність використання	Помірно зручний	Відносно зручний	Дуже зручний
Гнучкість	Дуже гнучкий	Дуже гнучкий	Менше гнучкий
Обчислювальний граф	Статичний граф (TensorFlow 1.x), Динамічний граф (TensorFlow 2.x)	Динамічний граф	Статичний граф
Підтримка спільноти	Велика і активна	Росте і активна	Велика і активна
Крива навчання	Спочатку крута через статичний граф (TensorFlow 1.x)	Середня	Легка
Розгортання	Кращий для виробничого розгортання в деяких сценаріях	Гнучкі опції розгортання	Інтегрований з TensorFlow для розгортання
Інтеграція з іншими бібліотеками	Обширна	Найбільш обширна	Менш обширна

Як бачимо PyTorch є дуже комплексним інструментом, який важкий у освоєнні але є в якомусь розумінні швейцарським ножем у роботі з нейронними мережами.

3.1.2 Файлове представлення моделей

На этапі тренування, валідації та ініціалізації ми отримаємо готовий бінарний файл моделі, де факто він представляє собою тензори з вагами кожного з нейронів нейронної мережі та деякі метаінформацію якщо потрібно, які потім будуть завантажуватись уже на этапі графічного інтерфейсу користувача. Окрім самої моделі, яку ми зберігаємо у якості бінарного файлу, також збереження бінарника відбувається для Токенізатору, який перетворює векторне представлення слова у словникове вбудовування, де всередині зберігаються словники слів, а також метаінформація, що стосується токенцізатора, а саме мітки масок, відступів, розділючих знаків тощо, більш детально можна прочитати у розділі 2, підрозділі 2.2.2. на сторінці 46. Як виглядать ці файли продемонструємо на рисунку 3.2.

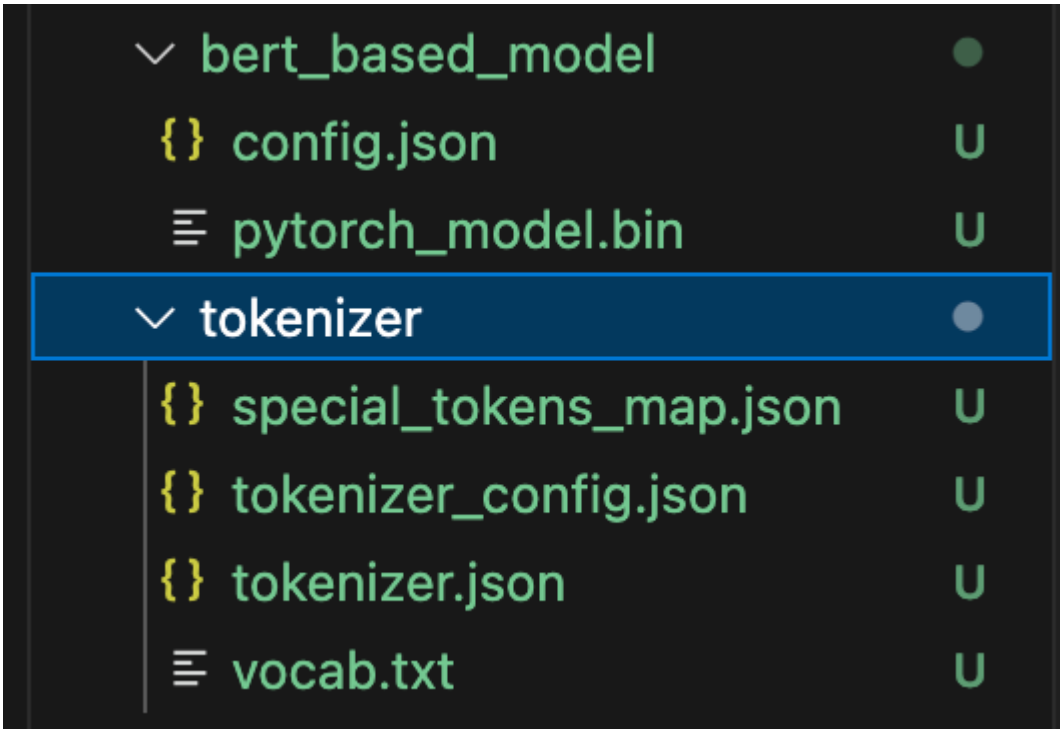


Рисунок 3.2 – Візуалізація бінарних файлів моделей

3.1.3 Логування експериментів

На етапі експериментів з моделями доволі часто потрібно тренувати модель багато ітерацій, але дуже часто проблемно стає відслідковувати результати кожного такого тренувального циклу, саме тому один з найзручніших рішень на сьогодні це бібліотека MlFlow для відстеження та порівняння результатів різних експериментів. Інтерфейс даної бібліотеки показано на рисунку 3.3:

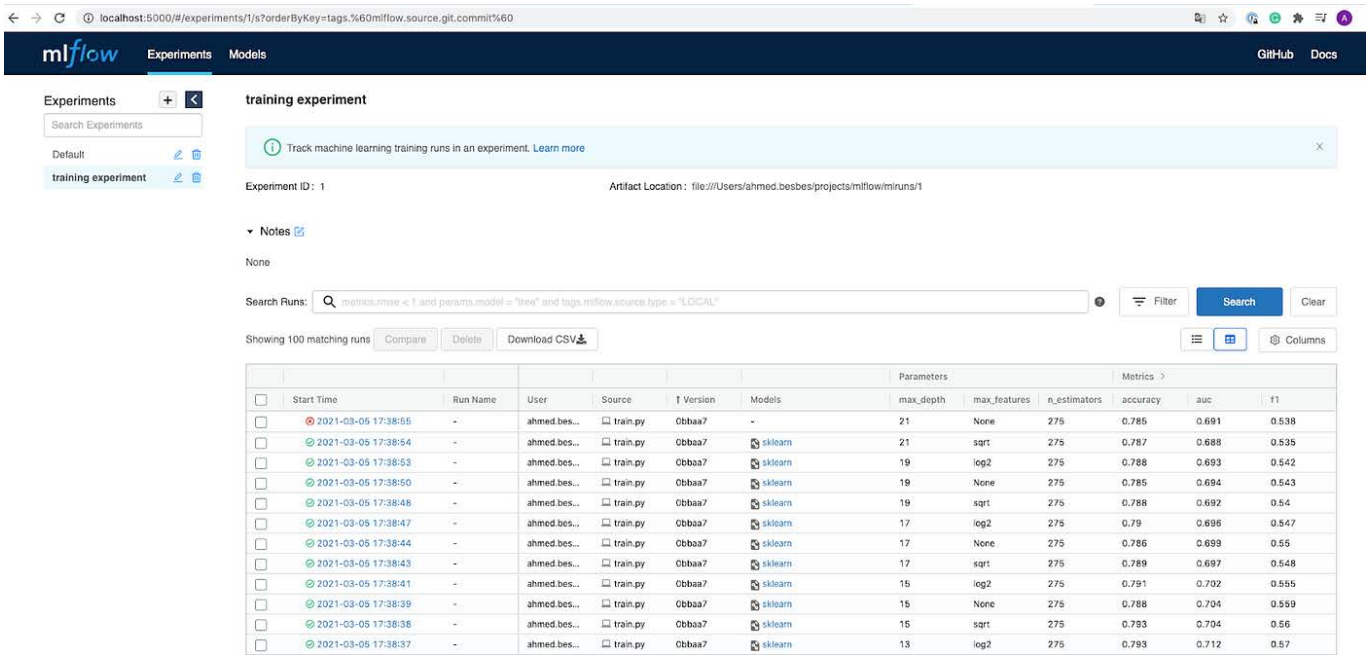


Рисунок 3.3 – Інтерфейс MLflow бібліотеки

Дана бібліотека дуже зручна для логування будь-яких артефактів, в тому числі графіків навчання моделі, метаінформацію пов’язану з тренуванням моделі, а також приклади вхідних та вихідних даних з типізацією, що дозволяє в майбутньому легше інтегрувати моделі у продакшен, а також налаштовувати автоматизацію моделей, які

будуть тренуватися автоматично порівнювати результати з попередніми моделями і в разі кращих метрик помічатися, як найкраща модель.

3.2 MCR компонента. Інтерфейс користувача

Для реалізації даного компоненту було використано популярний фреймворк з відкритим кодом Flask, його задача зчитати на ініціалізувати вже навчену модель та токенізатор, який буде використовуватись у інференсі. Далі проводиться користувацький ввід даних, варто відмітити, оскільки тренувальні дані представлені тільки англійською мовою, то класифікатор працювати буде тільки з англійським текстом, в майбутньому планується покращити даний фактор, та створити підтримку багатьох мов, після користувацького вводу, моделі опрацьовує вхідні данні та видає оцінку тексту від 1 до 5. Після цього відбувається http запит до серверу, де всередині модель робить передбачення і сервер видає результат вже до кінцевого користувача. Приклад взаємодії описано на рисунку 3.4:

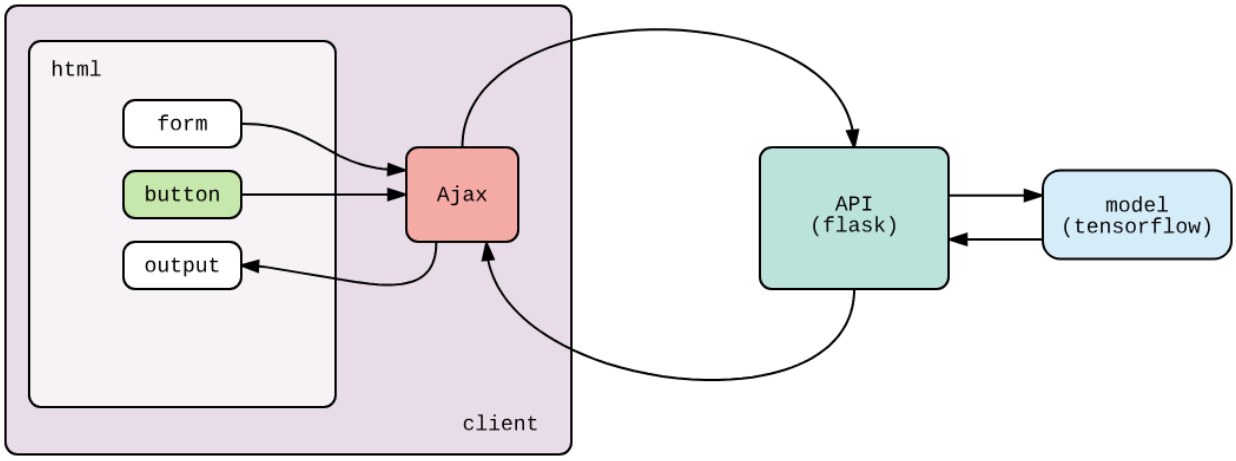


Рисунок 3.4 – Принцип роботи hhtps запитів на flask сервер

Як бачимо фронтенд частина реалізована на таких мовах програмування як: HTML, CSS, JavaScript. І він має такий вигляд:

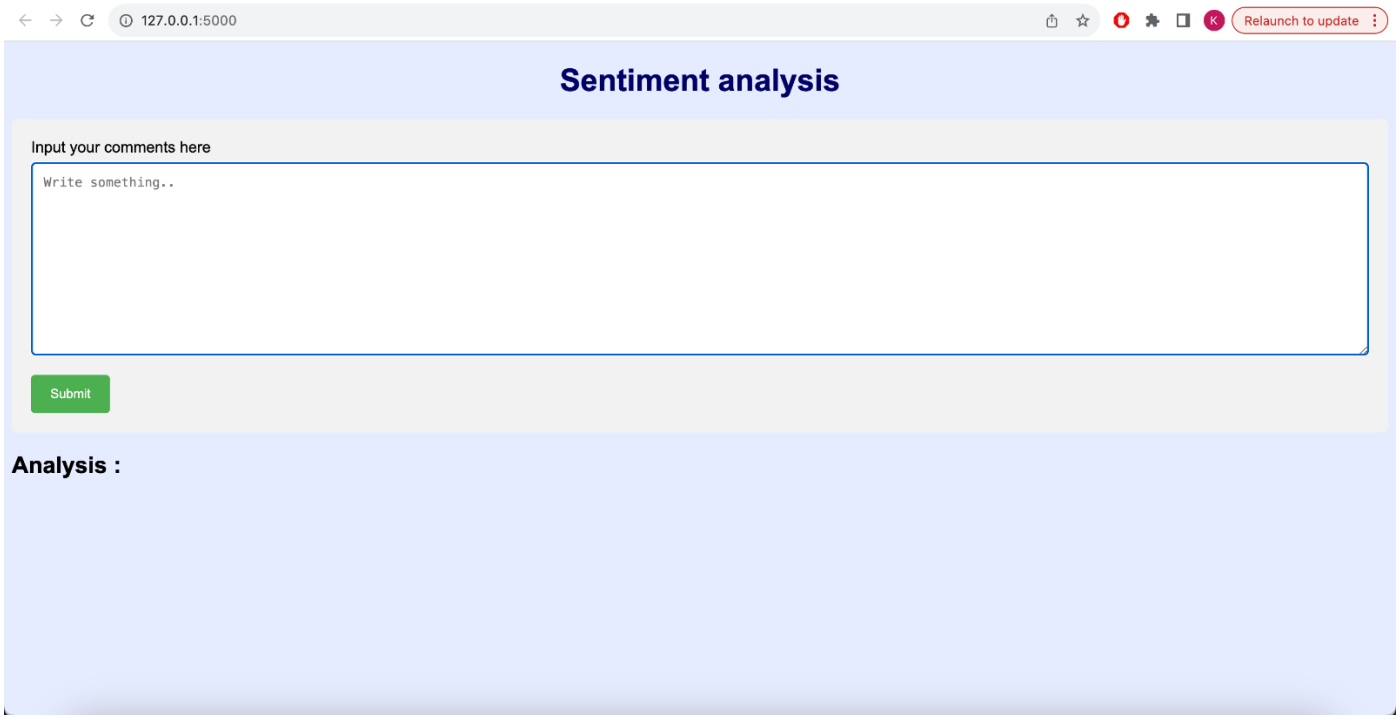


Рисунок 3.5 – Інтерфейс користувача

Далі продемонструємо приклади взаємодій з програмним засобом, для початку представимо результат введення одного коментаря, коментар візьмемо з відкритих джерел:

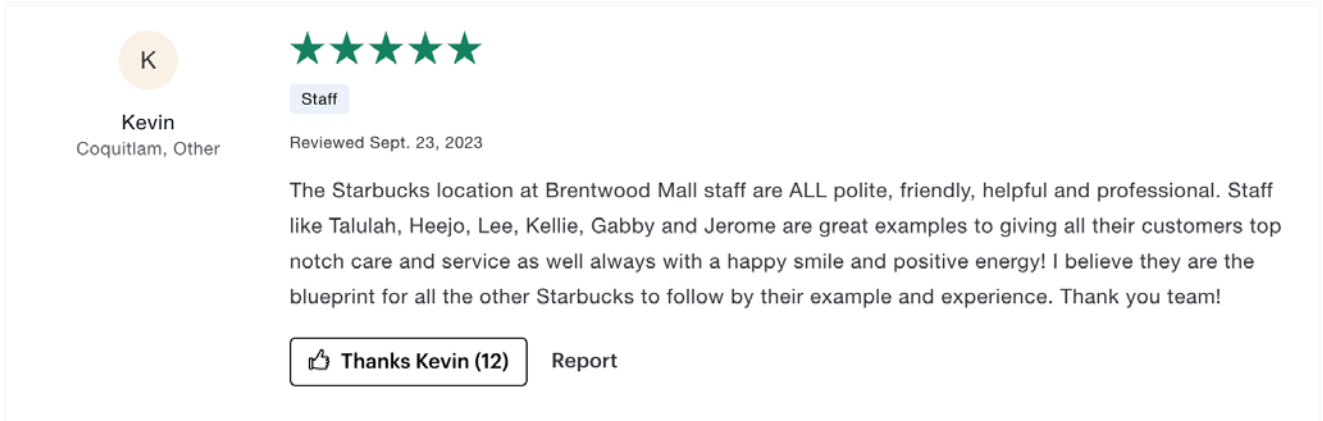


Рисунок 3.6 – Приклад коментаря з відкритих джерел

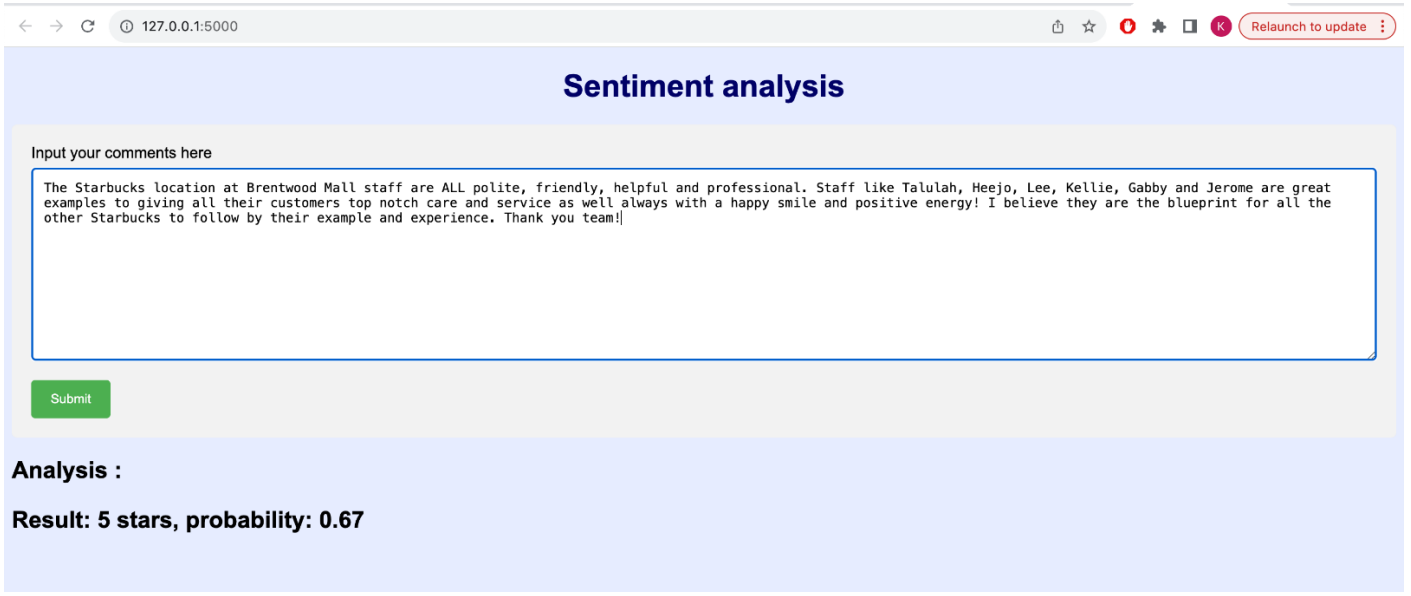


Рисунок 3.7 – Приклад взаємодії з програмним засобом

Як бачимо модель оцінила цей коментар на 5 балів, також модель видає вірогідність отриманого результату, рівень впевненості моделі. Але не зважаючи на це потрібно розуміти, що якщо модель видала низьку вірогідність якоїсь оцінки, то її зміщення в радіусі 1-го балу буде мати буде високу вірогідність і скоріше за все правильну оцінку.

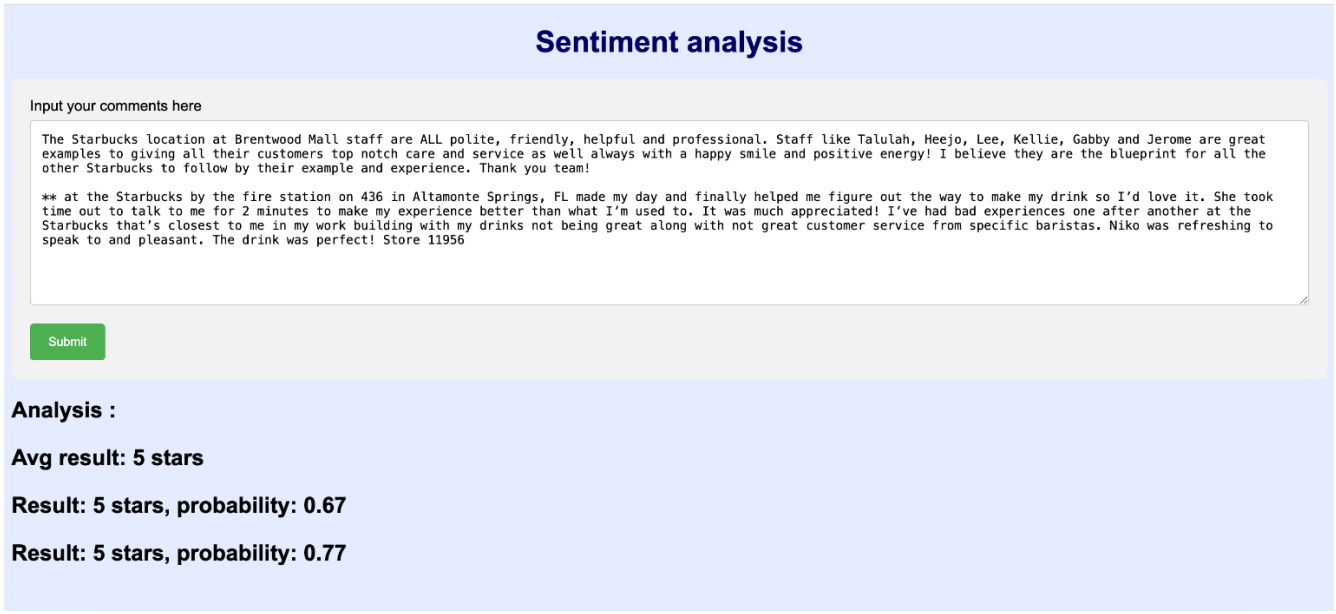


Рисунок 3.8 - Приклад взаємодії з програмним засобом

На рисунку 3.8 бачимо приклад входу кількох коментарів, в такому разі користувач отримає середню оцінку серед усіх коментарів, а також оцінку та вірогідність кожного з коментарів окремо, варто також згадати, що для входу n-ої кількості коментарів варто подавати їх відокремивши пустою лінією, як розмежувальним символом, в іншому випадку модель розцінить весь коментар як один. Продемонструємо мої слова на прикладі рисунку 3.9, де було об’єднано коментарі із рисунку 3.8

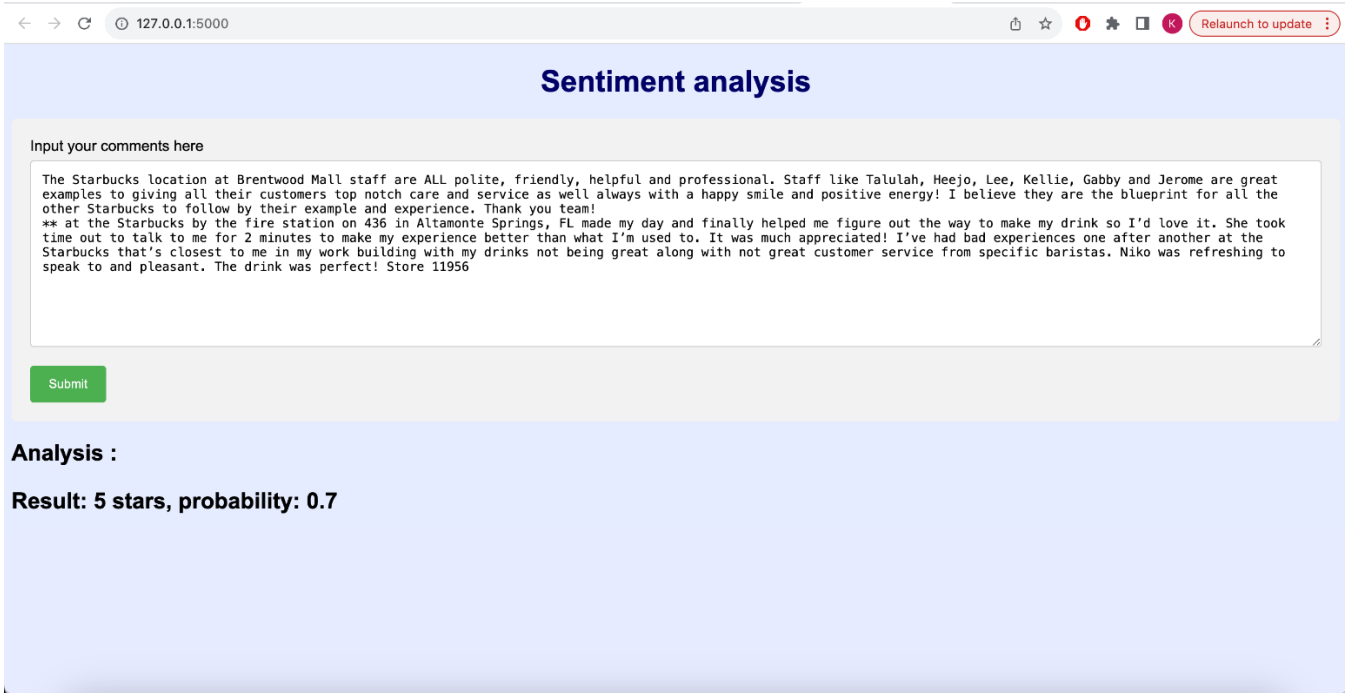


Рисунок 3.9 - Приклад взаємодії з програмним засобом

Також продемонструємо як програма усереднює результати у випадку, коли вхідні комментарі користувача різнобічні, щоб продемонструвати саме користь програми у тому що вона зекономить час користувачу у разі великої кількості різноманітних коментарів.

Приклад такої взаємодії показано на рисунку 3.10:

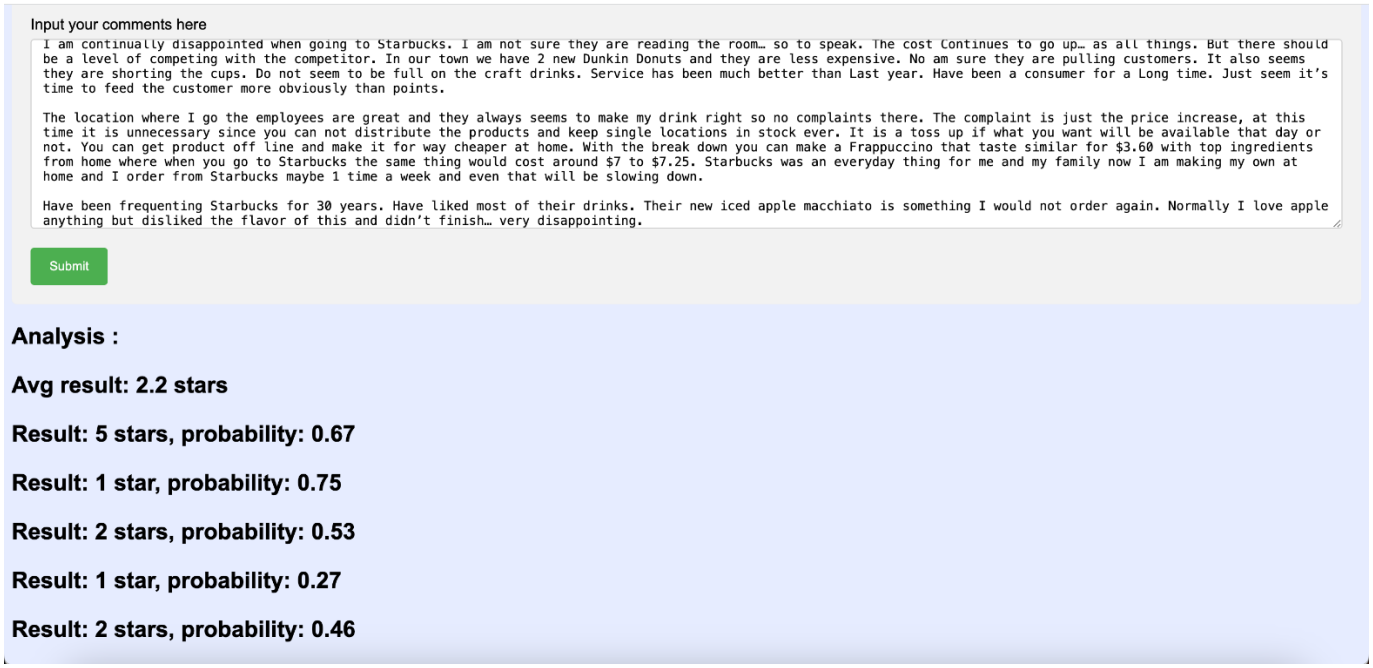


Рисунок 3.10 - Приклад взаємодії з програмним засобом

Як бачимо програма відпрацьовує, з доволі непоганою точністю, час передбачення моделі не займає більше 5 секунд, навіть на великих обсягах даних, оскільки для інференсу використовуються методи паралелізації процесів, що в свою чергу прискорює час роботи моделі значно.

3.3 Висновки до розділу

В цьому розділі було описано технічні нюанси імплементованого проекту, було показано UML діаграму принципів взаємодії програмних компонентів, було порівняно різні фреймворки для роботи з глибинними нейронними мережема – серед яких було обрано PyTorch для імплементації, оскільки він має дуже гнучку взаємодію між математичними компонентами нейронних мереж. Було описано бібліотеки для логування, спостереження та моніторингу нейронних мереж, описано

принципи зберігання нейронних мереж для подальшого використання у інференсі. Було продемонстровано графічний інтерфейс користувача та як з ним взаємодіяти, приведені приклади різних способів використання та оцінено швидкодію застосунку.

ВИСНОВКИ

В даній магістерській дисертації було отримано наступні теоретичні та практичні результати:

1. Було опрацьована література яка відноситься до тем пов'язанних з обробкою природньої мови, було вивчено методи та підходи до роботи з великими текстовими даними, проаналізовано передові технології у сфері глибинних нейроних мережах для роботи з текстовими даними, в результаті чого були обрані архітектури на базі рекурентних нейронних зв'язків та трансформерів, на наших тестових даних архітектура трансформерів показала кращий результат, де 0.93 – accuracy метрика, 0.904 – precision, 0.91 – recall, 0.907 - f1-score.

2. Вивчено методи попередньої обробки текстових даних, порівняно різні види векторних представлень текстових даних, обрано найбільш оптимальний для нашого випадку – розбиття за словами, а також алгоритм перетворення векторного представлення слів у словникові вбудовування для подальшої передачі у нейронну мережу. Також було запропоновано новий метод розбиття даних використовуючи хеш-функцію

3. Обрано та обумовлено вибір моделі BERT, архітектура якої лежить на основі трансформерів, вивчено принципи взаємодії її компонентів між собою, описано принципи як вона взаємодіє з словниковими вбудовуваннями.

4. Проведена порівняльна характеристика фреймворків для роботи з глибинними нейроними мережами, було обрано PyTorch, як один з найперспетивніших та гнучкою архітектурою фреймворку, дозволяючи налаштовувати як низькорівневі компоненти так і використовувати високорівневі конструкції, також він продемонстрував високу швидкість у тренуваннях великих моделей. Оглянуто рішення пов'язані з завантаженям моделей у якості бінарних файлів, що сприяло зменшеню розмірів файлів, які передавали стан натренованої

моделі. Вивчено рішення з автоматизації порівняння моделей, моніторингу та оновлення у продакшені, в якості такого інструменту було оглянуто MLflow

5. Створено графічний інтерфейс користувача для зручної взаємодії з математичною моделлю, оцінено час відгуку математичної моделі який склав до 5 секунд, описано принцип взаємодії програмних компонент системи та деталі імпелементації.

В майбутньому планується розширити функціонал математичної моделі для підтримки декількох мов, а також налаштувати ще більше етапів автоматизації роботи з нейрнними мережами, наприклад автоматичне оновлення найкращої моделі, налаштування нотифікації у випадках коли відбуваються якісь помилки з боку серверу. Також планується перенести це рішення з локальної розробки у хмару і відкрити бета-доступ для користувачів.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Liu B. Sentiment Analysis: Mining Opinions, Sentiments, and Emotions, 2nd Edition / B. Liu – 2015.- P. 110-128.

2. Sequence to Sequence Learning with Neural Networks. [Electronic resource] / I. Sutskever, O. Vinyals. – 2014. – Mode of access: https://arxiv.org/abs/1409.3215

3. Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780. MIT Press.

4. VanderPlas J. Python Data Science Handbook / J. VanderPlas – 2016. - P 203-217.

5. O. Vinyals. Data Science from Scratch, 2nd Edition / O. Vinyals – 2013.- P. 183- 188.

6. Attention is All You Need. [Electronic resource] / A. Vaswani – 2017. – Mode of access: https://arxiv.org/abs/1706.03762

7. D. Rothman. Transformers for Natural Language Processing / D. Rothman – 2019.- p. 493

8. IMDb Datasets [Electronic resource]. Mode of access: https://developer.imdb.com/non-commercial-datasets/ (дата звернення: 18.11.2022)

9. J. P. Muller. Python for Data Science For Dummies, 2nd Edition / J. P. Muller – 2015. – P. 483-486

10. Grus, J. Data Science from Scratch: First Principles with Python, 2nd Edition / J. Grus – 2019.- P. 152-157

11. E. Ameisen. Building Machine Learning Powered Applications: Going from Idea to Product, 1st Edition / E. Ameisen – 2020. – P. 321-322

12. O'Reilly, T., & Campbell, M. Designing Machine Learning Systems, 2019. P. 218-225

13. Treveil M., Omont N., Kenji L. Introducing MLOps / M. Treveil, N. Omont, L. Kenji – 2020.- P.421

14. Peng R., Matsui E. The Art of Data Science / R. Peng, E. Matsui – 2016. – P.208-211

15. Spiegelhalter D. The Art of Statistics: Learning from Data / D. Spiegelhalter – 2019. – P. 81-85

16. Lane, H., Howard, C., & Hapke, H. Natural Language Processing with PyTorch: Build Intelligent Language Applications Using Deep Learning, 1st Edition, 2019, P. 128-140.

17. Tunstall L., Werra L. Transformers in NLP: Building and Training Transformer-Based Language Models for Natural Language Processing / L. Tunstall, L. Werra, 2022.

18. Géron, A. Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems, 1st Edition, 2019, P 110-118.

19. O'Reilly, T., & Campbell, M. Practical MLOps: Operationalizing Machine Learning Models, 2020

20. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding [Electronic resource] / J. Devlin, Ming-Wei Chang, K. Lee, K. Toutanova – 2018.- Mode of access: https://arxiv.org/abs/1810.04805

21. Evolution of transfer learning in natural language processing [Electronic resource] / A. Malte, P. Ratadiya – 2019.- Mode of access: https://arxiv.org/abs/1910.07370

22. M. Ross Sheldon Introduction to Probability and Statistics for Engineers and Scientists, Fifth Edition / Sheldon M. Ross – 2014.- P. 201-203

23. The NLP Cookbook: Modern Recipes for Transformer based Deep Learning Architectures [Electronic resource] / S. Singh, A Mahmood –2021.- Mode of access: https://arxiv.org/abs/2104.10640

24. Ліскін В.О., Бевзюк К.А. Математичне та програмне забезпечення системи семантичного аналізу відгуків на заклади харчування // Прикладна математика та комп'ютинг. ПМК, 2023: п’ятнадцята наук. конф. магістрантів та аспірантів, Київ,

28—30 лист. 2023 р.: зб. тез доп. / [редкол.: Дичка І. А. та ін.]. — К.: Просвіта, 2023. — 680 с. с.28-33 ISBN 978-617-7010-27-1

Додаток А

Лістинг програм

Лістинг файлу app.py

```
import json

from flask import Flask, jsonify, render_template, request
from src.utils import PIPELINE_PATH, clean_text, get_avg_value
from transformers import pipeline

app = Flask(__name__)


@app.route("/")
def new():
    return render_template("index.html")


@app.route("/predict", methods=["POST"])  # /result route
def predict():
    model_pipeline = pipeline(
        task="text-classification",
        model=PIPELINE_PATH,
        use_fast=False,
        framework="pt",
    )

    text = request.form["name"]
    text_splitted = text.split("\n\n")
    if len(text_splitted) > 1:
        text_processed = list(map(clean_text, text_splitted))
        result = model_pipeline(text_processed)
        avg_result = get_avg_value(result)
        result_separate = modify_results(result)
        return jsonify(avg_result=avg_result, individual_results=result_separate)
    else:
```

```
        text_processed = clean_text(text)
        result = model_pipeline(text_processed)
        return jsonify(individual_results=modify_results(result))


def modify_results(model_output: list[dict[str:str]]):
    flatten_output = []
    for elem in model_output:
        flatten_output.append(
            {"stars": elem["label"], "probability": round(elem["score"], 2)}
        )
    return flatten_output


if __name__ == "__main__":
    app.debug = True
    app.run()
```

Лістинг файлу index.html

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>
<title>Flask Sentiment Analysis</title>
<meta name="viewport" content="width=device-width, initial-scale=1">
<!-- Web Application Manifest -->
<link rel="manifest" href='data:application/manifest+json,{ "name": "Flask App", "short_name": "Flask App", "display": "standalone" }' />
<style>
body {font-family: Arial, Helvetica, sans-serif;
background-color:   #e6ecff;}
* {box-sizing: border-box;}
input[type=text], select, textarea {
width: 100%;
padding: 12px;
border: 1px solid #ccc;
border-radius: 4px;
box-sizing: border-box;
margin-top: 6px;
margin-bottom: 16px;
resize: vertical;
}
input[type=submit] {
background-color: #4CAF50;
color: white;
padding: 12px 20px;
border: none;
border-radius: 4px;
cursor: pointer;}
```

```
input[type=submit]:hover {
background-color: #45a049;}
.container {
border-radius: 5px;
background-color:  #f2f2f2;
padding: 20px;
}
</style>
</head>
<body>
<h1 align="center" style="color: #000066">Sentiment analysis </h1>
<div class="container">
<form id="form">
<label for="subject">Input your comments here</label>
<textarea id="name" name="name" placeholder="Write something.." style="height:200px"></textarea>
<input type = "submit" name = "submit"> <!-- submit event -->
</form>
</div>
<h2>Analysis : <span id="polarity"><span></h2>
   <script>
      $("#form").submit(function (e) {
         e.preventDefault();

         // Clear previous results
         $("#polarity").empty();

         $.ajax({
            data: {
               name: $("#name").val()
            },
            type: 'POST',
            url: '/predict'
         }).done(function(data) {
            console.log(data);


            try {
               // Parse the JSON string
               var resultData = data;

               // Append the average result if present
               if (resultData.hasOwnProperty("avg_result")) {
                  $("#polarity").append("<p>Avg result: " + resultData["avg_result"] + " stars</p>");
               }

               // Append individual results
               resultData["individual_results"].forEach(function(result) {
                  $("#polarity").append("<p>Result: " + result["stars"] + ", probability: " + result["probability"] + "</p>");
               });
            } catch (error) {
```

```
            console.error("Error parsing JSON: " + error);
        }
    }).fail(function(jqXHR, textStatus, errorThrown) {
        console.error("AJAX Request Failed: " + textStatus, errorThrown);
            });
        });
        </script>
</body></html>
```

Лістинг файлу model_evaluation.py

```
import pandas as pd
import torch.nn.functional as F
from sklearn.metrics import f1_score
from transformers import (
    BertForSequenceClassification,
    BertTokenizer,
    pipeline,
)

from utils import sentiment_analyser

data = pd.read_csv("data/final/processed.csv")


tokenizer = BertTokenizer.from_pretrained("models/tokenizer")

inputs = tokenizer(data.cleaned_review.loc[11], return_tensors="pt")


print(data.cleaned_review.loc[11])
print(inputs["input_ids"])
# print(tokenizer.decode(inputs["input_ids"][0]))
print(inputs["token_type_ids"])
print(inputs["attention_mask"])


# model_pipeline = pipeline("text-classification", model="models/bert_base_pipeline")

model = BertForSequenceClassification.from_pretrained("models/bert_based_model")


data["sentiment_analysis_results_1"] = data["review"].apply(
    lambda x: sentiment_analyser(x, tokenizer, model)
)

data["sentiment_analysis_results_2"] = data["cleaned_review"].apply(
    lambda x: sentiment_analyser(x, tokenizer, model)
)
```

```
f1_score_one = f1_score(
    data["sentiment_analysis_results_1"], data["rating"], average="micro"
)
f1_score_two = f1_score(
    data["sentiment_analysis_results_2"], data["rating"], average="micro"
)

print(f1_score_one, f1_score_two)


# result = model(inputs)
# print(result)
# print(F.softmax(result.logits))


# print(
#     model_pipeline(
#         [
#             "This restaurant is awesome",
#             # "This restaurant is awful",
#         ]
#     )
# )
```

Лістинг файлу train_model.py

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))


from nltk.tokenize import WordPunctTokenizer
import re
import emoji
from bs4 import BeautifulSoup
import itertools

tok = WordPunctTokenizer()
pat1 = r'@[A-Za-z0-9]+'
pat2 = r'https?://[A-Za-z0-9./]+'
```

```
def tweet_cleaner(text): # ref: https://towardsdatascience.com/another-twitter-sentiment-analysis-bb5b01ebad90
    # removing UTF-8 BOM (Byte Order Mark)
    try:
        text1 = text.decode("utf-8-sig").replace(u"\ufffd", "?") # The UTF-8 BOM is a sequence of bytes (EF BB BF) that allows the reader to identify a file as being encoded in UTF-8
    except:
        text1 = text


    #replace consecutive non-ASCII characters with a space
    text1 = re.sub(r'[^\x00-\x7F]+',' ', text1)

    #remove emojis from tweet
    text2 = emoji_pattern.sub(r'', text1)

    # Remove emoticons
    # text3 = [word for word in text2.split() if word not in emoticons]
    # text3 = " ".join(text3)

    # contradictions and special characters
    # text4 = spl_ch_contra(text3)

    # HTML encoding
    soup = BeautifulSoup(text2, 'lxml') #HTML encoding has not been converted to text, and ended up in text field as ‘&amp’,’&quot’,etc.
    text5 = soup.get_text()

    # removing @ mentions
    text6 = re.sub(pat1, '', text5)

    # Removing URLs
    text7 = re.sub(pat2, '', text6)

    # Removing punctuations
    # text8 = re.sub("[\.\,\!\?\:\;\-\=\(\)\[\]\"\'\%\*\#\@]", " ", text7)

    # Fix misspelled words
    text9 = ''.join(''.join(s)[:2] for _, s in itertools.groupby(text7))# checking that each character should occur not more than 2 times in every word

    # Tokenizing ,change cases & join together to remove unneccessary white spaces
    text9_list = tok.tokenize(text9.lower())
    text10 = (" ".join(text9_list)).strip()

    return text10


# cleaning tweets
train_df['text_cleaned'] = list(map(lambda x:tweet_cleaner(x),train_df['text']) )
```

```
from transformers import BertTokenizer

# Load the BERT tokenizer.
print('Loading BERT tokenizer...')
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased', do_lower_case=True)


# Print the original sentence.
print(' Original: ', sentences[0])

# Print the tweet split into tokens.
print('Tokenized: ', tokenizer.tokenize(sentences[0]))

# Print the tweet mapped to token ids.
print('Token IDs: ', tokenizer.convert_tokens_to_ids(tokenizer.tokenize(sentences[0])))


max_len = 0

# For every sentence...
for sent in sentences:

    # Tokenize the text and add `[CLS]` and `[SEP]` tokens.
    input_ids = tokenizer.encode(sent, add_special_tokens=True)

    # Update the maximum sentence length.
    max_len = max(max_len, len(input_ids))

print('Max sentence length: ', max_len)


# Tokenize all of the sentences and map the tokens to thier word IDs.
input_ids = []
attention_masks = []

# For every sentence...
for sent in sentences:
    # `encode_plus` will:
    #   (1) Tokenize the sentence.
    #   (2) Prepend the `[CLS]` token to the start.
    #   (3) Append the `[SEP]` token to the end.
    #   (4) Map tokens to their IDs.
    #   (5) Pad or truncate the sentence to `max_length`
    #   (6) Create attention masks for [PAD] tokens.
    encoded_dict = tokenizer.encode_plus(
                        sent,                      # Sentence to encode.
                        add_special_tokens = True, # Add '[CLS]' and '[SEP]'
                        max_length = 75,           # Pad & truncate all sentences.
                        pad_to_max_length = True,
                        return_attention_mask = True,   # Construct attn. masks.
```

```
                    return_tensors = 'pt',     # Return pytorch tensors.
                )

    # Add the encoded sentence to the list.
    input_ids.append(encoded_dict['input_ids'])

    # And its attention mask (simply differentiates padding from non-padding).
    attention_masks.append(encoded_dict['attention_mask'])

# Convert the lists into tensors.
input_ids = torch.cat(input_ids, dim=0)
attention_masks = torch.cat(attention_masks, dim=0)
labels = torch.tensor(labels)

# Print sentence 0, now as a list of IDs.
print('Original: ', sentences[1])
print('Token IDs:', input_ids[1])


from torch.utils.data import TensorDataset, random_split

# Combine the training inputs into a TensorDataset.
dataset = TensorDataset(input_ids, attention_masks, labels)

# Create a 90-10 train-validation split.

# Calculate the number of samples to include in each set.
train_size = int(0.9 * len(dataset))
val_size = len(dataset) - train_size

# Divide the dataset by randomly selecting samples.
train_dataset, val_dataset = random_split(dataset, [train_size, val_size])

# print('{:>5,} training samples'.format(train_size))
# print('{:>5,} validation samples'.format(val_size))

# Checking whether the distribution of target is consitent across both the sets
label_temp_list = []
for a,b,c in train_dataset:
  label_temp_list.append(c)

print('{:>5,} training samples'.format(train_size))
print('{:>5,} training samples with real disater tweets'.format(sum(label_temp_list)))


label_temp_list = []
for a,b,c in val_dataset:
  label_temp_list.append(c)

print('{:>5,} validation samples'.format(val_size))
```

```
print('{:>5,} validation samples with real disater tweets'.format(sum(label_temp_list)))

from torch.utils.data import DataLoader, RandomSampler, SequentialSampler

# The DataLoader needs to know our batch size for training, so we specify it
# here. For fine-tuning BERT on a specific task, the authors recommend a batch
# size of 16 or 32.
batch_size = 32

# Create the DataLoaders for our training and validation sets.
# We'll take training samples in random order.
train_dataloader = DataLoader(
            train_dataset,  # The training samples.
            sampler = RandomSampler(train_dataset), # Select batches randomly
            batch_size = batch_size # Trains with this batch size.
        )

# For validation the order doesn't matter, so we'll just read them sequentially.
validation_dataloader = DataLoader(
            val_dataset, # The validation samples.
            sampler = SequentialSampler(val_dataset), # Pull out batches sequentially.
            batch_size = batch_size # Evaluate with this batch size.
        )

from transformers import BertForSequenceClassification, AdamW, BertConfig

# Load BertForSequenceClassification, the pretrained BERT model with a single
# linear classification layer on top.
model = BertForSequenceClassification.from_pretrained(
    "bert-base-uncased", # Use the 12-layer BERT model, with an uncased vocab.
    num_labels = 2, # The number of output labels--2 for binary classification.
                    # You can increase this for multi-class tasks.
    output_attentions = False, # Whether the model returns attentions weights.
    output_hidden_states = False, # Whether the model returns all hidden-states.
)

# Tell pytorch to run this model on the GPU.
model.cuda()

# Note: AdamW is a class from the huggingface library (as opposed to pytorch)
# I believe the 'W' stands for 'Weight Decay fix"
optimizer = AdamW(model.parameters(),
                  lr = 5e-5, # args.learning_rate - default is 5e-5
                  eps = 1e-8 # args.adam_epsilon  - default is 1e-8.
                )


from transformers import get_linear_schedule_with_warmup

# Number of training epochs. The BERT authors recommend between 2 and 4.
```

```
# We chose to run for 2,I have already seen that the model starts overfitting beyound 2 epochs
epochs = 2

# Total number of training steps is [number of batches] x [number of epochs].
# (Note that this is not the same as the number of training samples).
total_steps = len(train_dataloader) * epochs

# Create the learning rate scheduler.
scheduler = get_linear_schedule_with_warmup(optimizer,
                                            num_warmup_steps = 0, # Default value in run_glue.py
                                            num_training_steps = total_steps)



import random
import numpy as np

# This training code is based on the `run_glue.py` script here:
# https://github.com/huggingface/transformers/blob/5bfcd0485ece086ebcbed2d008813037968a9e58/examples/run_glue.py#L128

# Set the seed value all over the place to make this reproducible.
seed_val = 66

random.seed(seed_val)
np.random.seed(seed_val)
torch.manual_seed(seed_val)
torch.cuda.manual_seed_all(seed_val)

# We'll store a number of quantities such as training and validation loss,
# validation accuracy, and timings.
training_stats = []

# Measure the total training time for the whole run.
total_t0 = time.time()

# For each epoch...
for epoch_i in range(0, epochs):

    # ========================================
    #               Training
    # ========================================

    # Perform one full pass over the training set.

    print("")
    print('======== Epoch {:} / {:} ========'.format(epoch_i + 1, epochs))
    print('Training...')

    # Measure how long the training epoch takes.
    t0 = time.time()
```

```
# Reset the total loss for this epoch.
total_train_loss = 0

# Put the model into training mode. Don't be mislead--the call to
# `train` just changes the *mode*, it doesn't *perform* the training.
# `dropout` and `batchnorm` layers behave differently during training
# vs. test (source: https://stackoverflow.com/questions/51433378/what-does-model-train-do-in-pytorch)
model.train()

# For each batch of training data...
for step, batch in enumerate(train_dataloader):

    # Progress update every 40 batches.
    if step % 40 == 0 and not step == 0:
        # Calculate elapsed time in minutes.
        elapsed = format_time(time.time() - t0)

        # Report progress.
        print('  Batch {:>5,}  of  {:>5,}.    Elapsed: {:}.'.format(step, len(train_dataloader), elapsed))

    # Unpack this training batch from our dataloader.
    #
    # As we unpack the batch, we'll also copy each tensor to the GPU using the
    # `to` method.
    #
    # `batch` contains three pytorch tensors:
    #   [0]: input ids
    #   [1]: attention masks
    #   [2]: labels
    b_input_ids = batch[0].to(device)
    b_input_mask = batch[1].to(device)
    b_labels = batch[2].to(device)

    # Always clear any previously calculated gradients before performing a
    # backward pass. PyTorch doesn't do this automatically because
    # accumulating the gradients is "convenient while training RNNs".
    # (source: https://stackoverflow.com/questions/48001598/why-do-we-need-to-call-zero-grad-in-pytorch)
    model.zero_grad()

    # Perform a forward pass (evaluate the model on this training batch).
    # The documentation for this `model` function is here:
    # https://huggingface.co/transformers/v2.2.0/model_doc/bert.html#transformers.BertForSequenceClassification
    # It returns different numbers of parameters depending on what arguments
    # are given and what flags are set. For our usage here, it returns
    # the loss (because we provided labels) and the "logits"--the model
    # outputs prior to activation.
    loss, logits = model(b_input_ids,
                         token_type_ids=None,
                         attention_mask=b_input_mask,
```

```
                    labels=b_labels)

    # Accumulate the training loss over all of the batches so that we can
    # calculate the average loss at the end. `loss` is a Tensor containing a
    # single value; the `.item()` function just returns the Python value
    # from the tensor.
    total_train_loss += loss.item()

    # Perform a backward pass to calculate the gradients.
    loss.backward()

    # Clip the norm of the gradients to 1.0.
    # This is to help prevent the "exploding gradients" problem.
    torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)

    # Update parameters and take a step using the computed gradient.
    # The optimizer dictates the "update rule"--how the parameters are
    # modified based on their gradients, the learning rate, etc.
    optimizer.step()

    # Update the learning rate.
    scheduler.step()

# Calculate the average loss over all of the batches.
avg_train_loss = total_train_loss / len(train_dataloader)

# Measure how long this epoch took.
training_time = format_time(time.time() - t0)

print("")
print("  Average training loss: {0:.2f}".format(avg_train_loss))
print("  Training epcoh took: {:}".format(training_time))

# ========================================
#               Validation
# ========================================
# After the completion of each training epoch, measure our performance on
# our validation set.

print("")
print("Running Validation...")

t0 = time.time()

# Put the model in evaluation mode--the dropout layers behave differently
# during evaluation.
model.eval()

# Tracking variables
total_eval_accuracy = 0
```

```
total_eval_loss = 0
nb_eval_steps = 0

# Evaluate data for one epoch
for batch in validation_dataloader:

    # Unpack this training batch from our dataloader.
    #
    # As we unpack the batch, we'll also copy each tensor to the GPU using
    # the `to` method.
    #
    # `batch` contains three pytorch tensors:
    #   [0]: input ids
    #   [1]: attention masks
    #   [2]: labels
    b_input_ids = batch[0].to(device)
    b_input_mask = batch[1].to(device)
    b_labels = batch[2].to(device)

    # Tell pytorch not to bother with constructing the compute graph during
    # the forward pass, since this is only needed for backprop (training).
    with torch.no_grad():

        # Forward pass, calculate logit predictions.
        # token_type_ids is the same as the "segment ids", which
        # differentiates sentence 1 and 2 in 2-sentence tasks.
        # Get the "logits" output by the model. The "logits" are the output
        # values prior to applying an activation function like the softmax.
        (loss, logits) = model(b_input_ids,
                               token_type_ids=None,
                               attention_mask=b_input_mask,
                               labels=b_labels)

    # Accumulate the validation loss.
    total_eval_loss += loss.item()

    # Move logits and labels to CPU
    logits = logits.detach().cpu().numpy()
    label_ids = b_labels.to('cpu').numpy()

    # Calculate the accuracy for this batch of test sentences, and
    # accumulate it over all batches.
    total_eval_accuracy += flat_accuracy(logits, label_ids)


# Report the final accuracy for this validation run.
avg_val_accuracy = total_eval_accuracy / len(validation_dataloader)
print("  Accuracy: {0:.2f}".format(avg_val_accuracy))

# Calculate the average loss over all of the batches.
```

```
    avg_val_loss = total_eval_loss / len(validation_dataloader)

    # Measure how long the validation run took.
    validation_time = format_time(time.time() - t0)

    print("  Validation Loss: {0:.2f}".format(avg_val_loss))
    print("  Validation took: {:}".format(validation_time))

    # Record all statistics from this epoch.
    training_stats.append(
        {
            'epoch': epoch_i + 1,
            'Training Loss': avg_train_loss,
            'Valid. Loss': avg_val_loss,
            'Valid. Accur.': avg_val_accuracy,
            'Training Time': training_time,
            'Validation Time': validation_time
        }
    )

print("")
print("Training complete!")

print("Total training took {:} (h:mm:ss)".format(format_time(time.time()-total_t0)))
```

Додаток Б

Ілюстративний матеріал

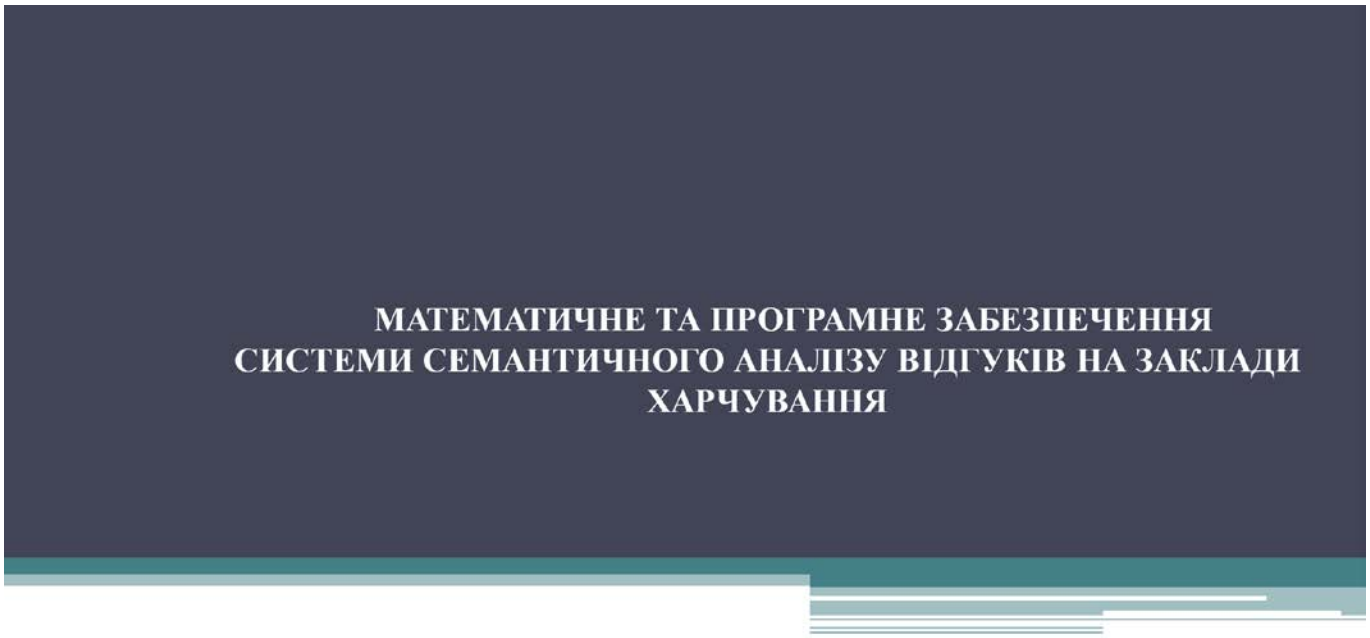


Рисунок Б.1 – Слайд 1

Актуальність теми

В сучасному світі, де доступ до інформації легко здійснюється через Інтернет та соціальні мережі, відгуки та рецензії на заклади харчування стали важливим джерелом інформації для споживачів. Оцінка ресторанів, кав'ярень, закладів швидкого харчування часто здійснюється за допомогою різноманітних платформ та додатків, які дозволяють користувачам залишати свої коментарі та оцінки. Однак велика кількість відгуків може бути важкою для аналізу, особливо коли мова йде про великі міські області, де кількість закладів харчування вражає своєю розмаїтістю.

Рисунок Б.2 – Слайд 2

Мета дослідження

Семантичний аналіз відгуків на заклади харчування, в якості прикладу буде взято кав'ярню «Starbucks».

Рисунок Б.3 – Слайд 3

Об'єкт та предмет дослідження

- Об'єкт дослідження: розробка, підбір параметрів та навчання моделі, яка зможе класифікувати великі обсяги текстових даних.

- Предмет дослідження: математичне та програмне забезпечення системи класифікації відгуків на заклади харчування за допомогою нейронних мереж на базі різних архітектур, це дозволить дати кількісну оцінку тексту і може бути використано для покращення якості обслуговування та задоволення потреб клієнтів у сфері гастрономії та гостинності.

Рисунок Б.1 – Слайд 4

Задачі дослідження

- Виконати порівняльний аналіз існуючих архітектур нейронних мереж для роботи з текстом.
- Дослідити методи обробки тексту для подальшої передачі у нейронну мережу
- Обрати найкраще рішення для виконання поставленої задачі, оптимізувати гіперпараметри нейронної мережі.
- Протестувати створену нейронну мережу на тренувальних данних
- Створити графічний інтерфейс для роботи з програмою

Рисунок Б.5 – Слайд 5

Існуючі рішення

- Рекурентні нейронні мережі
- Рекурентні нейронні мереж на основі LSTM шарів
- Нейронні мережі на основі трансформерів

Рисунок Б.6 – Слайд 6

Існуючі рішення. Архітектура на базі рекурентних нейронних мереж

Переваги:

- LSTM (Long Short-Term Memory) шари розроблені для ефективного управління довгостроковими залежностями в данних. Це особливо важливо в задачах, де важлива взаємодія між далекими елементами послідовності.
- LSTM має механізми входу та виходу, які дозволяють моделі вибирати, яку інформацію забути та яку зберегти. Це дозволяє ефективно працювати з різнорідними даними та уникати втрати важливої інформації.
- LSTM може ефективно працювати з послідовностями різної довжини, завдяки своїй здатності адаптуватися до входів різної довжини та вираховувати їхні залежності.

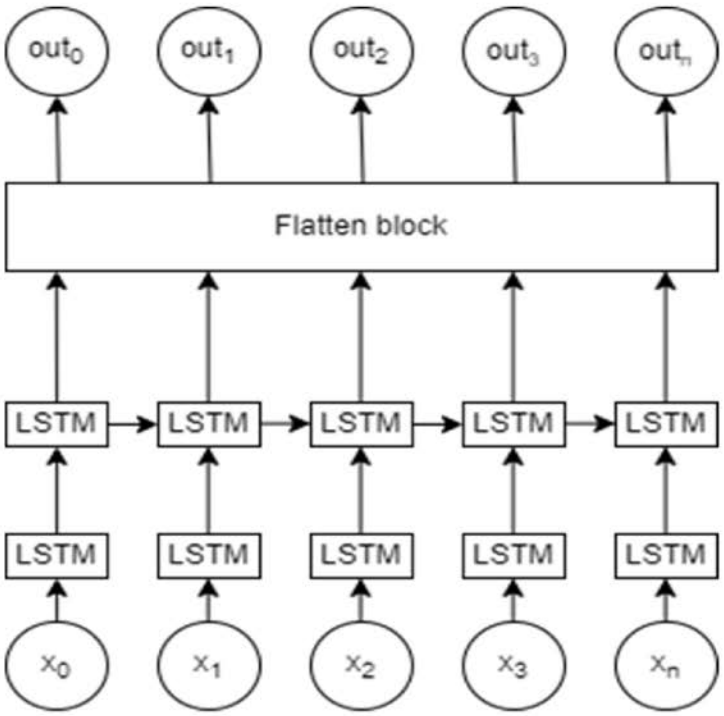
Рисунок Б.7 – Слайд 7

Існуючі рішення. Архітектура на базі рекурентних нейронних мереж

Недоліки:

- LSTM модель може бути вимогливим до обчислювань, що робить його менш практичним для використання в ресурсозмінних або обмежених обчислювальних середовищах.
- Моделі потребують великої кількості даних для ефективного навчання, особливо в умовах складних завдань, щоб уникнути перенавчання.
- Добре підібрані параметри є ключовим фактором для успішної роботи LSTM моделі. Невірно вибрані параметри можуть призвести до повільного навчання або недооцінення важливості деяких залежностей у данних

Рисунок Б.8 – Слайд 8



Типовий приклад моделі на базі LSTM шарів

Рисунок Б.9 – Слайд 9

Існуючі рішення. Архітектура на базі трансформерів

Переваги:

- Трансформери мають паралельне обчислення для кожного слова або токена в послідовності. Це робить їх ефективними для обробки довгих послідовностей та прискорює процес тренування.
- Механізм уваги в трансформерах дозволяє моделі приділяти різним частинам вхідної послідовності різний рівень уваги.
- Трансформери підходять для великих даних та мають високий рівень масштабованості.

Рисунок Б.10 – Слайд 10

Існуючі рішення. Архітектура на базі трансформерів

Недоліки:

- Трансформери можуть бути обчислювально витратними, особливо при роботі з великими моделями та даними.
- Для досягнення найкращих результатів трансформери часто потребують великої кількості тренувальних даних, щоб уникнути перенавчання та покращити узагальнюючу здатність.

Рисунок Б.11 – Слайд 11

Існуючі рішення. Архітектура на базі трансформерів

Трансформер блок

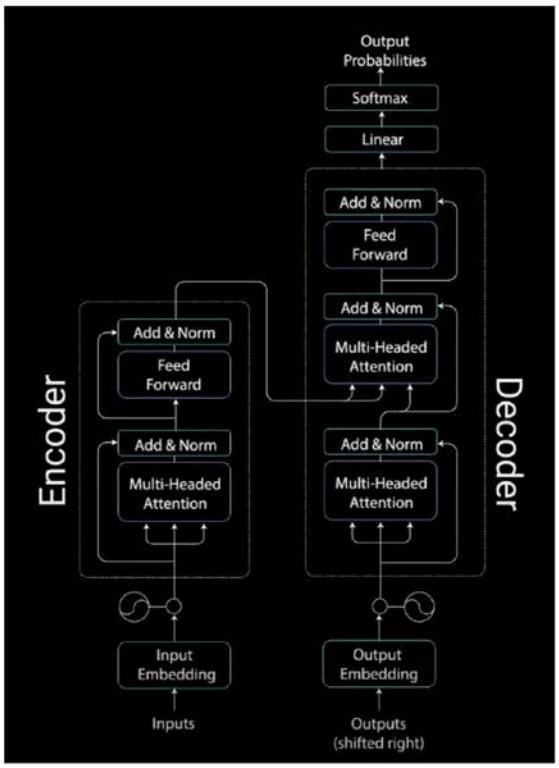


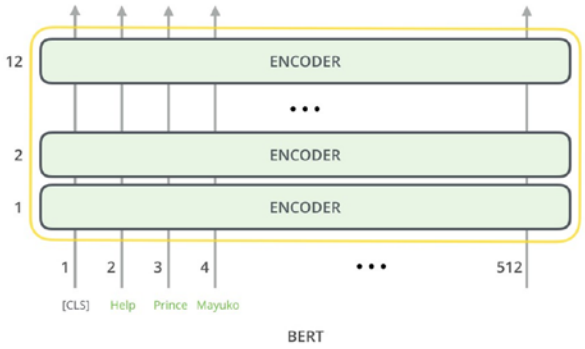
Рисунок Б.12 – Слайд 12

Оцінка ефективності запропонованих рішень

Methods	Accuracy	Precision	Recall	F1 score
RNN	0.86	0.85	0.75	0.7969
LSTM	0.87	0.83	0.88	0.854
Transfoemer	93	0.904	0.91	0.907

Рисунок Б.13 – Слайд 13

Запропоноване рішення. Архітектура BERT



Архітектура BERT складається з кількох величезних багатоярусних трансформерних енкодерів. Кожен трансформерний кодер включає два підшари: шар самоуваги та шар передачі зворотнього зв'язку. Ключовим технічним досягненням BERT є застосування двохстороннього тренування трансформера, популярної моделі уваги, для мовного моделювання.

Рисунок Б.14 – Слайд 14

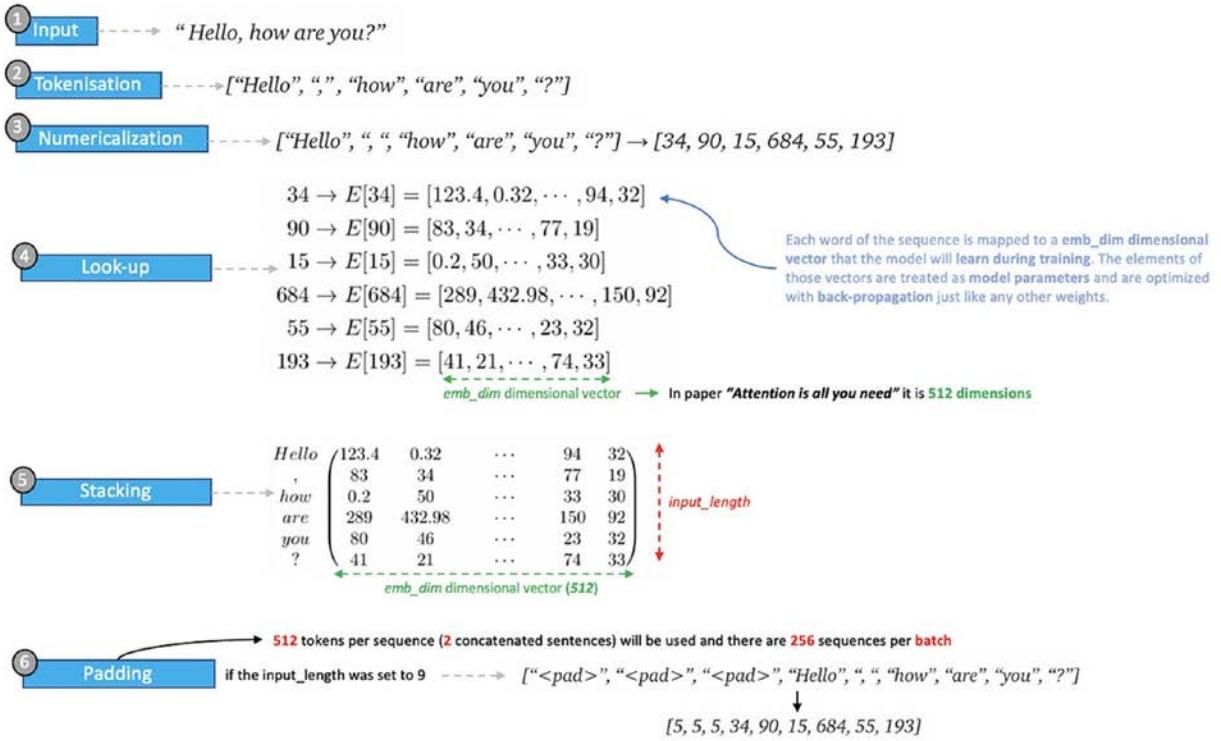
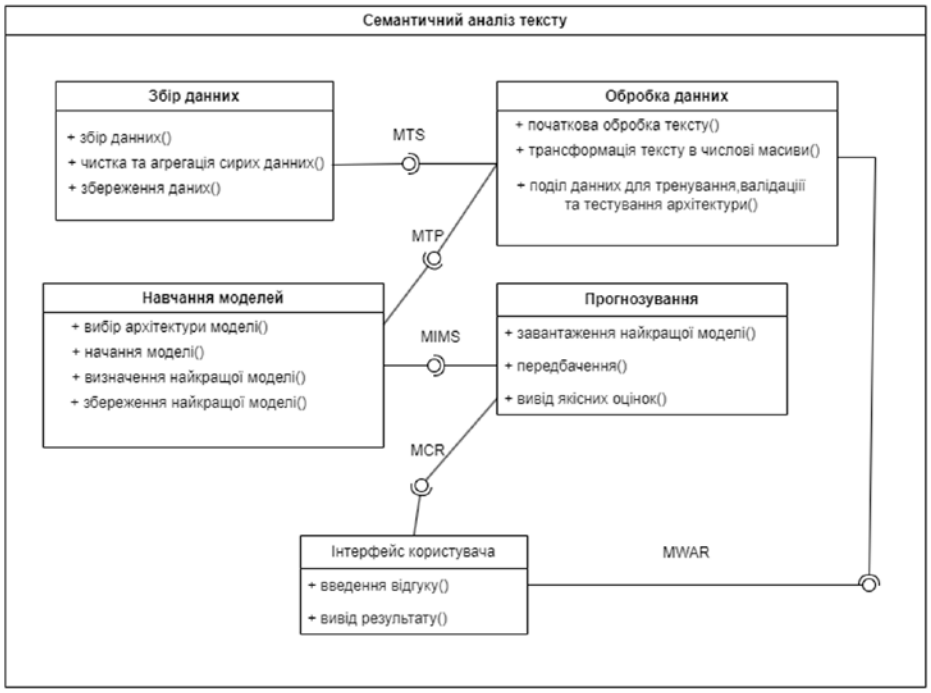


Рисунок Б.15 – Слайд 15

Реалізація у вигляді окремого спеціалізованого програмного засобу



Діаграма компонентів в UML нотації

Рисунок Б.16 – Слайд 16

Висновки

В даній роботі здійснено огляд існуючих рішень в різних джерелах інформації, пошук оптимального методу для обробки непідготовлених даних та вирішення поставленої задачі.

Зроблено порівняльний аналіз різних архітектур нейронних мереж та обрано найкращу архітектуру за точністю метрики (f1-score).

Перспективи подальших досліджень спрямовані на вдосконалення MLOps процесів для автоматизації версіювання моделей та даних. Також планується розгорнути веб-застосунок на хмарному сховищі.

Рисунок Б.17 – Слайд 17