МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики
(повне найменування назва факультету)

Кафедра програмування
(повна назва кафедри)

ДИПЛОМНА РОБОТА

Аналіз тональності тексту за допомогою нейронних мереж, на прикладі пошуку фейкових новин про російсько-українську війну

Виконала: студентка групи ПМІ-41
спеціальності 122 – комп'ютерні науки
(шифр і назва спеціальності)

Чорна М.І
(підпис) (прізвище та ініціали)

Керівник Рикалюк Р.Є
(підпис) (прізвище та ініціали)

Рецензент
(підпис) (прізвище та ініціали)

2023

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики

Кафедра програмування

Спеціальність 122 Комп'ютерні науки

«ЗАТВЕРДЖУЮ»
Завідувач кафедри Ярошко С.А.

"_ "_____ 2022 року

З А В Д А Н Н Я

НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Чорній Марті Ігорівні
(прізвище, ім'я, по батькові)

1. Тема роботи «Аналіз тональності тексту за допомогою нейронних мереж, на прикладі пошуку фейкових новин про російсько-українську війну»

керівник роботи Рикалюк Роман Євстахович, доцент, к. ф.-м. н. ,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені Вченою радою факультету від "13"_вересня 2022 року №_15_.

2. Строк подання студентом роботи 12 червня 2023 року

3. Вихідні дані до роботи літературні джерела, інтернет-ресурси, постановка задачі, наукові статті

4. Зміст дипломної роботи (перелік питань, які потрібно розробити)

1. Вивчити основні підходи та методи аналізу тональності тексту;

2. Розробити архітектуру нейронної мережі для аналізу тональності тексту;

3. Зібрати та підготувати відповідний корпус даних, що містить новини про російсько-українську війну;

4. Провести оцінку ефективності навченої моделі

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

_____

_____

6. Консультанти розділів роботи

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

7. Дата видачі завдання_____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	**Назва етапів магістерської роботи**	**Строк виконання етапів роботи**	**Примітка**
1	Ознайомлення з предметною областю	05.09.2022 – 09.09.2022	
2	Аналіз аналогів	09.09.2022 – 13.09.2022	
3	Написання специфікації вимог	13.09.2022 – 15.09.2022	
4	Вибір технологій	16.09.2022 – 18.09.2022	
5	Проектування архітектури ПЗ	19.09.2022 – 20.09.2022	

6	Пошук новинних даних	21.09.2022 – 10.03.2022	
7	Розробка нейронної мережі	10.03.2023 – 10.04.2023	
8	Тестування програми	01.05.2023 – 15.05.2023	
9	Оформлення магістерської роботи	20.05.2023 – 06.06.2023	
10	Подання магістерської роботи	12.06.2023	

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

Керівник роботи _____ Рикалюк Р.Є.
(підпис) (прізвище та ініціали)

Зміст

АНОТАЦІЯ

Дана дипломна робота присвячена розробці та застосуванню нейронної мережі BERT для аналізу тональності тексту з метою виявлення фейкових новин, пов'язаних з російсько-українською війною.

У роботі проведено детальний огляд літератури щодо аналізу тональності тексту та нейронних мереж, зокрема моделі BERT. Використання цієї нейронної мережі дозволяє отримати векторні представлення слів та фраз, що покращує точність аналізу тональності.

Для розробки моделі проведено попередню обробку текстових даних та підготовку тренувального набору даних, що включає пошук та аналіз фейкових новини про російсько-українську війну. Модель BERT навчено на цьому наборі даних та проведено експерименти для визначення оптимальних параметрів моделі.

У другому розділі детально розглянуто методи обробки природньої мови та теоретичну частину аналізу тональності тексту.

У третьому розділі описано роботу будову нейронних мереж та детальніше описані моделі, які застосовуються в аналізі тональності.

Реалізація нейронної мережі BERT, її архітектура і методи тренування розглянуті в четвертому розділі.

У п’ятому розділі аналіз результатів роботи нейронної мережі.

Загальний обсяг роботи – 50 сторінок.

ВСТУП

1.1. Актуальність теми

З появою Інтернету ділитись своїми думками стало врази легше, і появилась можливість обговорювати якусь ситуацію чи предмет, будучи в різних місцях. Тому можна помітити тенденцію, наприклад, в інтернет-магазинах, що товари з більшою кількістю відгуків, продаються частіше, ніж ті самі новинки.

Вплив таких коментарів на питання у дискусіях чи на прийняття рішень, змушують звернути увагу на таку онлайн активність. Почався активний період дослідження методів обробки природної мови (Natural Language Processing).

Аналіз тональності є однією із актуальних задач у сфері обробки природних мов. Однією з можливостей, яку надає нам такий аналіз текстів, є визначення емоційного стану читача. Це актуально і потрібно для таких сфер як маркетинг, політика, соціологія, тощо.

Однак зараз найбільш актуальним полем для аналізу є новини. Особливо актуально буде дослідження новин на їхню проросійкість, таким чином визначаючи надійність інформації. Відомо, що деяким людям властиво не перевіряти джерело інформації, і що вони можуть вірити їй без жодних сумнівів. Таким чином, можуть формуватися антиукраїнські настрої. Тому важливо мати можливість класифікувати новини на позитивну, негативну та нейтральну емоційність щодо України.

1.2. Методи дослідження

Для досягнення мети дослідження, я використовую наступні методи:

- аналіз літератури, досліджень та статистичних даних з області аналізу тональності тексту;
- використання бібліотек машинного навчання Python, зокрема TensorFlow та Keras, для побудови та навчання моделі нейронної мережі;

- використання засобів обробки природних мов Python для попередньої обробки тексту.

1.3. Мета та завдання дослідження

Таким чином, метою дослідження є створення нейронної мережі для класифікації новин про Україну на позитивну, негативну та нейтральну емоційність на основі аналізу їх тональності. Описати принципи її роботи та практично реалізувати її.

Для досягнення поставленої мети необхідно виконати такі завдання:

- проаналізувати методи та рішення у галузі визначення тональності текстів;
- розробити програмний продукт, що втілює запропонований метод.

Об’єкт дослідження – методи розпізнавання тональності текстів природної мови.

2. ТЕОРЕТИЧНІ АСПЕКТИ АНАЛІЗУ ТОНАЛЬНОСТІ ТЕКСТУ

2.1. Обробка природної мови

Обробка природної мови (Natural Language Processing, NLP) – це розділ комп'ютерної лінгвістики, який вивчає і розробляє методи та інструменти для обробки й аналізу людської мови. Мета обробки природної мови полягає в тому, щоб дозволити комп'ютерам розуміти, інтерпретувати та створювати людську мову, яка включає як письмовий текст, так і усне мовлення.

Мовне моделювання, машинний переклад, аналіз тональності тексту, вилучення інформації та системи запитань-відповідей — усе це приклади завдань NLP.

Основна складність обробки природної мови полягає в тому, що людська мова складна, неоднозначна і постійно розвивається. Природна мова відрізняється різноманітністю лінгвістичних явищ, таких як синтаксис, семантика та дискурс, що ускладнює розробку комп'ютерних алгоритмів, здатних точно і ефективно розуміти та обробляти мову.

Одним із важливих підходів до NLP є системи, засновані на правилах, які використовують набір правил для аналізу та створення мови. Такі системи корисні для простих і чітко визначених завдань, таких як перевірка орфографії та виправлення граматики. Однак через складність і варіативність мови, а також через складність охоплення всіх нюансів і винятків у природній мові такі системи можуть бути обмеженими.

2.2. Аналіз методів обробки природної мови

Для досягнення різних цілей використовують безліч методик та технік з обробки природної мови. Вони використовуються в багатьох галузях, наприклад,

машинний переклад, розпізнавання мови, класифікація текстів, аналіз тональності тощо.

Одним з таких методів є токенізація. Токенізація (Tokenization) — це процес поділу тексту на окремі слова чи фрази, які називаються токенами. Багато завдань NLP передбачає видалення знаків пунктуації, спеціальних символів та інших не алфавітних символів.

Наступний спосіб це позначення частин мови. Позначення частин мови (Part-of-Speech, POS) — це процес присвоєння граматичної категорії кожній лексемі, такій як іменник, дієслово, прикметник тощо. Це важливо в багатьох завданнях NLP, таких як аналіз тональності і класифікація тексту, де значення речення може змінюватися залежно від граматичної категорії вживаних слів.

Ще одна техніка це стемінг. Стемінг (Stemming) - це процес видалення закінчень слів для отримання кореневих (stem) форм слів. Наприклад, слова "running", "runs", "runned" будуть перетворені на кореневу форму "run". Оскільки однокореневі слова сприймаються як одне слово, це зручно для подальшого опрацювання та аналізу тексту. Для виконання стемінгу можна використовувати різні алгоритми. Найбільш відомим і часто використовуваним алгоритмом є алгоритм Портера (Porter stemming algorithm). Цей алгоритм використовує правила для видалення закінчень слів та збереження кореневих форм. Однак існують і інші алгоритми, такі як Snowball stemming algorithm, Lovins stemming algorithm та інші.

Подібний до стемінгу підхід – лематизація. Лематизація (lemmatization) - це процес приведення слів до їх базових форм (лем). Оскільки лематизація враховує не лише закінчення слова, але також його контекст і семантику, це більш точний метод, ніж стемінг. Наприклад, слово "dogs" буде лематизоване до слова "dog", а слово "better" до слова "good". Лематизація може допомогти уникнути недоліків стемінгу, таких як втрата семантичної інформації та інші. Для цього використовуються словники, які містять базові форми слів та правила перетворення слів на ці базові форми. Для реалізації цього підходу можна використовувати такі інструменти, як NLTK (Natural Language Toolkit), spaCy, Stanford CoreNLP та інші.

2.3. Аналіз тональності тексту

Аналіз тональності тексту (англ. Sentiment analysis) – це клас методів аналізу контенту в комп'ютерній лінгвістиці, що призначений для автоматичного виявлення емоційно забарвленої лексики та оцінки емоцій автора тексту стосовно об'єктів, про які йдеться у тексті.

В аналізі тональності тексту, сам текст можна поділити на два типи: факти і думки. Факти – це об’єктивний вираз про щось, коли думки є суб’єктивними висловлюваннями, які описують почуття й оцінки. Ключовим є визначення думки. Їх можна поділити на два типи:

- проста(пряма) думка;
- порівняння.

Пряма думка містить висновок автора про суб’єкт безпосередньо. Також проста думка може бути виражена явно чи неявно. Наприклад, «Якість зображення телефону А погана» і «Після довгого періоду занепаду, ситуація в країні покращилась». Отже, пряму думку можемо визначити формально – це кортеж, який складається з 5 елементів(entity, feature, sentiment value, holder, time). Тобто, у реченні автор(holder) висловив думку про певну ознаку(feature) об’єкту(entity) в момент часу(time), надавши певного виду емоції(sentiment value) : позитивну, негативну чи нейтральну.

Думку-порівняння можна розділити на три типи:

- порівняння ознак об’єктів на користь один-одного (non-equal gradable) – наприклад, «Ціни в магазині «А» нижчі, ніж в магазині «В»;
- порівняння ознак різних об’єктів на схожість(equative) – наприклад, «Ціни в магазині «А» і «В» майже однакові»;

- перевага одного об’єкта над іншим(superlative) – наприклад, «Магазин «А» кращий за магазин «В».

Другий тип думки можна визначити як кортеж (Obj1, Obj2, A, holder, time). Тобто, у реченні автор(holder) порівняв множину об’єктів(Obj1, Obj2) за певною ознакою(A) в момент часу(time). Можна помітити різницю між простою думкою і думкою порівняння – друга думка не має емоційної оцінки автора.

Отже сентимент-аналіз включає в себе такі завдання:

- визначення наявності емоційного забарвлення;
- визначення полярності;
- вилучення ознак з емоційно забарвленого тексту.

Визначення полярності тексту розглядаємо на декількох рівнях:

- рівень документу – класифікація повністю всього документа, чи є він відображенням позитивної чи негативної думки; кожен документ висловлює думку як єдину сутність; коментарі до новин відносяться до цієї категорії;
- рівень речення – аналіз чи речення висловлює в цілому позитивну чи негативну думку;
- рівень сутності чи ознаки – виконує більш детальний аналіз, ніж попередні два; аналіз об’єкту думки допомагає глибше зрозуміти проблему аналізу тональності; можна отримати структурований підсумок думок щодо властивостей об’єкту.

Найбільшим індикатором емоційності тексту є слова емоційного забарвлення (sentiment words), які використовуються для висловлення позитивної чи негативної думки. Наприклад, «чудово», «неймовірно», «добре» – слова, що висловлюють позитивну думку, але слова «жахливо», «поганий», «безглуздо» – негативну. Набір таких слів та ідіом називають лексичним словником. Хоч такий словник є дуже важливим елементом сентимент аналізу тексту, просто його використання не буде ефективним.

При використанні лексичного словника можна натрапити на такі проблеми:

- слово може приймати різні відтінки при використанні в іншій предметній області. Наприклад, «Фільм має передбачуваний сюжет» – негативна оцінка, а «Алгоритм виконав передбачувану дію» – позитивна;
- неологізми та помилки у слові. В інтернеті це дуже поширене явище, і неможливо проконтролювати всі нові слова, які створюються та як вони пишуться;
- нейтральний сентимент. Цей феномен властивий для двох типів речення: питального і умовного. Наприклад, «Чи є цей товар найкращим?» та «Якщо я захочу купити хороший телефон, я піду в магазин». В цих реченнях ми можемо помітити слова, які описують позитивний настрій, але жодне з них не висловлює конкретної думки щодо сутності;
- сарказм. Речення, які містять сарказм є дуже складними для аналізу. Наприклад, «Який прекрасний телефон! Перестав працювати вже наступного дня».

Прості думки ми також можемо поділити на явні (explicit) та неявні (implicit) думки. Явна думка – думка, яка висловлена безпосередньо щодо самої сутності або ознаки сутності. Неявна думка – це думка, що виражає неявно ознаку сутності або саму сутність. Наприклад, думка «Після цього препарату, я почуваюсь гірше» описує негативну думку щодо сутності «препарат», бо ця сутність погано впливає на сутність «самопочуття». Також ці категорії можна помітити і у порівняльних думках. Явні думки легше виявити та класифікувати, тому більшість сучасних досліджень фокусуються на них.

Ще одна важлива класифікація, це класифікація суб’єктивності думки. Об’єктивне висловлювання виражає певний факт, який є загальновідомим. Суб’єктивне речення висловлює почуття та думки окремої людини. Останні можуть виступати у таких формах, як думки, твердження, бажання, переконання та

інші. На жаль, існує проблема як відрізнити суб’єктивну думку від просто висловлення думки. Під висловлюванням ми маємо розуміти, що речення чи документ висловлює позитивну чи негативну думку. Що потрібно пам’ятати:

- суб’єктивне речення може не висловлювати ніякої думки. Наприклад, «Я думаю, він пішов додому»;
- об’єктивні речення можуть висловлювати думку на основі ствердження фактів. Наприклад, «Не дивлячись на те, що комп’ютер новий, він перестав працювати через місяць».

2.3.1. Процес аналізу тональності

Через те що аналіз тексту не є простим завданням важливо дотримуватись певних кроків для найкращих результатів.

Перший крок це збір даних. Перед початком аналізу потрібно зібрати достатню кількість даних, які, наприклад, в подальшому будуть використовуватись в тренуванні нейронних мереж. Це можуть бути тексти з наукових статей, форумів, соціальних мереж – всюди, де люди можуть висловити свою думку.

Далі відбувається надзвичайно важливий етап, від якого залежить точність аналізу – попередня обробка тексту. Цей етап може включати в себе токенізацію, лематизацію та видалення знаків пунктуації, стоп-слів. Цей пункт дозволяє представити всі слова в однаковому вигляді, чим поліпшує якість аналізу.

Для правильної оцінки емоційності слів, потрібно мати якесь джерело вже готових визначень, так званих словників. Ці словники можуть збиратись вручну, або використовувати вже готові дані. Цей етап надзвичайно важливий, якщо використовувати нейронні мережі для аналізу тональності, оскільки комп’ютер якраз і навчається на основі цих словників.

Після підготовки тексту та складання або вибору готового словника позитивних і негативних слів необхідно оцінити тональність кожного слова в тексті. Це може бути зроблено за допомогою двох підходів: лексичного аналізу та статистичного аналізу.

Кожному слову при лексичному розборі присвоюється значення на основі його тональності, яка описана в словнику. Якщо слово «happy», наприклад, є в словнику позитивних термінів, йому буде надано позитивне значення. У результаті кожне слово в тексті можна класифікувати як позитивне, негативне чи нейтральне.

У статистичному аналізі тональність слова визначається його використанням у контексті тексту. Часто використовується алгоритм машинного навчання, який вивчає набір текстів із відомою тональністю та виводить значення тональності кожного слова залежно від його контексту.

3. НЕЙРОННІ МЕРЕЖІ

Використання нейронних мереж для аналізу емоційності тексту може бути дуже ефективним інструментом для розуміння того, які емоції він викликає у людей. Зазвичай, для розпізнавання емоцій в реченнях використовуються навчені моделі, які мають здатність класифікувати текст за певними емоційними категоріями, такими як радість, сум, гнів, страх тощо.

3.1. Будова нейронних мереж

Штучні нейронні мережі побудовані за принципом природних. Штучний нейрон - це функція, що має нелінійний характер, де аргументом є лінійна комбінація всіх вхідних сигналів. Він складається з нелінійного перетворювача та суматора, тож всі нейрони є простими і однотипними елементами. Також нейрон характеризується своєю функцією активації. Функція активації – це функція залежності вихідного сигналу від вхідного, зазвичай обирають монотонно зростаючу функцію, інколи обирають диференційовану функцію.

Три найбільш відомі функції активації:

- Сигмоїдальна функція.
- Гіперболічний тангенс.
- Функція одиночний стрибок.

Персептрон – тип штучного нейрона, що розробив Френк Розенблат. Нейрон приймає на вхід значення x_1 , x_2 ,…,x_n і видає бінарний результат. Один із піонерів у цій галузі, Розенблат, запропонував використовувати ваги – числа, що відображають важливість кожного вхідного сигналу. Зважена сума NET (або ваги) порівнюється з граничним значенням (T – поріг), і на основі цього визначається яким буде результат – 0 чи 1.

Оскільки сигмоїдальні нейрони схожі на парсептрони, це дозволяє такій мережі навчатись. Важливою властивістю сигмоїдальної функції є її диференційованість. Застосування неперервної функції активації дозволяє використовувати при навчанні градієнтні метод.

3.2. Навчання нейронної мережі

Навчання нейронної мережі – це процес налаштування архітектури мережі (структури зв’язків між нейронами) і ваг синаптичних зв’язків для ефективного вирішення поставленої задачі. Протягом процесу навчання, мережа повинна правильно реагувати на вхідні дані.

Виділяють три типи навчання:

- Навчання без вчителя (Unsupervised Learning) – у такому навчанні, дані не розмічені, і машина сама намагається знайти закономірності. На практиці такі алгоритми використовують рідше, зазвичай як методи аналізу та підготовки даних, а не як основний алгоритм, що вирішує конкретні завдання за допомогою цих даних. У реальності добре розмічені дані – це велика рідкість, тому для їх розмітки зазвичай використовують або спеціальні сервіси, або спеціальні алгоритми для розмітки (які, в свою чергу, можуть також використовувати машинне навчання)[15].
- Навчання з вчителем (Supervised Learning) – у цьому випадку машина має "наставника" - учителя, який навчає її правильним діям. Учитель передбачає всі необхідні дані, за допомогою яких машина навчається на конкретних прикладах. В термінах машинного навчання, учитель відображає втручання людини в процес обробки інформації. З вчителем машина навчається краще і швидше, тому такі алгоритми широко використовуються для вирішення практичних завдань. До алгоритмів навчання з вчителем належать регресія і класифікація, які вирішують різні типи завдань..
- Навчання з підкріпленням (Reinforcement Learning) – використовується для ситуацій, де потрібно вижити в реальному середовищі, а не аналізувати дані.

Середовище може бути різним: реальний світ, симуляція або навіть комп'ютерна гра. Задача таких систем - мінімізувати помилки або максимізувати вигоду, а не розраховувати кожен крок. Навчання з підкріпленням подібне до навчання людей - машину карають за помилки і заохочують за правильні дії.

3.3. Огляд існуючих архітектур нейронних мереж для роботи з текстом

У сучасному світі обробка та розуміння тексту стали одними з найважливіших завдань у галузі обробки природної мови (NLP). Завдяки великому обсягу текстових даних, які щодня генеруються в Інтернеті, соціальних медіа, електронних документах і т.д., виникла необхідність у розробці потужних і ефективних алгоритмів та моделей для автоматичної обробки цих даних.

Нейронні мережі, спеціально призначені для роботи з текстовими даними, стали ключовим інструментом у досягненні цих цілей. Архітектури таких мереж базуються на ідеї використання глибокого навчання та штучних нейронних мереж для автоматичного виявлення та розуміння закономірностей в тексті. Ці мережі дозволяють моделювати складні залежності у текстових даних, враховувати контекст, виявляти семантичні взаємозв'язки та здійснювати високоякісний аналіз тексту.

3.3.1. Рекурентні нейронні мережі(RNN)

Рекурентні нейронні мережі (RNN) – це штучні нейронні мережі, які мають зворотні зв'язки, що працюють у протилежному напрямку разом з прямими зв'язками від входів (рецепторів) мережі до виходів (ефекторів). Рекурентні нейронні мережі є динамічними системами, що обробляють послідовності вхідних даних і перетворюють їх на послідовності реакцій. Поведінка рекурентних нейромереж відображає навчені стереотипи, що робить їх схожими на цілеспрямовані адаптивні динамічні системи, здатні досягати заздалегідь

визначених цілей. Проте програмування поведінки рекурентних нейромереж здійснюється шляхом навчання на прикладах і не вимагає формального визначення цілей. Вони ефективно працюють у ситуаціях невизначеності [14]. Рекурентні нейронні мережі нагадують нервову систему живих організмів за своєю архітектурою та здатністю адаптуватись до оточуючого середовища. Тому вивчення цих мереж має важливе значення як для практичних застосувань, так і для загального розвитку науки, оскільки воно допомагає розуміти явища адаптації в живій природі, розкривати механізми пам'яті, інтерпретувати відомі дані нейрофізіології і розробляти нові методи діагностики та лікування нервових і психічних захворювань.

Структуру рекурентної нейронної мережі можна побачити на схемі, зображеній на рисунку 1 [7]. В мережі є шари рецепторних і ефекторних нейронів або просто лінії передачі даних, які відповідають входам і виходам. Між цими шарами розташовані один або кілька шарів прихованих нейронів. Вхідні сигнали нейронів кожного шару мають прямі зв'язки з виходами нейронів попереднього шару і можуть мати зворотні зв'язки з виходами нейронів свого та наступних шарів. Зворотні зв'язки зазвичай містять елементи затримки, що надають нейромережі властивостей оперативної пам'яті. Нейрони в різних шарах можуть мати однакові або відрізнятись за типом активаційних функцій і характером нейропарадигми. Завдяки наявності затриманих зворотних зв'язків, рекурентні мережі є динамічними системами, їх поведінка має зовнішню складову, що відповідає спостережуваним значенням входу і виходу, а також приховану складову, що характеризує внутрішній стан нейромережі.

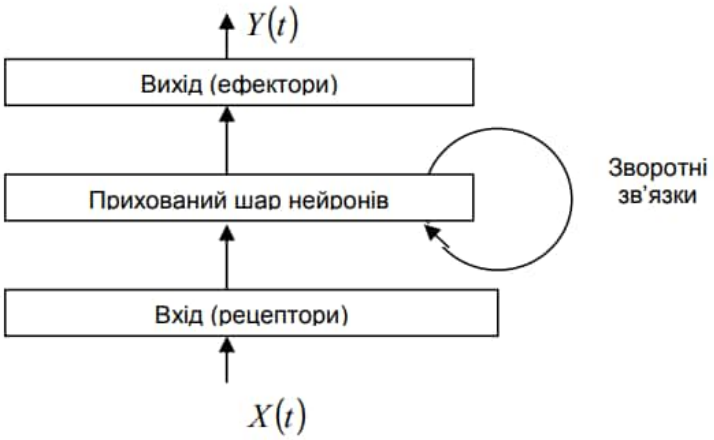

Рисунок 1. Загальна структура рекурентної нейронної мережі

Найбільш популярними та ефективними є рекурентні нейронні мережи типу LSTM та GRU.

LSTM (Long Short-Term Memory – довга короткострокова пам'ять). Рекурентні нейронні мережі, засновані на цьому підході, мають більш просунутий і складний спосіб обчислення значення прихованого шару. У цьому підході, крім вхідних значень і попереднього стану мережі, використовуються фільтри (gates), які визначають, як інформація буде використовуватися для обчислення вихідних значень на поточному шарі і значень прихованого шару на наступному кроці. LSTM широко використовується в області обробки природної мови, машинного перекладу, розпізнавання мови, генерації тексту та багатьох інших завдань, де важлива контекстна інформація. Кожен його вузол має три основні фільтри: вхідний фільтр (input gate), вихідний фільтр(output gate) та фільтр забування (forget gate). Ці фільтри контролюють потік інформації, яка пропускається через вузол. Вхідний фільтр регулює, яка інформація повинна бути оновлена в пам'яті вузла LSTM. Вихідний фільтр визначає, яка частина пам'яті повинна бути відправлена до наступного вузла. Фільтр забування визначає, яка частина пам'яті має бути забута, щоб уникнути накопичення непотрібної інформації. Крім того, LSTM має запам’ятовуючий блок(memory cell), який може зберігати та оновлювати інформацію протягом довгого періоду часу. Це дозволяє мережі запам'ятовувати

контекстну інформацію та використовувати її для передбачення наступних елементів в послідовності.

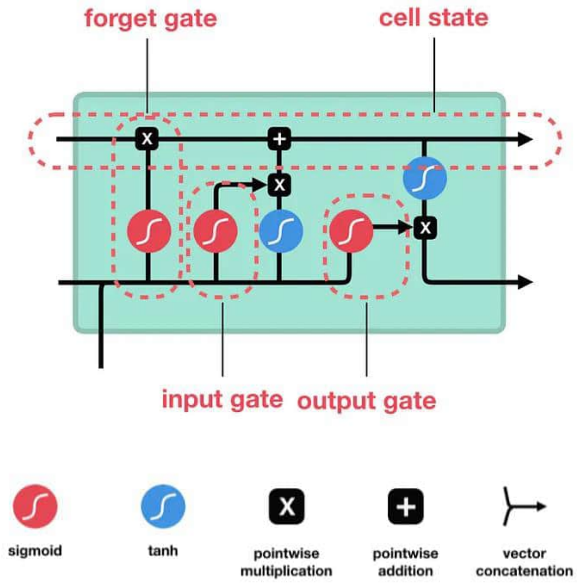

Рисунок 2. Архітектура LSTM шару

Модель GRU (Gated Recurrent Unit), заснована на тих же принципах, що і LSTM, але використовує менше фільтрів і операцій для обчислення значень прихованого шару. GRU використовує два фільтри: фільтр оновлення (update gate) і фільтр скидання (reset gate). Ці фільтри дозволяють контролювати потік інформації в мережі і забезпечують гнучкість для вирішення завдань моделювання послідовностей. Фільтр оновлення регулює, яка частина попереднього стану повинна бути передана до поточного часового кроку. Фільтр скидання визначає, яка інформація з попереднього стану повинна бути проігнорована в поточному кроці. Це дозволяє мережі враховувати поточну вхідну інформацію та залежності між попередніми і поточними станами. Одна з особливостей GRU полягає в тому, що вона не має окремого запам'ятовуючого блок(memory cell), як у LSTM. Замість цього, GRU використовує один внутрішній стан, який зберігає та передає інформацію від одного часового кроку до наступного. GRU є менш складним за структурою порівняно з LSTM, що робить його легшими у навчанні та використанні.

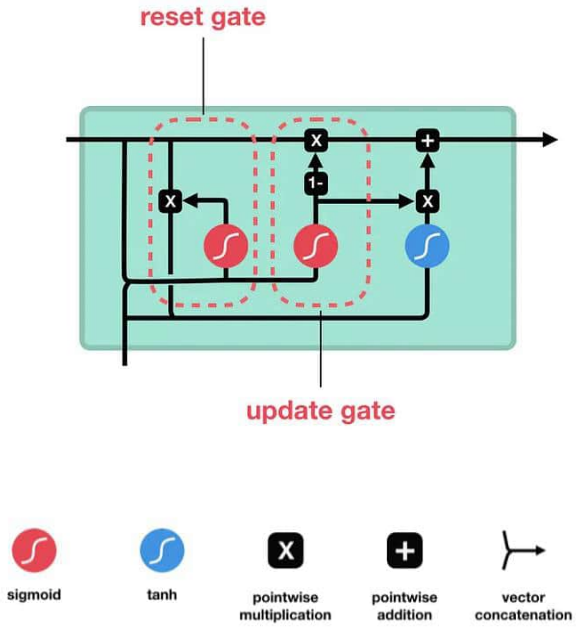

Рисунок 3. Архітектура GRU шару

3.3.2. Згорткові нейронні мережі(CNN)

Згорткові нейронні мережі(CNN) – це клас нейронних мереж, який спеціалізується на обробці даних, які мають сітчасту топологію, наприклад зображення, але також використовують і для аналізу тексту. У випадку обробки тексту, вхідні дані зазвичай представляються у вигляді векторів слів або закодованих символів. Подібно до того, як кожен нейрон реагує на стимули лише в обмеженій області поля зору, яка називається сприйнятливим полем у системі біологічного зору, кожен нейрон у CNN обробляє дані лише у своєму сприйнятливому полі. Шари розташовані таким чином, щоб спочатку виявляти прості залежності, а далі – складніші.

CNN зазвичай має три шари: згортковий, агрегувальний та повноз’єднаний шар.

Згорткові шари використовують фільтри або ядра, які рухаються по вхідних даним з кроком (крок згортки). Згортка — це математична операція, у якій елементи фільтра множаться поелементно на вхід, на якому присутній фільтр, і відповідні добутки підсумовуються щоб отримати вихідний елемент (рис.4). Фільтр продовжує перевіряти вхідні дані, виконуючи згортку та отримуючи вихідні

елементи. Згорткові нейронні мережі — це просто мережі, де присутні шари, які виконують згортки. В одному згортковому шарі може бути кілька фільтрів, які допомагають отримувати інформацію про різні вхідні функції.

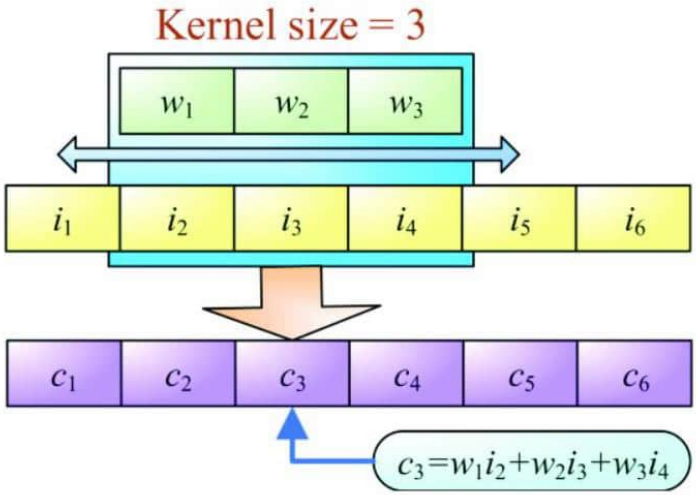

Рисунок 4. Приклад процесу згортки

Фільтри/ядра в CNN можуть допомогти визначити відповідні шаблони в текстових даних – біграми, триграми або n-грами (суміжна послідовність з n слів) залежно від розміру ядра. Оскільки CNN є інваріантними для перекладу, вони можуть виявляти ці моделі незалежно від їхньої позиції в реченні. Порядок слів не такий важливий у класифікації тексту, тому CNN можуть ефективно виконувати це завдання. Кожен фільтр/ядро виявляє певну особливість, наприклад, якщо речення містить позитивні («добре», «дивовижно») або негативні («погано», «жахливо») терміни у випадку аналізу тональності. Подібно до аналізу тональності, більшість завдань класифікації тексту визначаються наявністю або відсутністю деяких ключових фраз у будь-якому місці пропозиції. Такі завдання можуть бути ефективно змодельовані за допомогою CNN, які добре виділяють локальні та позиційно-інваріантні характеристики з даних.

3.3.3. Трансформери та BERT

3.3.3.1. Проблематика Seq2seq

У sequence-to-sequence(послідовності до послідовності)[9] проблемах, таких як нейронний машинний переклад, початкові пропозиції їх вирішення базувалися на використанні RNN в архітектурі кодера-декодера. Ці архітектури мали великі обмеження при роботі з довгими послідовностями, їх здатність зберігати інформацію з перших елементів була втрачена, коли нові елементи були включені в послідовність. У кодувальнику прихований стан на кожному кроці пов’язується з певним словом у вхідному реченні, зазвичай одним із останніх. Тому, якщо декодер отримує доступ лише до останнього прихованого стану декодера, він втратить відповідну інформацію про перші елементи послідовності.

Наприклад, в реченні “Хмари є на _____” наступним словом очевидно буде “небо”, оскільки є зв’язок зі словом “хмари”. Якщо відстань між “хмари” та передбачуваним словом – коротка, RNN може це легко передбачити. Розглянемо інший приклад: “Я виріс у Німеччині разом зі своїми батьками, провів там багато років і добре знаю їхню культуру. Ось чому я вільно розмовляю ____”. Очікуване слово “німецькою”, яке безпосередньо пов'язане з “Німеччина”. Однак у цьому випадку відстань між “Німеччина” та передбачуваним словом більша, тому RNN важче передбачити правильне слово. Отже, у міру того, як відстань зростає, RNN стають нездатними знайти зв’язки, через малу потужність пам’яті.

Тож, щоб усунути проблеми, які були в RNN, було введено нову концепцію – механізм уваги.

Замість того, щоб звертати увагу на останній стан енкодера, як це зазвичай робиться з RNN, на кожному кроці декодера ми переглядаємо всі стани кодера, маючи можливість отримати доступ до інформації про всі елементи вхідної послідовності. Це те, що робить “увага”, вона витягує інформацію з усієї послідовності, зваженої суми всіх минулих станів кодера. Це дозволяє декодеру

призначати більшу вагу або важливість певному елементу входження для кожного елемента виходу.

Але цей підхід продовжує мати важливе обмеження: кожна послідовність повинна оброблятися по одному елементу за раз. І кодер, і декодер повинні чекати до завершення кроків t-1, щоб обробити крок. Тому для великих об'ємів входжень, це займає багато часу.

3.3.3.2. Трансформери

Трансформери – нейронна мережа, архітектура якої спрямована на вирішення послідовних завдань, одночасно з легкістю обробляючи довгострокові залежності. Вперше це було запропоновано в статті "Увага — це все, що вам потрібно"[8]:

"У цій праці ми презентуємо Transformer, архітектуру моделі, яка уникає повторення і натомість повністю покладається на механізм уваги для створення глобальних залежностей між входом і виходом. Transformer дозволяє значно більше розпаралелювати. Transformer є першою моделлю трансдукції, яка повністю покладається на механізм самоуважності для обчислення представлень своїх вхідних і вихідних даних без використання вирівняних послідовностей RNN або згортки."

Модель Трансформерів виділяє характеристики для кожного слова за допомогою механізму самоуважності, щоб з'ясувати, наскільки важливі всі інші слова в реченні в відношенні до попередніх слів. І жодні повторювані одиниці не використовуються для отримання цих функцій, це просто зважені суми та активації, тому вони можуть бути дуже розпаралелюваними та ефективними.

3.3.3.3. Механізм самоуважності – self-attention

Механізм самоуважності — це операція seq2seq (послідовність-до-послідовності): послідовність векторів входить, а послідовність векторів виходить. Назвемо вхідні вектори *x1, x2,…, x_t* і відповідні вихідні вектори *y1, y2,…, y_t*. Усі

вектори мають розмірність k. Щоб створити вихідний вектор y_i, операція самоуважності просто бере зважене середнє за всіма вхідними векторами, найпростішим варіантом є скалярний добуток.

У механізмі самоуважності моделі є три елементи: Queries(запити), Values (значення) та Keys(ключі).

Кожен вхідний вектор використовується ці три елемента. У кожній ролі він порівнюється з іншими векторами, щоб отримати власний вихід y_i (Query), отримати j-й вихід y_j (Key) і обчислити кожен вихідний вектор після встановлення вагових коефіцієнтів (Value).

Щоб отримати ці ролі, нам потрібні три вагові матриці розмірів k x k і обчислити три лінійні перетворення для кожного x_i (рис.5).

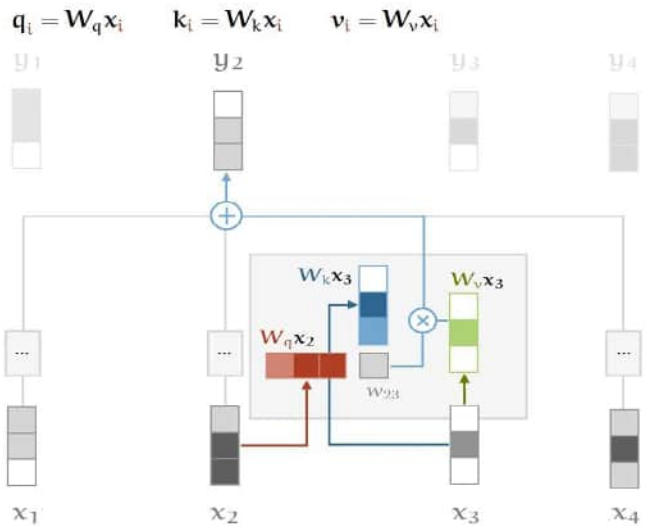

Рисунок 5. Процес створення трьох елементів механізму самоуважності

Ці три матриці зазвичай відомі як K, Q і V, три вагові шари, які можна навчати, які застосовуються до одного закодованого вхідного сигналу. Отже, оскільки кожна з цих трьох матриць надходить з одного входу, ми можемо застосувати механізм уваги вхідного вектора до самого себе – “самоуважність”.

Потім матриці Q, K і V використовуються для розрахунку показників уваги. Оцінки визначають, наскільки зосереджено увагу на інших місцях або словах у

вхідній послідовності порівняно зі словом у певній позиції. Тобто скалярний добуток вектора Q на K вектор відповідного слова, яке ми оцінюємо.

Далі застосовується коефіцієнт «масштабування», щоб мати більш стабільні градієнти. Функція *softmax* не може належним чином працювати з великими значеннями, що призводить до зникнення градієнтів і уповільнення навчання. Після її використання, результат множиться на матрицю V, щоб зберегти значення слів, на яких ми хочемо зосередитися, і мінімізуємо або видаляємо значення для нерелевантних слів (її значення в матриці V має бути дуже малим).

3.3.3.4. BERT

Розвитком архітектури Трансформерів стали сім'я моделей BERT. BERT, скорочення від Bidirectional Encoder Representations from Transformers, — це модель машинного навчання для обробки природної мови (рис. 6). Він був розроблений у 2018 році дослідниками з Google AI Language і слугує рішенням для 11+ найпоширеніших мовних завдань, таких як аналіз настроїв і розпізнавання іменованих об’єктів.

Величезний набір даних із 3,3 мільярдів слів сприяв постійному успіху BERT. BERT пройшов спеціальне навчання з Вікіпедії (~2,5 млрд слів) і Google BooksCorpus (~800 млн слів). Ці великі інформаційні масиви сприяли глибоким знанням BERT не лише англійської мови. Навчання на такому великому наборі даних займає багато часу. Навчання BERT стало можливим завдяки новій архітектурі Трансформерів і пришвидшено завдяки використанню TPU (Tensor Processing Units – спеціальна схема Google, створена спеціально для великих моделей ML).

Ключовою технічною інновацією BERT є застосування двонаправленого навчання Трансформера до мовного моделювання. Це відрізняється від попередніх спроб, які розглядали послідовність тексту зліва направо або комбінували навчання зліва направо і справа наліво. Відповідно до результатів статті[8], мовна модель, яка навчається двонаправлено, може мати глибше відчуття мовного контексту та

потоку, ніж однонаправлені мовні моделі. У статті дослідники описують нову техніку під назвою Masked LM (MLM), яка дозволяє використовувати двонаправлене навчання на моделях, у яких раніше це було неможливо.

BERT використовує механізм уваги Трансформеру, який вивчає контекстні зв’язки між словами у тексті. У своїй базовій формі Трансформер містить два окремі механізми — кодер, який зчитує введений текст, і декодер, який створює прогноз для завдання. Оскільки метою BERT є створення мовної моделі, необхідний лише механізм кодування.

На відміну від прямих моделей, які зчитують введений текст послідовно (зліва направо або справа наліво), кодувальник Трансформер одночасно обробляє всю послідовність слів. Тому його можна вважати двонаправленим, хоча точніше було б сказати, що він ненаправлений. Ця характеристика дозволяє моделі вивчати контекст слова на основі всього вхідного потоку даних.

Є два етапи в архітектурі BERT:

- Попереднє навчання (Pre-training): модель навчається на немаркованих даних у двох неконтрольованих завданнях попереднього навчання (моделювання замаскованої мови Masked LM та передбачення наступної послідовності NSP).
- Точне налаштування (Fine-tunning): модель BERT ініціалізується попередньо підготовленими параметрами, і ці параметри точно налаштовуються з використанням маркованих даних.

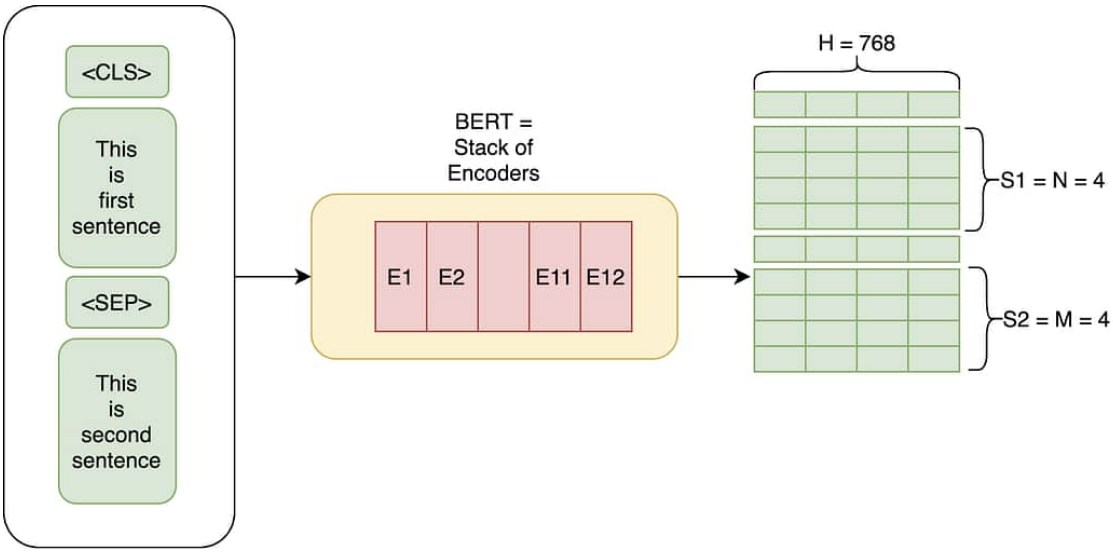

Рисунок 6. Архітектура BERT

BERT виконує широкий спектр завдань, тому представлення вхідних даних має бути подібним для одного речення або пари речень. Процес представлення вхідних даних складається (рис.7) з токенізації(Tokenization) та вкладення слів(Embedding — процес, в результаті якого кожному слову або токену присвоюється векторна представленість у просторі). Існує два основних токени: [CLS] і [SEP].

Токен [CLS] — це перший маркер, який додається до кожної послідовності, це спеціальний маркер класифікатора. Токен [SEP] використовується для позначення розділення пар речень. Для пар речень ми також додаємо вивчений ембендінг, звідки воно походить (речення A або речення B).

Звідти кожне слово розбивається на фрагменти символів, присутні в словнику (токени). Ця токенізація потім перетворюється на ембендінг (вектори).

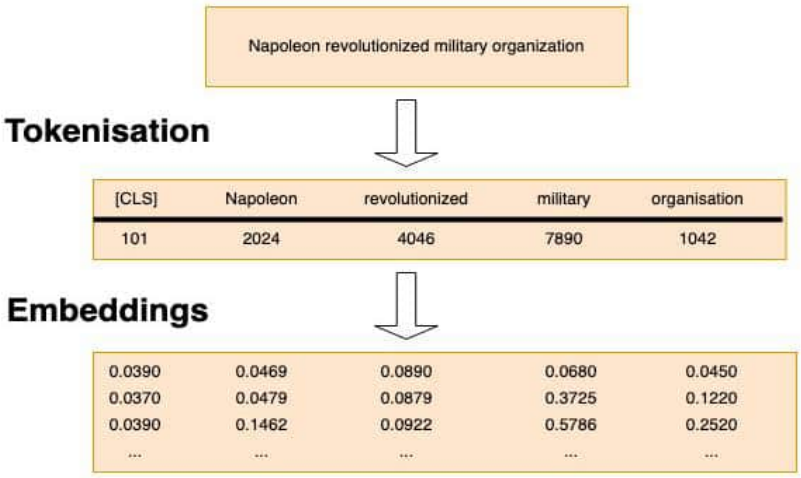

Рисунок 7. Процес токенізації та ембендінгу

Отже структура подання вхідних даних має такий вигляд як на рисунку 8.

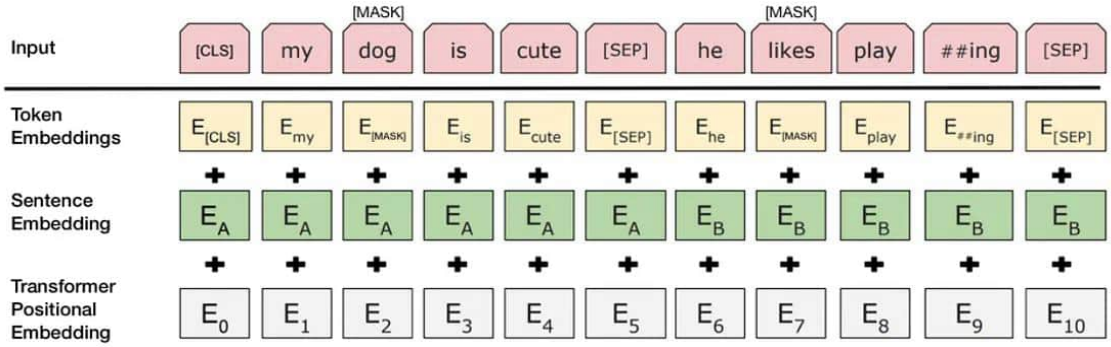

Рисунок 8. Процес подання тренувальних даних

У цьому прикладі для двох речень «мій пес милий» і «він любить гратись» (рис. 9) BERT спочатку використовує токенізацію текстового фрагмента, щоб перетворити послідовність у токени, і додає маркер [CLS] на початку та маркер [SEP] у початок і кінець другого речення, тому вхідні дані мають такий вигляд:

Рисунок 9. Початок перетворення вхідних даних

Token Embeddings (вбудовування маркерів): потім ми отримуємо вбудовування маркерів, індексуючи матрицю розміром 30000x768(H). Тут 30000 — це довжина Vocab після токенізації WordPiece (рис.10). Вага цієї матриці визначається під час навчання.

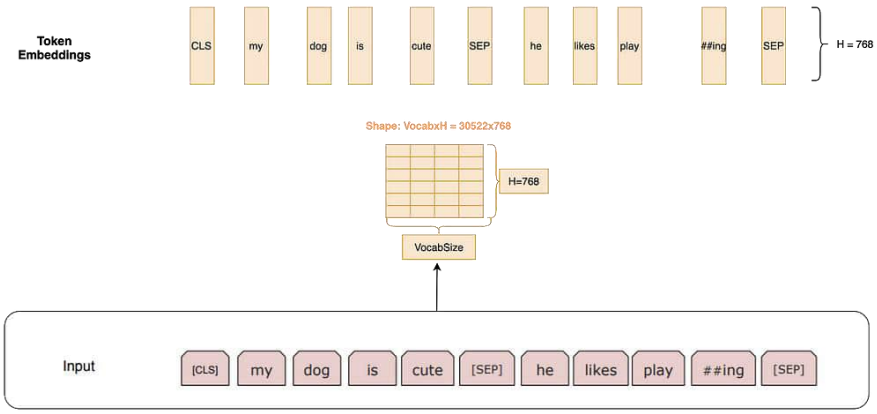

Рисунок 10. Token Embeddings відбувається шляхом індексування матриці розміром Vocab*x*H.

Segment Embeddings (вбудовування сегментів): для таких завдань, як відповіді на питання, ми повинні вказати, з якого сегмента це речення. Це або 0-вектор довжини H, якщо вхідні дані з першого речення, або вектор з 1, якщо дані з другого речення.

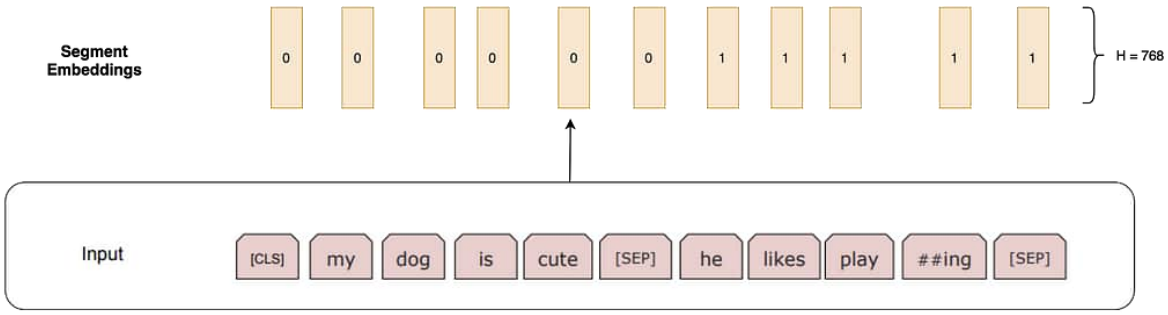

Рисунок 11. Segment Embeddings – це 0 або 1-вектори, що визначають 1-е або 2-е речення.

Position Embeddings (вбудовування позиції): це вбудовування, яке використовується для визначення позиції слів у послідовності, так само, як в архітектурі Трансформерів. Отже, по суті, маємо постійну матрицю з деяким попередньо встановленим шаблоном. Ця матриця має 768 стовпців. Перший рядок цієї матриці – це ембендінг для маркера [CLS], другий рядок – це ембендінг для слова «my», третій рядок – це ембендінг для слова «dog» і так далі.

Таким чином, остаточні вхідні дані, надані BERT, це вбудовування маркерів + вбудовування сегментів + вбудовування позицій.

Тепер принципи завдяки яким BERT відрізняється від інших. Якби просто ввели двонаправлене обумовлення, це дозволило б кожному слову опосередковано бачити себе в багатошаровому контексті. Це було б величезною катастрофою. Щоб цього не сталося, ми використовуємо моделювання замаскованої мови або Masked LM (рис.12).

Ідея в Masked LM проста:

- випадково замасковується 15% слів у вхідних даних — замінивши їх маркером [MASK]

- пропустити всю послідовність через кодер на основі принципу самоуважності BERT
- передбачити лише замасковані слова на основі контексту, наданий іншими незамаскованими словами в послідовності

Під час навчання функція втрат BERT враховує лише передбачення замаскованих токенів і ігнорує передбачення незамаскованих. Це призводить до моделі, яка сходиться набагато повільніше, ніж моделі зліва направо або справа наліво, але це компенсується її підвищеним усвідомленням контексту.

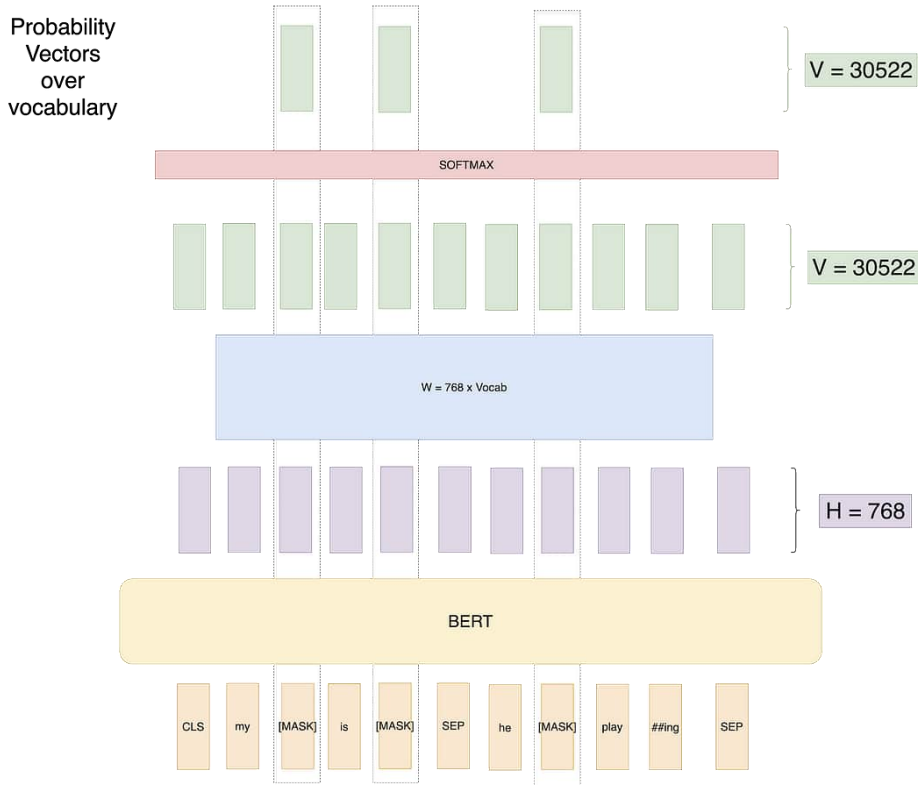

Рисунок 12. Masked LM

Щоб зрозуміти зв’язок між двома реченнями, навчання BERT моделі також використовує передбачення наступного речення(NSP) (рис.13). Під час навчання модель отримує вхідні пари, і вчиться передбачати, чи є друга думка наступною в оригінальному тексті.

Під час навчання модель отримує два вхідних речення одночасно так, щоб:

- У 50% випадків друге речення йде після першого.

- У 50% випадків це випадкове речення з усього тексту.

Потім використовується вихідне значення токена CLS, щоб отримати двійкові втрати, які також передаються через мережу, щоб дізнатися ваги.

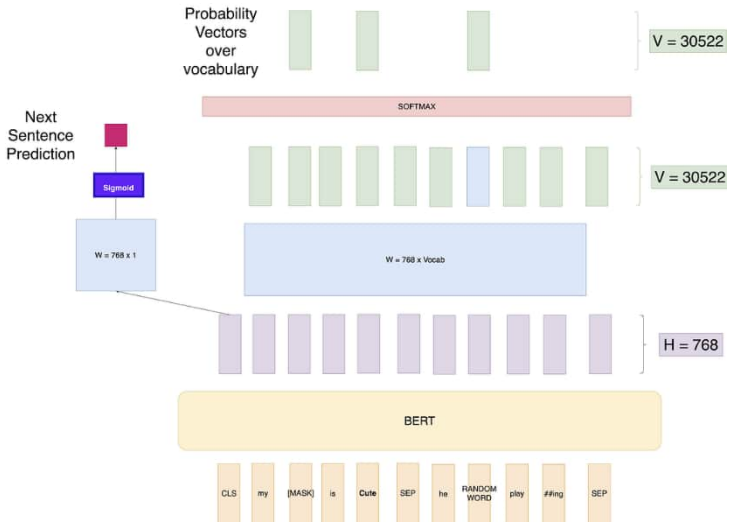

Рисунок 13. Навчання BERT із завданням NSP

Модель навчається як за Masked LM, так і за допомогою NSP разом. Це робиться для мінімізації комбінованої функції втрат двох стратегій.

Для виконання більшості завдань машинного навчання BERT потрібен лише один додатковий шар поверх результату [CLS] і точно налаштувавши ваги. Наприклад, для задач класифікації (рис. 14) потрібно додати шари Linear + Softmax поверх результату CLS розміром 768.

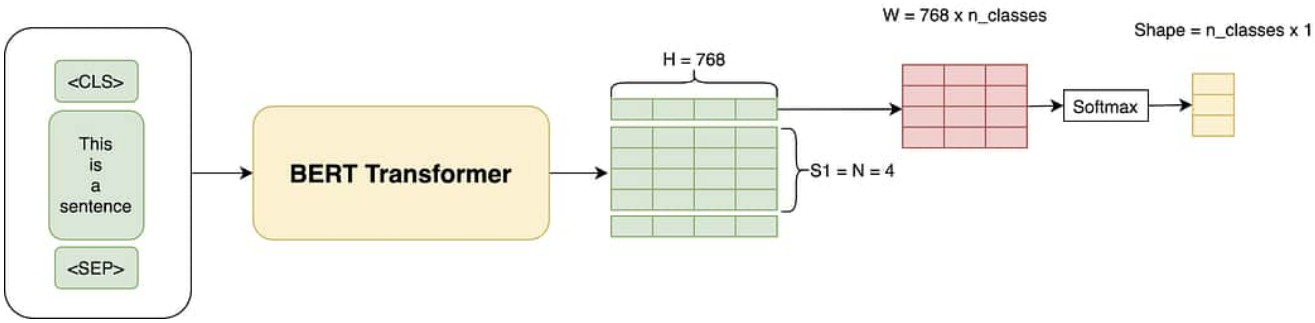

Рисунок 14. Архітектура BERT для задачі класифікації

4. РОЗРОБКА МОДЕЛІ НЕЙРОННОЇ МЕРЕЖІ ДЛЯ АНАЛІЗУ ТЕКСТУ

4.1. Інструменти для розробки нейронної мережі

4.1.1. Python

В якості основної мови програмування було обрано мову програмування Python. Python створений Гвідо ван Россумом і перший реліз відбувся в 1991 році. Об’єктно-орієнтований підхід допомагає програмістам написати чіткий, логічний код для малих та великих проектів. Через простий синтаксис і його кросплатформеність, ця мова швидко розвивається і є дуже популярною для проведення наукових обрахунків та машинного навчання. Однак найбільшою перевагою Python є велика кількість бібліотек, що можуть бути використані для наукових досліджень з використанням машинного навчання та нейронних мереж.

4.1.2. BERT

В попередньому розділі, я описала різні архітектури нейронних мереж для роботи з текстом, такі як рекурентні нейронні мережі(RNN), згорткові нейронні мережі(CNN) та інші. Цей огляд дозволив отримати уявлення про переваги та обмеження кожної з архітектур та їхню ефективність у вирішенні конкретних завдань.

І для свого завдання вирішила обрати модель BERT. Обираючи модель BERT, я врахувала кілька ключових факторів. По-перше, BERT здатний ефективно моделювати контекст та залежності між словами у тексті завдяки механізму самоуважності, який використовується в Трансформерах. Це дозволяє BERT виявляти семантичні зв'язки та розуміти сенс тексту на глибшому рівні.

По-друге, BERT є пре-тренованою моделлю, що означає, що вона була попередньо навчена на великому обсязі текстових даних. Це дозволяє моделі

засвоїти загальні залежності та властивості мови, що призводить до кращої продуктивності при доопрацюванні на конкретні завдання.

4.1.3. Transformers

Одним з ключових факторів, які призвели до вибору трансформерів для роботи з BERT, була їхня здатність до ефективної моделювання контексту в тексті. BERT використовує трансформерну архітектуру, щоб врахувати широкий контекст та взаємодії між словами в тексті. Це дозволяє BERT виявляти семантичні залежності та розуміти сенс тексту на більш глибокому рівні. Тому, обрання трансформерів для BERT було обумовлено їхньою потужністю в моделюванні контексту та здатністю до роботи з великими обсягами текстових даних.

4.1.4. Torch

Torch(та його оболонка PyTorch) – один із найпопулярніших пакетів для розробки нейронних мереж та реалізації алгоритмів оптимізації. Torch є високоефективною бібліотекою, оптимізованою для роботи з нейронними мережами. Вона використовує прискорювачі, такі як графічні процесори (GPU), для швидкого обчислення та тренування моделей. Він надає гнучкі інструменти для розробки нейронних мереж. Вона дозволяє легко створювати та налаштовувати складні архітектури, такі як трансформери, і має багато вбудованих функцій для роботи з текстовими даними.Torch може легко інтегруватися з іншими популярними бібліотеками, такими як NumPy та Pandas, що дозволяє зручно обробляти та аналізувати дані перед використанням їх у нейронних мережах.

4.1.5. Hugging Face

Hugging Face, Inc.[11] — американська компанія, яка розробляє інструменти для створення програм за допомогою машинного навчання. Отримала відомість завдяки створенню бібліотеки Transformers для обробки природної мови і

платформи, яка дозволяє користувачам обмінюватися моделями машинного навчання та наборами даних.

Hugging Face надає широкий вибір готових моделей, включаючи BERT, які можна використовувати безпосередньо. Це дозволяє значно скоротити час розробки та тренування моделі. Забезпечує простий та зрозумілий інтерфейс для завантаження, конфігурації та використання моделей.

Hugging Face надає додаткові інструменти та бібліотеки для роботи з нейронними мережами, включаючи різні модулі для обробки текстових даних, оцінки моделей та зручного візуалізації результатів.

Hugging Face має велику та активну спільноту розробників та дослідників, що сприяє постійному оновленню та вдосконаленню платформи. Також існує багато ресурсів, які допомагають вирішувати питання та проблеми, що виникають під час роботи з моделями BERT.

4.1.6. Jupyter Notebook

Проект Jupyter [12] – платформа для інтерактивних обчислень, яка працює з усіма мовами програмування шляхом використання вільного програмного забезпечення та відкритих стандартів. Ця платформа, яка підтримує більше 40 мов програмування, є особливо популярною для розробки коду на Python, R, Julia та Scala.

Головним елементом Jupyter є "записники" (notebooks) - файловий формат, який містить код та його виконані результати, підтримує машинне навчання і проекти з аналізу даних.

Основні переваги Jupyter Notebook:

- редагування коду в браузері, з автоматичним підсвічуванням синтаксису;
- можливість виконувати код із браузера з результатами обчислень, створені кодом;
- відображення результату обчислення з використанням мультимедійних форматів, таких як HTML, LaTeX, PNG, SVG тощо;

- редагування форматованого тексту в браузері з використанням мови розмітки Markdown, яка дозволяє додавати коментарі до коду, не обмежуючись простим текстом;
- можливість легко включати математичні позначення в клітинки коментування за допомогою LaTeX і відтворюватися MathJax.

4.2. Збір даних для проекту

Для ефективної роботи нейронної мережі та її навчання, потрібно вибрати правильні дані. Оскільки моя тема пов’язана з пошуком фейкових новин, пов'язаних з російсько-українською війною, саме їх і потрібно було разом з правдивими новинами.

Однак я зіткнулася з проблемою, бо готових датасетів по цій тематиці немає. Відповідно потрібно було збирати дані самостійно з початку.

Джерелом даних, я обрала різні сайти з новинами як українські так і російські. Це дозволило отримати різноманітність даних щодо російсько-української війни. Також пізніше знайшла сайт[13], який збирав окремо фейкові новини, і спростовував їх. Цей ресурс став важливим джерелом для збору фейкових новин про війну. Під час збирання даних, додатково проводила аналіз кожної новини. Перевірила джерело, дату публікації, автора новини і шукала, чи ще десь згадувалась ця новина.

Через те, що я вручну вводила все з самого початку, я одразу і маркувала ці новини в колонці Label, де позначала 0 для правдивих новин, а 1 для фейкових. Надалі, на основі цих міток нейронна мережа вчилась та тренувалась.

4.3. Архітектура нейронної мережі

Для виконання поставленого завдання я використовую модель нейронної мережі BERT. Як вже описувалось в попередніх пунктах, вона підходить для задач з обробки природної мови і не потребує надзвичайно великої кількості даних.

4.3.1. Завантаження BERT моделі та токенізатора

Я використовую pre-trained модель BERT для обробки тексту. Конкретно, у коді ініціалізується об'єкт моделі та токенізатора BERT, що дозволяє використовувати натреновану модель для подальшої обробки тексту (рис.15).

Пройдемось по кожному рядка коду:

1. *bert = AutoModel.from_pretrained('bert-base-multilingual-uncased')* – цей рядок ініціалізує об'єкт моделі BERT за допомогою функції from_pretrained з бібліотеки Hugging Face Transformers. В моєму випадку використовується pre-trained модель BERT з назвою bert-base-multilingual-uncased, яка була натренована на великому корпусі текстів на різних мовах, враховуючи і українську, з нижніми регістрами.
2. *tokenizer=BertTokenizerFast.from_pretrained('bert-base-multilingual-uncased')* – цей рядок ініціалізує об'єкт токенізатора BERT за допомогою функції from_pretrained з бібліотеки Hugging Face Transformers. Я використовую токенізатор, який відповідає pre-trained моделі BERT з назвою bert-base-multilingual-uncased. Токенізатор розбиває вхідний текст на токени (слова або підрядки слів), які потім можна використовувати для подальшої обробки.

Після виконання цих двох рядків коду bert та tokenizer готові до використання для обробки тексту.

```
bert = AutoModel.from_pretrained('bert-base-multilingual-uncased')
tokenizer = BertTokenizerFast.from_pretrained('bert-base-multilingual-uncased')
```

Рисунок 15. Завантаження BERT моделі та токенізатора

4.3.2. Підготовка даних для BERT моделі

Вхідні дані я розділяю на тренувальний, валідаційний та тестовий, використовуючи функцію *train_test_split* з бібліотеки *scikit-learn* для розподілу даних на піднабори з заданими розмірами (рис.16). Основні аргументи функції *train_test_split*:

- data['text'] та data['label'] є вхідними даними, де data['text'] містить текстові дані, а data['label'] – відповідні мітки або класи;
- random_state визначає початкове значення для генератора випадкових чисел, що забезпечує відтворюваність результатів;
- test_size вказує відсоток даних, який буде виділений для тестування. У даному випадку 0.3 означає, що 30% даних буде використано для тестування;
- stratify вказує, що розподіл класів у вихідних даних буде збережено в розділених піднаборах, щоб забезпечити репрезентативність даних у кожному піднаборі.

Результатом цього коду є отримання змінних *train_text, val_text, test_text, train_label, val_label* та *test_label,* які містять відповідні текстові дані та мітки для тренувального, валідаційного та тестового наборів даних. Це дає змогу використовувати ці змінні для подальшої підготовки та навчання моделі машинного навчання.

```
train_text, temp_text, train_label, temp_label = train_test_split(data['text'],
                                                                  data['label'],
                                                                  random_state=2018,
                                                                  test_size=0.3,
                                                                  stratify=data['label'])

val_text, test_text, val_label, test_label = train_test_split(temp_text,
                                                              temp_label,
                                                              random_state=2018,
                                                              test_size=0.5,
                                                              stratify=temp_label)
```

Рисунок 16. Функція train_test_split()

Наступним етапом є використання токенізатора для обробки тексту. Код здійснює токенізацію текстових даних та конвертацію їх у числовий формат, який може бути використаний для тренування моделі BERT на цих даних. В попередніх кроках я ініціалізувала об’єкт tokenizer, і використовою його метод *batch_encode_plus()* з такими параметрами(рис.17):

1. train_text.tolist() – це список текстових даних, які ми хочемо токенізувати та конвертувати у числовий формат.
2. max_length – це параметр, який визначає максимальну довжину тексту після токенізації. Якщо текст перевищує цю максимальну довжину, то він буде обрізаний до максимальної довжини або заповнений токеном padding, якщо його довжина менша за максимальну.
3. pad_to_max_length – це параметр, який вказує, чи потрібно заповнювати текст токенами padding до максимальної довжини, якщо він коротший за неї.
4. truncation – це параметр, який вказує, чи потрібно обрізати текст, якщо він перевищує максимальну довжину.

Після виконання цього коду tokens_train містить токенизовані та конвертовані в числовий формат текстові дані. Також токенізацію роблю і для валідаційного та тестового набору даних.

```
tokens_train = tokenizer.batch_encode_plus(
    train_text.tolist(),
    max_length = MAX_LENGTH,
    pad_to_max_length = True,
    truncation = True
)
```

Рисунок 17. Токенізація тренувального набору даних

Потім я беру токенізовані та конвертовані до числового формату текстові дані з попереднього коду та конвертую їх у формат, який можна використовувати для тренування моделі з використанням бібліотеки PyTorch(рис.18).

Покрокове пояснення коду:

1. tokens_train['input_ids'] – це числовий представник токенів тексту, який міститься в словнику tokens_train, згенерованому після токенізації тексту за допомогою BertTokenizerFast. Ці числа відповідають індексам слів у відповідному словнику моделі BERT.

2. train_seq = torch.tensor(tokens_train['input_ids']) – конвертуємо список індексів токенів у формат torch.tensor з використанням фреймворку PyTorch, щоб можна було використовувати ці дані для тренування моделі. Таким чином, train_seq містить числове представлення текстових даних.
3. tokens_train['attention_mask'] – це маска уваги, яка позначає, які токени є реальними, а які - заповнювачами (padding tokens) у вхідному реченні. Це потрібно, щоб модель BERT знала, на які токени варто звернути увагу, а на які - ні, при обробці вхідних даних.
4. train_mask = torch.tensor(tokens_train['attention_mask']) – конвертуємо список масок у формат torch.tensor, щоб потім використати ці дані для тренування моделі. Таким чином, train_mask містить маску уваги.
5. train_label.tolist() – це список міток (labels), які відповідають кожному текстовому запису в наборі даних для тренування моделі.
6. train_y = torch.tensor(train_label.tolist()) – конвертуємо список міток у формат torch.tensor, щоб використати ці дані для тренування моделі. Таким чином, train_y містить мітки, які використовуються для тренувань.

```
train_seq = torch.tensor(tokens_train['input_ids'])
train_mask = torch.tensor(tokens_train['attention_mask'])
train_y= torch.tensor(train_label.tolist())
```

Рисунок 18. Конвертування списків в тензори

4.3.3. Створення моделі BERT_Arch

Тепер я визначаю архітектуру моделі BERT, яка базується на попередньо завантаженій BERT-моделі. Додаю до неї кілька шарів, щоб отримати вихідний результат для задачі класифікації тексту.

Основна частина коду це клас BERT_Arch(nn.Module), який є підкласом nn.Module з бібліотеки PyTorch (рис. 19). В методі __init__, першим аргументом передається вхідна модель bert. Після цього визначаються інші шари:

- dropout – шар Dropout для запобігання перенавчання
- relu – функція активації ReLU,
- fc1 та fc2 – два повнозв'язних шари з розмірами вихідних та прихованих шарів
- softmax – шар, який перетворює вихідний результат в імовірності.

У методі forward, модель отримує вхідні дані sent_id та mask (текст та маска), та виконує передачу даних через модель bert, щоб отримати останній прихований стан (cls_hs). Далі cls_hs передається через повнозв'язний шар fc1, після чого застосовується функція активації ReLU, а потім Dropout. Наступним кроком є передача вихідного результату через повнозв'язний шар fc2, після чого отримується вихідний результат, який подається через шар softmax для перетворення в імовірності.

```
class BERT_Arch(nn.Module):
  def __init__(self,bert):
    super(BERT_Arch,self).__init__()
    self.bert = bert
    self.dropout = nn.Dropout(0.1)
    self.relu = nn.ReLU()
    self.fc1 = nn.Linear(768,512)
    self.fc2 = nn.Linear(512,2)
    self.softmax = nn.LogSoftmax(dim=1)
  def forward(self,sent_id,mask):
    _,cls_hs = self.bert(sent_id, attention_mask = mask, return_dict=False)
    x = self.fc1(cls_hs)
    x = self.relu(x)
    x = self.dropout(x)
    x = self.fc2(x)
    x = self.softmax(x)
    return x

model = BERT_Arch(bert)
```

Рисунок 19. Визначення архітектури моделі BERT

4.4. Тренування нейронної мережі

4.4.1. Оптимізація моделі

Перш ніж почати тренування моделі, я її оптимізую (рис.20). Для цього я використовую *AdamW* – оптимізатор, який є покращеним алгоритмом оптимізації Adam з додатковою регуляризацією ваг. Це дозволяє зменшити перенавчання моделі та покращити її загальну точність. *lr* - це швидкість навчання моделі, яка задається в якості параметру *AdamW*.

В якості функції втрат для вимірювання точності моделі використовую *nn.NLLLoss()* – Negative Log Likelihood Loss. Ця функція відповідає задачі класифікації з логарифмічною функцією активації.

Epochs - це кількість повторень навчання моделі на всьому навчальному наборі даних. Спочатку, модель буде навчатися 3 рази на всьому навчальному наборі даних. Якщо ж результат буде не досить хороший, є можливість збільшити кількість “епох”.

In [30]:
```
import torch
from torch.optim  import AdamW
#from tensorflow.keras.optimizers import Adam

optimizer = AdamW(model.parameters(),lr=1e-5)
cross_entropy = nn.NLLLoss()
epochs =3
```

Рисунок 20. Код створення оптимізатора та функції втрат

4.4.2. Функції тренування та оцінювальна функція

Створюю функцію для тренування моделі (рис.21). Спочатку встановлюю саму модель у режим навчання – *model.train()*. Визначаю змінні для обрахунку втрати і точності, щоб потім отримати середнє значення для моделі. Далі циклом

проходжусь через тренувальний даталоадер, де кожна операція це одна партія даних

Вже в самому циклі, я розпаковую партію даних на окремі змінні такі як sent_id,mask,labels. Метод zero_grad() очищає градієнти, щоб уникнути їх накопичення з попередніх ітерацій. Рядок *preds = model(sent_id, mask)* отримує передбачення *preds* для кожного зразка, передаючи частину даних через модель. Потім я обчислюю втрату за допомогою функції *NLLLoss* і накопичую для подальшого обрахунку. Рядок *torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)* виконує обмеження градієнтів для уникнення проблеми з вибухом/зникненням градієнтів. І останнє, виконується оптимізаційний крок, оновлюючи параметри моделі на основі обчислених градієнтів.

Після завершення циклу обчислюється середня втрата *avg_loss* шляхом поділу загальної втрати на кількість партій у тренувальному даталоадері. Функція повертає середню втрату *avg_loss* для подальшого моніторингу та оцінки навчання моделі.

In [31]:
```
def train():
  model.train()
  total_loss, total_accuracy = 0,0

  for step,batch in enumerate(train_dataloader):
    if step%50 == 0 and not step == 0:
      print('  Batch {:>5,} of {:>5,}.'.format(step, len(train_dataloader)))
    batch = [r for r in batch]
    sent_id,mask,labels = batch
    model.zero_grad()
    preds = model(sent_id,mask)
    loss = cross_entropy(preds,labels)
    total_loss = total_loss + loss.item()
    loss.backward()
    torch.nn.utils.clip_grad_norm_(model.parameters(),1.0)
    optimizer.step()
    preds = preds.detach().cpu().numpy()

  avg_loss = total_loss / len(train_dataloader)
  return avg_loss
```

Рисунок 21. Функція тренування моделі

Далі я створюю оцінювальну функцію для моделі, яка виконує процес оцінки моделі на валідаційному наборі даних (рис. 22). Спочатку я встановлюю модель в

режим оцінки за допомогою методу *model.eval()*. Потім маю схожі кроки що і в функції *train()* однак тепер циклом проходжусь по валідаційному даталоадеру. І також використовую with *torch.no_grad()* метод, який використовує контекст без обчислення градієнтів, щоб вимкнути автоматичне обчислення градієнтів та зменшити використання пам'яті.

Результатом *evaluate()* є також середня втрата *avg_loss*, яка обраховувалась впродовж циклу.

In [32]:
```
def evaluate():
  print("\nEvaluating...")
  model.eval()
  total_loss, total_accuracy = 0,0
  for step,batch in enumerate(val_dataloader):
    if step%50 == 0 and not step == 0:
      print('  Batch {:>5,} of {:>5,}.'.format(step, len(val_dataloader)))
    batch = [t for t in batch]
    sent_id,mask,labels = batch
    with torch.no_grad():
      preds = model(sent_id,mask)
      loss = cross_entropy(preds,labels)
      total_loss = total_loss + loss.item()
      preds = preds.detach().cpu().numpy()
  avg_loss = total_loss / len(val_dataloader)
  return avg_loss
```

Рисунок 22. Оцінювальна функція моделі

4.4.3. Тренування моделі

Створивши попередні функції, можна розпочинати саме тренування моделі (рис. 23).

Створюю цикл, який проходить через кожну епоху навчання. Викликаю функцію навчання train() та отримую середнє значення втрат на тренувальних даних. Потім викликаю функцію оцінки evaluate() і також отримую середнє значення втрат, але вже на валідаційних даних. Якщо валідаційна втрата була кращою, тобто меншою, зберігаю параметри моделі.

In [33]:
```
best_valid_loss = float('inf')
train_losses = []
valid_losses = []
```

In [36]:
```
for epoch in range(epochs):
  print('\n Epoch {:} / {:}'.format(epoch+1,epochs))
  train_loss = train()
  valid_loss = evaluate()
  if valid_loss < best_valid_loss:
    best_valid_loss = valid_loss
    torch.save(model.state_dict(),'c1_new_model_weights.pt')
  train_losses.append(train_loss)
  valid_losses.append(valid_loss)

  print(f'\n Training loss: {train_loss:.3f}')
  print(f'\n Validation loss: {valid_loss:.3f}')
```

Рисунок 23. Тренування моделі

5. Результати моделі

Моя модель навчалась на двох наборах даних об’ємом 5000 записів та 40000 записів. Оскільки на початку дослідження було важко знайти потрібно дані, було спробувано натренувати на малій кількості даних і провести пізніше порівняльній аналіз залежності якості моделі залежно від об'єму датасету.

Оскільки при навчанні, я обраховувала втрати на тренувальних даних і на валідаційних, тепер можна побачити як з кожною епохою навчання втрати змінювались. При більшому датасеті, втрати зменшувались поступово і різниця в значеннях помітно зменшується, що видно на рисинку 24. Проте на меншому датасеті (рис. 25) видно, що значення зменшується зовсім трохи, а втрата на валідаційних даних взагалі майже не зменшується.

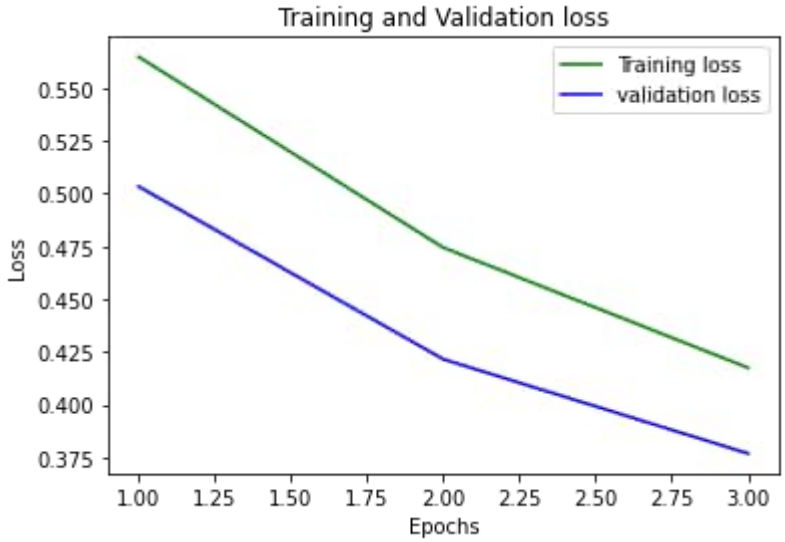

Рисунок 24. Втрати при навчанні моделі на кількості даних 40000

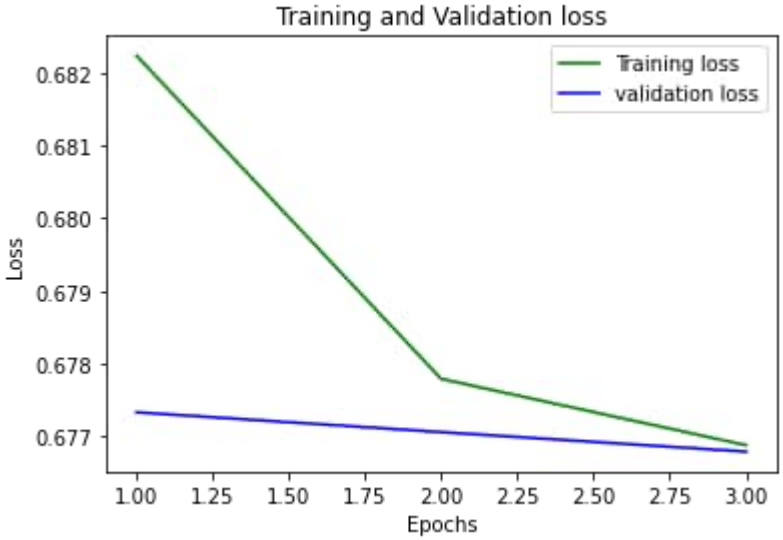

Рисунок 25. Втрати при навчанні моделі на кількості даних 5000

Для опису ефективності моєї моделі, я виводжу матрицю невідповідностей. За допомогою матриці невідповідностей можна отримати наступні метрики для оцінки результатів моделі:

1. Точність (Accuracy): Відсоток правильно класифікованих екземплярів відносно загальної кількості екземплярів.
2. Підрахунок чутливості (Recall): Відсоток правильно виявлених позитивних екземплярів відносно всіх фактичних позитивних екземплярів
3. Передбачувана позитивна вартість (Positive Predictive Value) або Точність (Precision): Відсоток правильно класифікованих позитивних екземплярів відносно всіх прогнозованих позитивних екземплярів.
4. F-міра (F-measure): Гармонічне середнє між показником чутливості та показником передбачуваної позитивної вартості. Використовується для оцінки збалансованості між чутливістю та точністю моделі.

Для моделі навченої на більшому датасеті (рис. 26) можна побачити, що точність моделі для фейкових новин (label - 1) – 0.92, а для правдивих (label - 0) – 0.79. Чутливість моделі 0.78 і 0.92 відповідно. Тож як видно метрики дуже високі, що означає модель добре навчилась і може прогнозувати правильно.

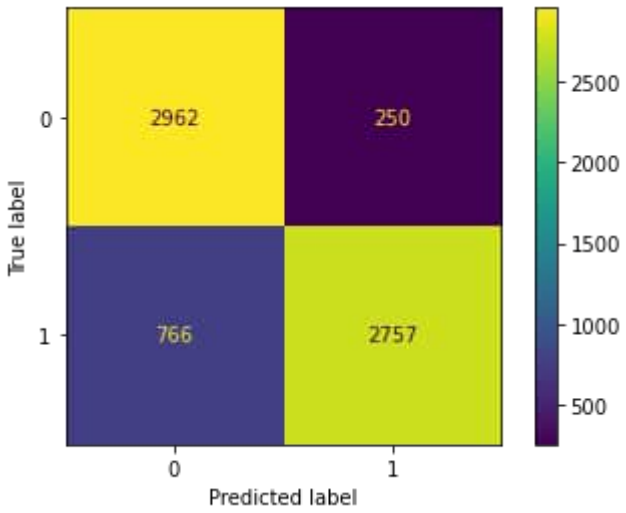

Рисунок 26. Матриця невідповідностей(40000 записів)

Для моделі навченої на меншому датасеті (рис. 27), можна побачити що точність моделі для фейкових новин(label - 1) – 0.58, а для правдивих(label - 0) – 0.45. Чутливість моделі 0.98 і 0.03 відповідно. Можемо побачити, що даних було занадто мало для якісного навчання, і в результаті модель може добре визначати фейкові новини, але погано справляється з правдивими. Тому ця модель всі новини позначає неправдивими.

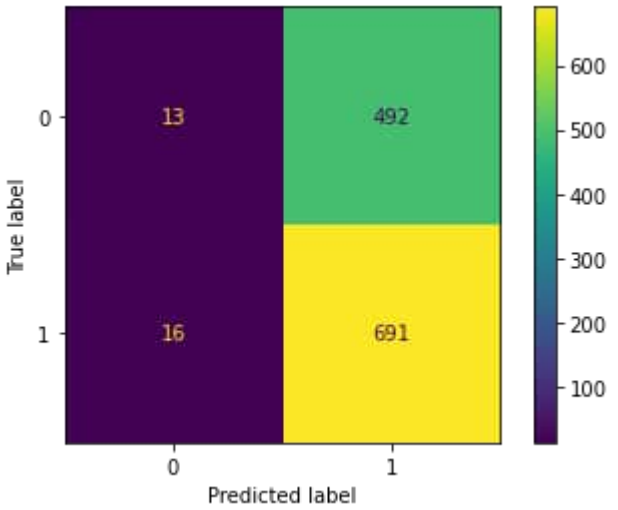

Рисунок 27. Матриця невідповідностей(5000 записів)

Також, ще один варіант оцінки якості класифікації, крива ROC – графік, що відображає співвідношення між часткою об'єктів від загальної кількості носіїв

ознаки, правильно класифікованих до загальної кількості об'єктів, що не несуть ознаки, помилково класифікованих, як такі, що мають ознаку. Загалом, добрий результат можна оцінити, спираючись на такі характеристики ROC-кривої:

1. Чим більше площа під кривою, тим кращий результат моделі.
2. Якщо ROC-крива відхиляється від діагоналі, це означає, що модель виконує класифікацію краще, ніж проста випадкова класифікація.

Видно, що перша модель (рис. 28) відповідає цим характеристикам, що знову доводить що вона навчена добре. І в противагу бачимо другу модель (рис. 29), яка діє як випадкова класифікація.

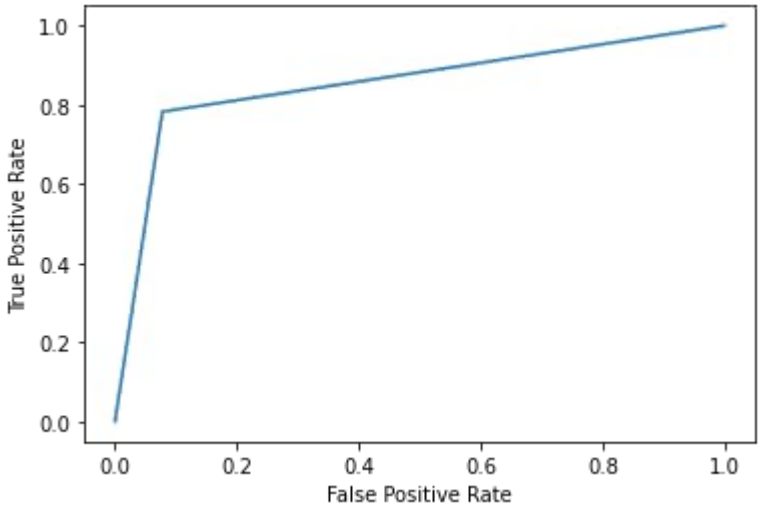

Рисунок 28. ROC-крива(40000 записів)

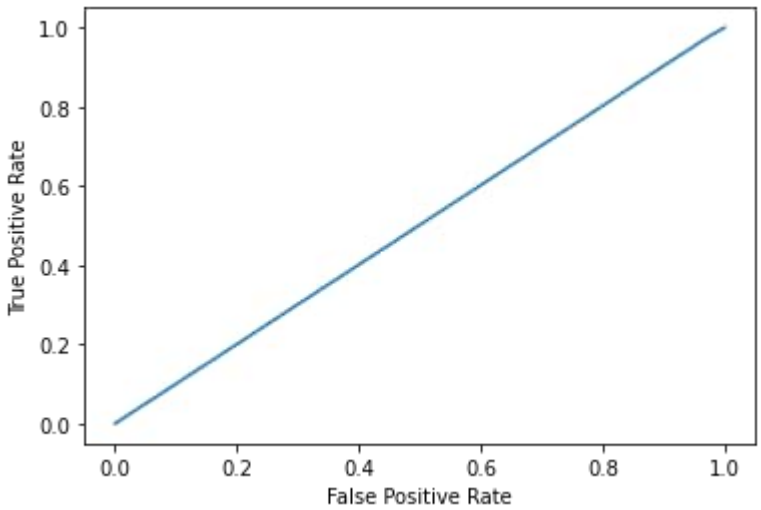

Рисунок 29. ROC-крива(5000 записів)

Порівнюючи метрик модель, можна зробити такий висновок. Якщо модель, натренована на більшому обсязі даних, показує кращі показники метрик (наприклад, вищу точність, чутливість або специфічність), можна стверджувати, що більший обсяг даних допомагає покращити продуктивність моделі. Це може бути обумовлено більшою репрезентативністю та різноманітністю даних, що сприяють кращому навчанню моделі.

Висновки

Під час війни надзвичайно важливо споживати правдиву інформацію. Тому метою моєї роботи було розробити нейронну мережу для класифікації новин на правдиві та фейкові.

Щоб досягти цілі я застосовую емоційний аналіз тональності тексту. Він допомагає розібратися у закономірностях природної мови та навчити комп'ютер сприймати її на рівні, близькому до людського сприйняття.

Нейронні мережі є потужним інструментом для розв’язання задачі аналізу тональностей тексту. Оскільки ця галузь активно розвивається, є дуже багато різних варіантів для різних задач, не тільки для аналізу тональностей. Незважаючи на вибір навчання нейронної мережі (з вчителем чи без), обов’язково потрібно мати правильно створені вхідні дані. Ця частина є однією з найзатратніших при аналізі емоційного забарвлення тексту.

Після збору даних я створила модель BERT, яка була навчена на цих даних для проведення емоційного аналізу та виявлення фейкових новин. Модель проходила тренування та оптимізацію, щоб досягти оптимальних результатів. Проведені дослідження показали, що модель BERT досягає кращих метрик якості при використанні більшого набору даних для тренування.

Це підкреслює важливість наявності великого набору даних при навчанні нейронних мереж. Більший обсяг даних надає моделі більше інформації для вивчення загальних закономірностей та особливостей мови, що дозволяє покращити її здатність до точного аналізу текстів. Завдяки більшому набору даних модель може вивчати більш різноманітні зв'язки та контексти, що позитивно впливає на її здатність до точної класифікації та розуміння тональності тексту.

Отже, результати дослідження показали, що модель BERT демонструє добру ефективність в емоційному аналізі та виявленні фейкових новин. Її здатність аналізувати текст та розрізняти правдиві та фейкові новини стала цінним інструментом для боротьби з дезінформацією, особливо в контексті російсько-української війни.

Джерела

1. Jurafsky, D., & Martin, J. H. (2019). Speech and language processing (3rd ed.). Pearson.
2. Bird, S., Klein, E., & Loper, E. (2009). Natural language processing with Python: Analyzing text with the natural language toolkit. O'Reilly Media, Inc.
3. Manning, C. D., & Schütze, H. (1999). Foundations of statistical natural language processing. MIT Press.
4. Pang, B., & Lee, L. (2008). Opinion mining and sentiment analysis. Foundations and Trends® in Information Retrieval, 2(1-2), 1-135.
5. Turney, P. D. (2002). Thumbs up or thumbs down?: semantic orientation applied to unsupervised classification of reviews. Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, 417-424.
6. Socher, R., Perelygin, A., Wu, J. Y., Chuang, J., Manning, C. D., Ng, A. Y., & Potts, C. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, 1631-1642.
7. Boden M. A (2001) Guide to recurrent neural networks and backpropagation
8. Vaswani, Ashish & Shazeer, Noam & Parmar, Niki & Uszkoreit, Jakob & Jones, Llion & Gomez, Aidan & Kaiser, Lukasz & Polosukhin, Illia. (2017*)* Attention Is All You Need
9. Hewitt, John; Kriz, Reno (2018). Sequence 2 sequence Models
10. Horev, R. (2018, 10 листопада). *BERT Explained: State of the art language model for NLP*. Medium. https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270
11. https://huggingface.co
12. https://jupyter.org/
13. https://voxukraine.org/oglyad-provokatsij-ta-dezinformatsiyi-rf
14. Є. С. Сакало, Аналіз процесів роботи динамічних об'єктів
15. Т. Г. Баган (2021). Методи машинного навчання при проєктуванні автоматизованих систем керування [Електронний ресурс] : навч. посіб. для аспірантів спеціальності 151 «Автоматизація та комп’ютерно-інтегровані технології»