КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики
Кафедра теорії та технології програмування

Кваліфікаційна робота
на здобуття ступеня магістра
за освітньо-науковою програмою «Інформатика»
спеціальності 122 «Комп'ютерні науки»
на тему:

АНАЛІЗ СЕМАНТИЧНОЇ ПОДІБНОСТІ РЕЧЕНЬ З ВИКОРИСТАННЯМ МЕТОДІВ ШТУЧНОГО ІНТЕЛЕКТУ

Виконав студент 2-го курсу магістратури
Онищенко Ігор Орестович _____
(підпис)

Науковий керівник:
Професор, доктор фіз.-мат. наук
Марченко Олександр Олександрович _____
(підпис)

Консультант від кафедри:
Професор, доктор фіз.-мат. наук
Шкільняк Степан Степанович _____
(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Роботу розглянуто і допущено до захисту на засіданні кафедри теорії та технології програмування.
«27» квітня 2021 р.,
Протокол № 10
Завідувач кафедри
Нікітченко М.С. _____
(підпис)

Київ-2021

РЕФЕРАТ

Обсяг роботи 59 сторінок, 11 ілюстрацій, 8 таблиць, 17 джерел посилання.

ГЛИБОКА МОВНА МОДЕЛЬ, МЕТРИЧНЕ НАВЧАННЯ, МЕХАНІЗМ УВАГИ, МІРА ПОДІБНОСТІ, НЕЙРОННА МЕРЕЖА, ПЕРЕФРАЗУВАННЯ, СЕМАНТИЧНА ПОДІБНІСТЬ, ТОНКЕ НАЛАШТОВУВАННЯ МОДЕЛІ.

Об'єктом дослідження є речення природної мови та відношення семантичної подібності між ними.

Метою та завданням даної кваліфікаційної роботи є аналіз сучасних підходів до моделювання семантики слів та речень, що базуються на глибоких нейронних мережах, дослідження та проектування методів визначення перефразувань та семантичної подібності речень, а також експериментальне вимірювання їхньої ефективності шляхом розробки програмної системи.

Методами та засобами дослідження є глибокі мовні моделі на основі архітектури Transformer, окремі методи штучного інтелекту, фреймворк машинного навчання PyTorch та корпуси текстових даних.

Результати роботи продемонстрували значний потенціал глибоких мовних моделей та окремих методів штучного інтелекту для аналізу семантичної подібності речень та моделювання семантики загалом.

Запропоновані та досліджені підходи до визначення семантичної подібності можуть бути застосовані як в практичній сфері, наприклад, в системах виявлення плагіату та різноманітних пошукових системах, так і в науковій сфері для аналізу та систематизації можливостей розглянутих моделей та методів.

Стрімкий розвиток методів штучного інтелекту та актуальність розглянутої задачі спонукають на подальші дослідження у даній галузі.

ЗМІСТ

ВСТУП

Аналіз та визначення семантичної подібності речень є одним з головних завдань в галузі обробки природної мови, особливо в задачах кластеризації, пошуку інформації, генерації підсумків та виявленні плагіату. В загальному випадку задача визначення семантичної подібності полягає у присвоюванні парі речень певного значення подібності. Водночас в даній роботі ми здебільшого розглядатимемо поставлену задачу в бінарному ключі, інакше кажучи, будемо відповідати на питання, чи є два речення семантично ідентичними, тобто перефразуваннями. При цьому відповідь на дане питання найчастіше буде базуватись на певній неперервній мірі подібності та деякому пороговому значенні. Акт перефразування зазвичай визначають як переформатування (або повторне використання) тексту, подаючи те саме значення в іншій формі. Проблема моделювання перефразованих взаємозв'язків між висловлюваннями на природній мові останнім часом привертає усе більшу увагу. Для прикладних лінгвістів вирішення цієї проблеми може пролити світло на те, як найкраще моделювати семантику речення. Для інженерів, що працюють в галузі обробки природних мов, дане завдання має важливе застосування у різних типах реальних проблем, які включать вимірювання семантичного перекриття між реченнями, наприклад у задачах підсумування, генерації машинної відповіді на поставлене запитання та пошуку відповіді у FAQ системах, а також для побудови систем машинного перекладу або систем автоматичної ідентифікації порушення авторських прав тощо. Окрім цього, системи визначення перефразувань можуть використовуватись для видалення дублікатів у текстах, тим самим оптимізуючи використання пам'яті. Загалом, задача визначення та аналізу семантичної подібності речень є однією з важливих проблем галузі обробки природної мови із широкою сферою застосування.

Задача аналізу та визначення семантичної подібності, у нашому випадку визначення перефразувань, в галузі обробки природної мови відноситься до задач класифікації або логістичної регресії. Іншими словами, нехай маємо два речення S_1 та S_2, причому $S_1 \neq S_2$. Нашим завданням буде пошук відповіді на питання, чи правильним є твердження $sem(S_1) = sem(S_2)$. Хоча визначення перефразувань формулюється в семантичних термінах, часто підходи до вирішення даної задачі базуються на статистичних класифікаторах, що використовують поверхневі лексичні та синтаксичні ознаки. Зазвичай в таких підходах для подання речення застосовують моделі bag-of-words, n-грами та TF-IDF, а потім певні методи обчислення подібності (такі як відстань редагування Левенштейна, найдовший спільний підрядок, коефіцієнт Джакарда та косинусна відстань) використовуються для вимірювання міри подібності між двома реченнями. Однак перефразування зазвичай здійснюється шляхом заміни слова синонімами/антонімами, синтаксичної модифікації, скорочення речень, поєднання, реорганізації, перемішування слів, узагальнення понять та специфіки для зміни зовнішнього вигляду оригінального тексту, зберігаючи при цьому семантику вихідного речення, що робить дані підходи малоефективними.

Інші підходи більш детально використовують синтаксичні ознаки, тобто враховують структуру речення. Припускається, що подібні речення мають подібні синтаксичні структури. Однак, дане припущення, що базується на схожості синтаксичних структур без урахування семантики, не може вирішити проблему "однакової семантики, але різних синтаксичних структур".

В останні роки традиційні підходи до моделювання семантики слів та речень поступаються місцем підходам, що базуються на штучних нейронних мережах. Такі моделі здатні вчитись знаходити приховані ознаки і, найголовніше, визначати взаємозв’язки як між словами, так і між реченнями, що є ключовим в задачі визначення семантичної подібності. Глибокі нейронні моделі, на відміну від традиційних підходів, моделюють контекстуальне подання

слів; це означає, що два однакових слова, але вжитих в різних контекстах, будуть мати різне векторне подання.

Метою та завданням даної кваліфікаційної роботи є аналіз сучасних підходів до моделювання семантики слів та речень, що базуються на глибоких нейронних мережах, дослідження та проектування методів визначення перефразувань та семантичної подібності речень, а також експериментальне вимірювання їхньої ефективності шляхом розробки відповідної програмної системи.

Об'єктом дослідження є речення англійської мови та відношення семантичної подібності між ними. *Методами та засобами дослідження* є глибокі моделі природної мови, які базуються на енкодері архітектури Transformer, окремі методи штучного інтелекту, фреймворк PyTorch та корпуси текстових даних для тренування нейронних мереж. Усі експерименти виконувались на одній відеокарті Nvidia GeForce GTX-1060 з 6GB відеопам'яті.

Отримані результати можуть бути застосовані як в практичній сфері, наприклад, в системах виявлення плагіату та різних пошукових системах, так і в науковій сфері для аналізу та систематизації розглянутих моделей та підходів.

РОЗДІЛ І

ГЛИБОКІ МОВНІ МОДЕЛІ

1.1. Порівняння основних підходів до моделювання природної мови

Рекурентні нейронні мережі, зокрема LSTM та GRU, довгий час були визнаними передовими підходами до моделювання послідовностей та проблем трансдукції, тобто їх перетворення, таких як моделювання мови та машинний переклад. Численні дослідження в цій галузі продовжують розширювати можливості рекурентних мовних моделей та архітектур енкодер-декодер.

Рекурентні моделі, як правило, проводять обчислення, спираючись на позиції слів на вхідних та вихідних послідовностях. Узгоджуючи позиції слів з кроками обчислення, вони генерують послідовність прихованих станів h_t, як функцію попереднього прихованого стану h_{t-1} та вхідних даних для позиції t. Такий послідовний характер обчислень унеможливлює паралелізм при навчанні, що стає критичним для послідовностей великої довжини, оскільки обмеження по пам’яті обмежують групування (розмір батчу) тренувальних прикладів. Нещодавні розробки змогли досягти суттєвих покращень обчислювальної ефективності за допомогою прийомів факторизації та умовних обчислення, одночасно покращуючи результативність моделі у випадку останнього. Проте фундаментальне обмеження, що полягає в послідовному обчисленні, залишається. Підсумовуючи, рекурентні моделі мають наступні недоліки:

- Послідовне обчислення унеможливлює паралелізм.
- Відсутнє явне моделювання довгих та коротких залежностей.
- Відстань між позиціями слів є лінійною.

Для вирішення даних проблем був запропонований механізм уваги, який став невід’ємною частиною в моделюванні послідовностей та трансдукції в різних завданнях, який дозволяє моделювати залежності без урахування їх відстані в вхідні або вихідні послідовності. Для рекурентних нейронних мереж замість того, щоб кодувати у прихованому стані лише ціле речення, кожне слово має відповідний прихований стан, який передається до стадії декодування. Ця ідея полягає в тому, що в кожному слові речення може бути певна інформація. Отже, щоб декодування було точним, йому потрібно враховувати кожне слово вхідної послідовності, використовуючи механізм уваги. Але деякі проблеми, які ми обговорювали, досі не вирішуються з використанням механізму уваги для рекурентних мереж. Наприклад, паралельна обробка вхідних слів неможлива. Для великого корпусу тексту це збільшує час роботи моделі.

Згорткові нейронні мережі допомагають у вирішенні цих проблем. Вони дозволяють:

- Розпаралелити процес обчислення по шарах.
- Досліджувати локальні залежності.
- Відстань між позиціями слів є логарифмічною.

Причина, по якій згорткові нейронні мережі можуть працювати паралельно, полягає в тому, що кожне слово на вході може оброблятися одночасно і не обов'язково залежить від попередніх опрацюваних слів. Крім цього, відстань між вихідним словом та будь-яким входом для згорткової нейронної мережі знаходиться в порядку $log(N)$ - це розмір висоти дерева, що генерується з вихідного слово на вхідну послідовність. Це набагато краще, ніж відстань, яку ми отримуємо в рекурентній нейронній мережі, яка має порядок N.

Незважаючи на усі переваги згорткових мереж, суттєвим недоліком є те, що вони вимагають додавання багатьох шарів в нейронну мережу для фіксації довгих глобальних залежностей у послідовній структурі даних, що часто не дає

бажаного ефекту або робить мережу настільки великою, що вона банально стає непрактичною.

Здатність моделі вивчати залежності між позиціями слів вхідної послідовності швидко зменшується з відстанню, що робить критичним пошук іншого підходу, який може розпаралелювати ці послідовні дані та вивчати залежності між словами, незважаючи на відстань. Вирішити дану проблему здатні глибокі нейронні мережі типу Transformer, які використовують механізми уваги, відмовившись від будь-якої рекурентності.

1.2. Попередня обробка та токенізація

Перш ніж приступати до розгляду архітектури Transformer, розглянемо процес попередньої обробки слів і речень та перетворення їх у певний формат, що підходить для взаємодії з нейронними мережами.

Для того, щоб нейронна мережа могла обробляти будь-який текст, записаний природною мовою, нам потрібно розбити його на певні складові, які будуть для неї зрозумілі. Ось тут і з’являється концепція токенізації в галузі обробки природних мов. Простіше кажучи, ми не можемо працювати з текстовими даними без попередньої токенізації. Проте токенізація – це не лише процес поділ тексту на частини, вона загалом грає значну роль при обробці текстів.

Токенізація – це спосіб поділу фрагмента тексту на менші одиниці, які називаються токенами або лексемами. Тут токени можуть бути як словами, символами, так і підсловами. Отже, токенізацію можна класифікувати загалом на 3 типи - токенізація на рівні слів, символів та підслів (n-грам).

Оскільки токени є будівельними елементами природної мови, найпоширеніший спосіб обробки вхідного тексту відбувається на рівні токенів.

Токенізація – це основний базовий крок під час моделювання текстових даних. Токенізація проводиться на певному текстовому корпусі для отримання токенів. Після цього вони використовуються для підготовки словника. Словник – це множина унікальних токенів даного корпусу. Загалом словник можна побудувати, розглядаючи кожен унікальний токен у корпусі або беручи до уваги лише найбільш вживані. Одним з найпростіших методів підвищення результативності мовної моделі є створення словника з K найпоширеніших слів.

Традиційні підходи, такі як Count Vectorizer та TF-IDF, використовують словник в якості ознак, тобто кожне слово в словнику розглядається як унікальний вектор ознак. В глибоких моделях в галузі обробки природної мови словник використовується для перетворення вхідного речення в послідовність токенів – індексів відповідних слів в словнику, які подаються на вхід нейронній мережі.

Як обговорювалося раніше, токенізація може виконуватися на рівні слова, символу або підслова. Тому виникає питання, який тип токенізації слід використовувати під час розв’язання певної задачі в галузі обробки природної мови?

Токенізація на рівні слів - це найбільш часто використовуваний алгоритм токенізації. Він розбиває фрагмент тексту на окремі слова на основі певного роздільника. Залежно від роздільників формуються різні лексеми на рівні слова. Такі векторні подання слів, як Word2Vec та GloVe, використовують саме цей тип токенізації. Проте у такого підходу присутні деякі суттєві недоліки.

Однією з головних проблем при токенізації на рівні слів є робота зі словами, що які відсутні у словнику (Out Of Vocabulary (OOV) words). Термін OOV слова відноситься до нових слів, які зустрічаються під час тестування, тобто їх не існує в поточному словнику. Отже, даний тип токенізації не справляється з обробкою OOV слів. Проте був запропонований певний вихід з цієї ситуації. Його суть полягає у тому, щоб сформувати словник за допомогою

найбільш вживаних слів і замінити рідкісні слова в начальному корпусі даних невідомими токенами типу UNK (unknown). Це допомагає моделі вивчити подання OOV слів в термінах токена UNK. Таким чином, під час тестування будь-яке слово, якого немає у словнику, отримає тип UNK. Проблема цього підходу полягає в тому, що вся інформація про слово втрачається, оскільки ми перетворюємо OOV слово на UNK токен. Структура слова може бути корисною для його точного подання. І ще один недолік полягає в тому, що кожне OOV слово отримує однакове значення у словнику.

Інша проблема з токенізацією на рівні слів пов’язана з розміром словника. Як правило, мовні моделі навчаються на великому за обсягом корпусі тексту. Тому побудова відповідного словника з кожним унікальним словом зробить його розмір надто великим і непрактичним для використання. Усі ці недоліки відкривають двері для токенізації на рівні символів.

Токенізація на рівні символів розділяє текст на множину символів. Це дозволяє подолати недоліки, які присутні в токенізації на рівні слів. Такий тип токенізації послідовно обробляє OOV слово, розбиваючи його на символи та зберігаючи інформацію про нього. Також при такому підході розмір словника обмежується кількістю букв в алфавіті відповідної природної мови.

Така символьна токенізація вирішує проблему OOV слів, але довжина вхідних та вихідних речень швидко зростає, оскільки ми представляємо речення як послідовність символів. Як результат, стає складнішим моделювати зв’язки між символами, щоб утворити змістовні слова. Цей недолік підводить нас до іншого типу токенізації, відомого як токенізація на рівні підслів, яка є поєднанням двох попередніх.

Токенізація на рівні підслів розбиває фрагмент тексту на підслова (n-грами). Наприклад, такі слова, як *higher* та *debug*, можна розділити як *high-er* та *de-bug*. Сучасні мовні моделі на базі Transformer використовують саме цей тип токенізації для побудови навчального словника. Одним із найпопулярніших

алгоритмів токенізації на рівні підслів є Byte Pair Encoding (BPE) представлений в [1].

BPE ефективно справляється з проблемами, які присутні при використанні попередніх токенізацій:

- Ефективно обробляє OOV слова: він розділяє їх на підслова та подає як набір відповідних токенів.
- Довжина вхідних та вихідних речень після BPE менша порівняно з токенізацією на рівні символів.

Загалом алгоритм BPE можна подати наступним чином:

1. Розділити слова в корпусі, додавши символ </w> до кожного, на символи.
2. Ініціалізувати словник унікальними символами в корпусі.
3. Обчислити частоту появи пари символів або послідовностей символів у корпусі.
4. Об’єднати та додати в словник найбільш вживану пару в корпусі.
5. Повторити кроки 3-4 для певної кількості ітерацій або до певного розміру словника.

Під час тестування, коли зустрічається OOV слово, відбувається схожа процедура об’єднання символів та підслів даного слова, проте відповідні пари беруться з уже побудованого словника.

Іншими популярними алгоритмами токенізації на рівні підслів є WordPiece [2], Unigram [3], SentencePiece [4].

Підсумовуючи, токенізація та побудова словника є важливим інструментом для попередньої обробки тексту та абсолютно необхідною при роботі з глибокими мовними моделями.

1.3. Архітектура глибокої нейронної мережі Transformer

Архітектура Transformer (рисунок 1), запропонована в [5], стала своєрідним каноном в сучасній галузі обробки природної мови. Ідеї та підходи, описані в оригінальній статті, є базовими елементами в побудові численних сучасних глибоких мовних моделей.

Більшість сучасних нейронних моделей для перетворення послідовностей має структуру енкодер-декодер. Енкодер відображає вхідну послідовність символів $(x_1, \dots, x_n)$ в послідовність символів $z = (z_1, \dots, z_n)$. З послідовності z декодер генерує вихідну послідовність $(y_1, \dots, y_m)$ по символу за раз. На кожному кроці модель є авторегресивною, тобто кожен попередньо згенерований символ стає додатковим вхідним символ для генерації наступного.

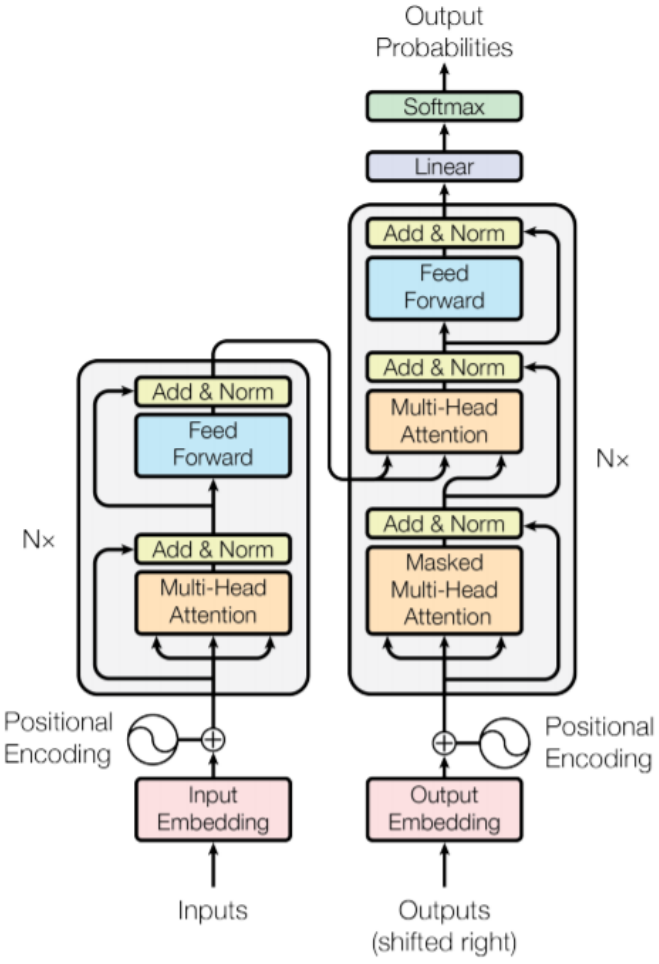

Рисунок 1 - Архітектура Transformer

Модулі Енкодер та Декодер

Енкодер: Енкодер складається з $N = 6$ однакових шарів. Кожен шар містить два підшари. Перший - це мультиголовий механізм самоуваги, а другий – повнозв'язна мережа прямого поширення. До кожного з підшарів застосовується залишковий зв'язок (residual connection), запропонований в [6], з подальшою нормалізацією шару. Тобто, виходом кожного підшару є $LayerNorm(x + Sublayer(x))$, де $Sublayer(x)$ – власне результат самого підшару. Для узгодженності залишкових зв'язків, усі підшари та вкладення (embeddings) мають однакову розмірність виходів d_{model} = 512.

Декодер: Декодер також складається з $N = 6$ однакових шарів. Окрім двох підшарів, як в енкодера, декодер містить ще третій підшар, який реалізує мультиголовий механізм уваги над виходом енкодера. Аналогічно, до кожного з двох підшарів застосовується залишковий зв'язок з подальшою нормалізацією шару. Модифікація механізму самоуваги, яка полягає в операції маскування, відповідного підшару в декодері та зсув вихідних вкладень на одиницю вправо забезпечують те, що результат моделі для позиції i залежить лише від попередніх позицій, менших за i.

Механізм уваги

Функцію уваги можна описати як відображення вхідних даних, які складаються із запитів та наборів пар ключ-значення, на множину виходів, де запити, ключі, значення та вихід представлені векторами. Вихідний результат обчислюється як зважене середнє значень, де ваговий коефіцієнт, присвоєний кожному значенню, задається функцією сумісності для запиту та відповідного ключа.

Механізм уваги, запропонований авторами, отримав назву "Масштабований скалярний механізм уваги (Scaled Dot-Product Attention)" (рисунок 2). На вхід подаються запити та ключі розмірності d_k та значення розмірності d_v. Обчислюється скалярний добуток запиту з усіма ключами та

ділиться на $\sqrt{d_k}$, після цього застосовується функція софтмакс (softmax) для одержання вагових коефіцієнтів вхідних значень.

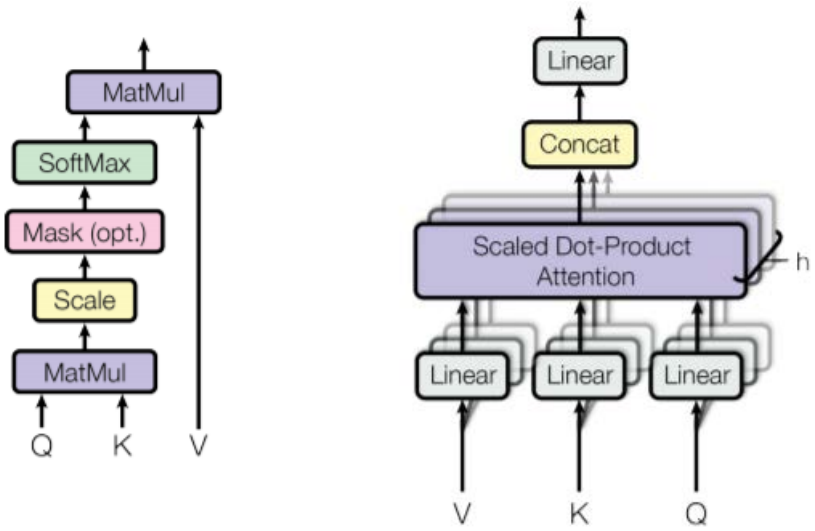

Рисунок 2 - Масштабований скалярний механізм уваги та мультиголовий механізм уваги

На практиці, функція уваги обчислюється на множині запитів одночасно, які утворюють матрицю Q. Ключі та значення також представлені матрицями K та V. Матриця виходів обчислюється наступним чином:

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

При великих значеннях d_k, скалярний добуток запиту та ключа набуває все більших абсолютних значень, витісняючи функцію softmax в регіони, де вона має надзвичайно малі градієнти. Щоб протидіяти цьому ефекту, скалярний добуток масштабується коефіцієнтом $\frac{1}{\sqrt{d_k}}$.

В запропонованій архітектурі обчислюється не одна функція уваги з ключами, значеннями та запитами розмірності d_{model}, а відбувається лінійне проектування запитів, ключів та значень h разів з різними тренованими лінійними проекціями на розмірності d_k, d_k та d_v відповідно. Після цього паралельно обчислюється функція уваги від усіх спроектованих запитів, ключів та значень. Потім результати конкатенуються, відбувається ще одна лінійна проекція і, в підсумку, отримуємо остаточні значення (рисунок 2). Описаний

механізм називається мультиголовим механізмом уваги. Він дозволяє моделі одночасно отримувати інформацію з різних підпросторів на різних позиціях. Наступні формули описують даний механізм:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_\text{h})W^O$$

$$\text{head}_\text{i} = \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right)$$

де $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}, W_i^K \in \mathbb{R}^{d_{model} \times d_k}, W_i^V \in \mathbb{R}^{d_{model} \times d_v}$ та $W^O \in \mathbb{R}^{hd_v \times d_{model}}$.

У своїй роботі автори використовують $h=8$ паралельних шарів уваги, або голів. Кожен з них має розмірності $d_k = d_v = d_{model}/h = 64$. Завдяки зменшенню розмірності, загальна обчислювальна складність співставна з однією функцією уваги над повною розмірністю.

Transformer використовує мультиголовий механізм уваги у три різні способи:

- В механізмі уваги, реалізованому в шарах декодера, запити приходять від попереднього шару декодера, а ключі та значення від виходу енкодера. Це дозволяє декодеру проглядати усі позиції вхідної послідовності.
- Енкодер містить шари самоуваги, в яких запити, ключі та значення приходять з одного джерела – попереднього шару енкодера. Це дозволяє кожній позиції в енкодері відвідувати усі позиції попереднього шару.
- Схожа ситуація в декодері, де шар самоуваги дозволяє декодеру проглядати усі позиції, включно з поточною. Маскування (рисунок 2) запобігає застосуванню уваги щодо заборонених позицій та зберігає властивість авторегресивності.

Продемонструємо, як працює механізм самоуваги для однієї голови моделі. На рисунку 3 наведена візуалізація розподілення уваги однієї голови енкодера для слова “it”. Як бачимо, механізм уваги фокусується на фразі “The animal” і додає її подання з великим ваговим коефіцієнтом до остаточного подання слова “it”.

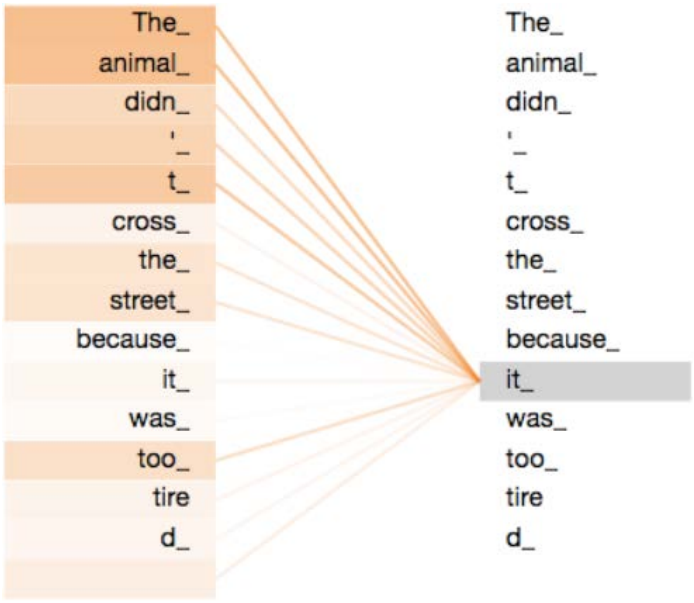

Рисунок 3 - Розподіл уваги для слова "it" в одній голові моделі

Повнозв'язна нейронна мережа прямого поширення

Окрім підшарів з механізмами уваги, кожен з шарів енкодера та декодера містить повнозв'язну нейронну мережу прямого поширення, яка працює з кожним словом окремо та однаково. Вона складається з двох шарів - лінійних перетворень - з функцією активації *ReLU* після першого шару.

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

Хоча дана мережа застосовує однакові лінійні перетворення до кожної позиції, її параметри є унікальними та різними для кожного шару моделі. Розмірність вхідного та вихідного шарів дорівнює $d_{model} = 512$, розмірність внутрішнього шару дорівнює $d_{ff} = 2048$.

Вкладення та функція софтмакс

Подібно до інших моделі в галузі обробки природної мови, Transformer використовує треновані вкладення (embeddings) для вхідних та вихідних послідовностей токенів, отриманих після попередньої обробки тексту та згаданої вище токенізації, щоб перетворити їх у вектори розмірності d_{model}. Вкладення дають нам ефективний кластерний спосіб подання, у якому схожі слова мають

схоже подання. Варто зазначити, що ми не будуємо це подання вручну. Вкладення – це вектор дійсних чисел, розмір якого є гіперпараметром. Значення цього вектору отримуємо в результаті тренування. Модель, отримана в результаті такого тренування, може розглядатися як функція, що ставить у відповідність токену, тобто індексу слова в словнику, вектор заданої розмірності.

На виході декодера застосовується треноване лінійне перетворення та функція Softmax, щоб отримати ймовірності для передбачення наступного слова. В Transformer обидва шари вкладень (енкодера та декодера), а також лінійне перетворення після декодера, але перед функцією Softmax, використовують ту ж матрицю ваг. В шарах вкладень, ці ваги множаться на $\sqrt{d_{model}}$.

Позиційне кодування

Оскільки Transformer не містить рекурентності чи згорток, необхідно, щоб модель мала інформацію про порядок слів у послідовності. З цією метою вводиться “позиційне кодування”, яке дозволить моделі отримати певну інформацію про відносну чи абсолютну позицію слів. Позиційне кодування задається вектором розмірності d_{model} і додається до вхідних вкладень. Вибір позиційного кодування є досить великим, проте автори Transformer використовують функції синус та косинус з різними частотами:

$$PE_{(pos,2i)} = sin\left(pos/10000^{2i/d_{model}}\right)$$

$$PE_{(pos,2i+1)} = cos\left(pos/10000^{2i/d_{model}}\right)$$

де, pos – позиція слова у вхідній послідовності, а i – координата вектору вкладення. Як бачимо, кожна координата позиційного кодування відповідає синусоїді. Довжини хвиль формують геометричну прогресію від 2π до $10000 \cdot 2\pi$. Завдяки такому поданню позиції, модель краще розумітиме відносне розташування слів, бо для будь-якого фіксованого k, PE_{pos+k} може бути представлено як лінійна функція від PE_{pos}. Окрім того, синусоїдне позиційне

кодування дозволить моделі провести екстраполяцію у випадку послідовності, довжина якої більша, ніж тренувальних послідовностей.

1.4. BERT

Автори статті [7] представили мовну модель BERT (**B**idirectional **E**ncoder **R**epresentations from **T**ransformers). BERT призначена для попереднього тренування (pre-train) глибоких двонаправлених подань слів в реченні шляхом врахування контекстів справа та зліва у всіх шарах моделі. В подальшому такі подання можуть бути використанні для кінцевого тренування в конкретних задачах обробки природної мови.

Існують дві стратегії застосування попередньо натренованих подань слів: на основі ознак (feature-based) та тонкого налаштування (fine-tuning). Підхід, на основі ознак, використовує специфічні для задачі архітектури, які розглядають попередньо натреновані подання слів як додаткові ознаки. Підхід тонкого налаштування передбачає додавання мінімальної кількості специфічних для конкретної задачі параметрів та навчання шляхом налаштування всіх попередньо навчених параметрів.

BERT удосконалює стратегію тонкого налаштування, використовуючи “модель маскованої мови” (Masked Language Model (MLM)) в якості цільової функції попереднього навчання. Модель маскованої мови випадковим чином маскує деякі вхідні токени, а завданням навчання є здатність передбачити замасковане слово, використовуючи лише його контекст. На відміну від попереднього навчання однонаправленої (зліва направо) мовної моделі, MLM дозволяє об’єднати лівий та правий контекст, що дозволяє натренувати глибокий двонаправлений Transformer. Окрім MLM, іншою цільовою функцією попереднього навчання є задача “передбачення наступного речення”, яка дозволяє підготувати подання для пар речень.

Розглянемо архітектуру BERT та підходи до її навчання та використання. Можна виділити два основних етапи при роботі з BERT: попереднє навчання та подальше тонке налаштування. На рисунку 4 зображені дані етапи для деяких конкретних задач обробки природної мови.

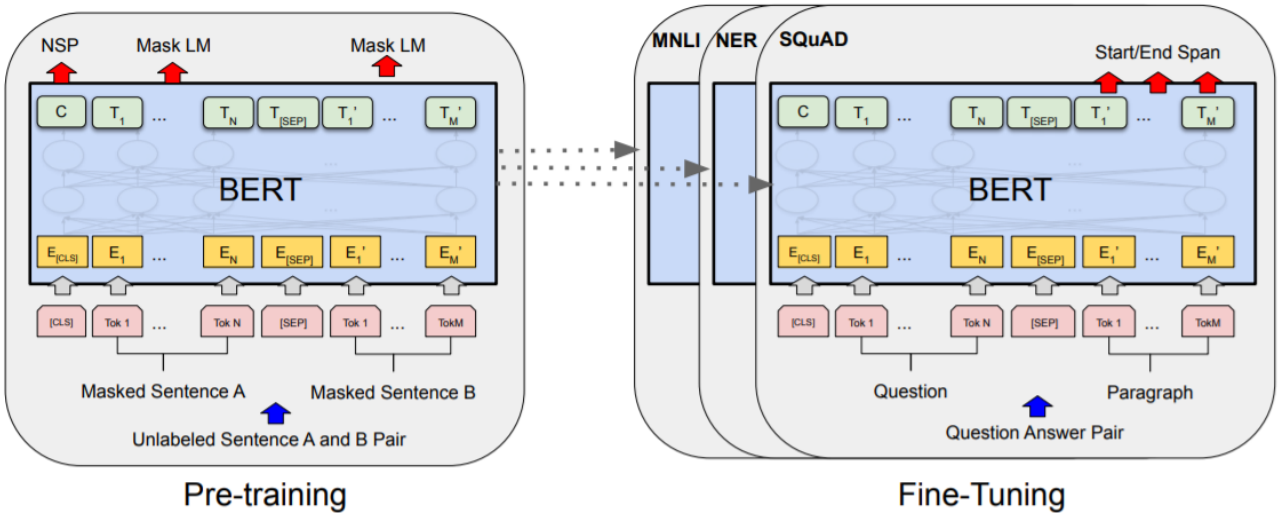

Рисунок 4 - Головні етапи при роботі з BERT

Під час попереднього навчання модель тренується на нерозмічених текстових даних. Для тонкого налаштування BERT спочатку ініціалізується попередньо навченими параметрами і в подальшому всі параметри налаштовуються згідно з розміченими даними для конкретної задачі. Таким чином, в результаті тонкого налаштування для кожної задачі отримуємо окрему модель BERT, хоча кожна з них була однаково ініціалізована.

Архітектура BERT складається з багатошарового двонаправленого енкодера Transformer. В залежності від кількості блоків, розмірності прихованих шарів та кількості голів самоуваги архітектура має різну кількість параметрів. Найбільш вживаними є BERT_{base} та BERT_{large}.

Для того, щоб модель BERT могла використовуватись для різноманітних конкретних задач з різною структурою вхідних даних, однозначне вхідне подання, запропоноване авторами, здатне зображати як одне речення, так і пару речень (наприклад, для визначення семантичної подібності, відповіді на питання

тощо), як одну послідовність токенів. Під терміном “речення” мається на увазі будь-яка послідовність слів, а не лише в лінгвістичному сенсі.

Для токенізації використовується WordPiece з розміром словника 30000 токенів. Першим токеном кожної послідовності завжди є спеціальний класифікаційний токен ([CLS]). Останній прихований стан, що відповідає цьому токену, використовується як узагальне подання вхідної послідовності для подальшої класифікації. Даний токен буде детальніше розглянутий в наступному розділі при моделюванні семантики речення для визначення семантичної подібності. Як було сказано, пара речень об’єднується в одну послідовність. Для того, що модель розрізняла кожне речення, вони розділяються двома наступними способами. Спочатку розділяємо їх спеціальним токеном ([SEP]). Після цього до кожного подання слова додаємо ще одне треноване подання, яке вказує чи належить дане слово реченню (сегменту) A або реченню B. Загалом, вхідне подання слова будується як сума векторного подання відповідного токена, подання сегменту та позиційного кодування.

Як вже згадувалось раніше, для попереднього тренування BERT використовуються цільові функції для двох завдань: модель маскованої мови (MLM) та передбачення наступного речення (Next Sentence Prediction (NSP)).

MLM використовується для навчання двонаправленого подання. Для цього деякий відсоток вхідних токенів випадково маскується і робитись передбачення цих токенів. Генератор тренувальних вхідних послідовностей випадковим чином вибирає 15% токенів. 80% цих токенів замінюються на спеціальний токен [MASK], 10% - на випадковий інший токен, а решта 10% залишається без змін. Після цього, фінальний прихований стан кожного такого токена застосовується для передбачення оригінального токена, використовуючи в якості функції втрат перехресну ентропію.

Багато задач в галузі обробки природної мови базуються на розумінні відношень між реченнями, як і у випадку визначення семантичної подібності.

Для того, щоб модель BERT розуміла відношення між реченнями, використовується цільова функція для бінарного передбачення наступного речення. Тренувальні приклади можна легко отримати з послідовного корпусу тексту. Коли обираємо речення A та B, у 50% випадків B дійсно є наступним реченням після A і позначаємо таку пару IsNext, в інших 50% випадків вибираємо довільне речення з корпусу і позначаємо NotNext.

Корпусом тренувальних даних послужили BooksCorpus (800 мільйонів слів) та English Wikipedia (2.5 мільярди слів).

Для тонкого налаштування на вхід BERT подаються конкретні вхідні та вихідні послідовності, які залежать від поточної задачі. Виходи моделі подаються на шар класифікації або передбачення. Після цього будується конкретна цільова функція та починається процес тренування.

Хоча потужність BERT полягає саму в тонкому налаштуванні моделі під конкретну задачу, підхід, що базується на основі отриманих з попередньо тренованої моделі ознак, теж має свої переваги. По-перше, не усі задачі в галузі обробки природної мови можуть бути оформлені в термінах архітектури Transformer та вимагають специфічної додаткової архітектури. По-друге, велика обчислювальна перевага полягає в тому, щоб попередньо зберегти отримані з моделі векторні подання тренувальних даних та проводити експерименти з не такими громіздкими архітектурами. Даний підхід стане ключовим при моделюванні семантичної подібності речень.

Підсумовуючи, останні емпіричні вдосконалення завдяки механізму передачі знань (transfer learning) за допомогою мовних моделей продемонстрували, що попереднє навчання без вчителя є невід’ємною частиною багатьох систем розуміння природної мови. BERT є прикладом попередньо навченої глибокої моделі з двонаправленою архітектурою, яка здатна успішно вирішувати широкий спектр задач в галузі обробки природної мов.

1.5. RoBERTa

Модель BERT стала каталізатором появи нових моделей, які базуються на її архітектурі та використовують підходи та принципи, запропоновані в оригінальній статті. Тренування таких моделей є досить дорогим задоволенням, яке вимагає значних обчислювальних потужностей, величезних корпусів текстових даних та багато годин експериментів для вдалого підбору гіперпараметрів. Автори статті [8] провели дослідження щодо впливу усіх цих факторів на ефективність оригінальної моделі BERT. Вони виявили її прихований потенціал, який дозволяє змагатись на рівних з більш новими моделями. Було проаналізовано важливість вибору відповідного дизайну та його наслідки для остаточної ефективності моделі.

Багато моделей, які з'явились після BERT, продемонстрували значний приріст ефективності, проте є проблематичним визначити, які саме аспекти запропонованих методів, мали найбільший вплив. Тренування є обчислювально дорогим та довготривалим процесом, що обмежує достатню кількість експериментів задля визначення оптимальних параметрів. Крім того, тренування відбуваються з використанням приватних корпусів текстових даних різних розмірів, що ускладнює якісну оцінку запропонованих підходів.

Автори [8] запропонували підхід до тренування моделей на архітектурі BERT, який вони назвали RoBERTa (**R**obustly **o**ptimized **BERT a**pproach). Даний підхід базується на досить простих модифікаціях: довший час тренування з більшим розміром батчу та на більшому корпусі тексту; відмова від цільової функції NSP; тренування на довших вхідних послідовностях; динамічне маскування, яке застосовується до тренувальних даних. Також був запропонований новий великий корпус текстових даних CC-News, який дозволяє контролювати ефекти розміру тренувальних даних. Розглянемо кожну з модифікацій детальніше.

Оригінальна реалізація BERT використовувала статичне маскування вхідних послідовностей, тобто процес маскування відбувався на етапі попередньої обробки даних. Щоб уникнути використання однакових масок під час кожної епохи, автори BERT продублювали вхідні дані десять разів так, щоб кожна вхідна послідовність маскувалась десятьма різними способами. Таким чином, наприклад, при тренуванні моделі протягом сорока епох кожна послідовність зустрічалась чотири рази з тією ж маскою.

Автори RoBERTa запропонували динамічне маскування, при якому маска генерується на етапі формування вхідного батчу. Це стає важливим при тренуванні протягом більшої кількості епох або з використанням більших корпусів даних.

В оригінальній процедурі тренування BERT на вхід моделі подаються два об’єднані сегменти тексту, які з ймовірністю ½ були взяті неперервно з одного документу або різних. Окрім MLM, модель додатково тренується передбачати чи ці сегменти були взяті з одного документа чи ні, мінімізуючи похибку відповідної цільової функції – Next Sentence Prediction. Припускалось, що відсутність даної цільової функції погіршує ефективність моделі на задачах, які працюють з парами речень. Проте пізніше було поставлено під питання необхідність такого рішення. Щоб краще зрозуміти суть проблеми, розглянемо декілька форматів тренування:

- SEGMENT-PAIR+NSP: даний формат відповідає оригінальній статті про BERT. Кожна вхідна послідовність складається з пари сегментів, які можуть містити кілька речень, проте загальна довжина послідовності не перевищує 512 токенів.
- SENTENCE-PAIR+NSP: кожна вхідна послідовність складається лише з двох речень. Оскільки вхідні послідовності значно коротші ніж 512 токенів, тому розмір вхідного батчу було збільшено так, щоб загальна кількість токенів залишалась співмірною з SEGMENT-PAIR+NSP.

- FULL-SENTENCES: кожна вхідна послідовність складається з повних речень, які беруться з одного або кількох документів так, щоб загальна довжина не перевищувала 512 токенів. Межі документів можуть перетинатись. Якщо ми досягли кінця документу, то далі беруться речення з наступного документу, додаючи відповідний токен-роздільник. NSP відсутня.
- DOC-SENTENCES: вхідні послідовності будуються подібно до FULL-SENTENCES, проте межі документів не перетинаються. Якщо беруться речення з кінця документа, довжиною менше ніж 512 токенів, то розмір батчу динамічно збільшується, щоб отримати загальну кількість токенів як в FULL-SENTENCES. NSP відсутня.

Провівши експерименти з даними форматами, автори RoBERTa отримали наступні результати.

Порівнюючи SEGMENT-PAIR та SENTENCE-PAIR, можна стверджувати, що використання індивідуальних речень погіршує ефективність моделі на подальших задачах. Можна припустити, що модель не в змозі вивчити залежності на довгих відстанях.

Результати тренувань з DOC-SENTENCES показали, що відсутність NSP не заважає моделі демонструвати таку ж, і навіть дещо кращу, точність як і оригінальна BERT.

Обмеження на вибір речень з одного документу (DOC-SENTENCES) показує дещо кращу точність ніж з різних (FULL-SENTENCES). Проте динамічний розмір батчу створює певні труднощі при тренуванні та об’єктивному порівнянні з іншими моделями.

В галузі машинного навчання тренування з міні-батчами великого розміру пришвидшує процес збіжності та покращує загальну точність моделі. Тому автори RoBERTa провели дослідження щодо впливу розміру тренувального батчу на підсумкову ефективність даної мережі. Оригінальна BERT тренувалась

з міні-батчем розміру 256. RoBERTa показала кращі результати з міні-батчем розміру 2048 та 8192. Крім того, такі великі розміри сприяють більш ефективному дистрибутивному навчанню.

Також варто згадати, що автори RoBERTa збільшили розмір словника токенів з 30 тисяч до 50 тисяч, використовуючи дещо модифіковану версію токенізації, яка є в BERT.

Таким чином, застосування усіх вище згаданих технік допомогло покращити ефективність оригінальної моделі BERT, створивши її модифікацію – RoBERTa. Тренування моделі протягом більшої кількості кроків, збільшення розміру тренувального батчу, відмова від цільової функції NSP та динамічне маскування вхідної послідовності – це ключові моменти, які дозволяють оригінальній моделі BERT залишатись на рівні з більш сучасними суперниками.

1.6. ALBERT

Збільшення розміру моделі при попередньому тренуванні для отримання векторних подань слів часто покращує точність моделі на конкретних задачах. Проте в певний момент подальше збільшення моделі стає складнішим з врахуванням обмежень пам’яті в графічних картах та часу тренування. Щоб подолати дані обмеження, автори [9] запропонували дві техніки, які дозволяють зменшити кількість параметрів моделі BERT та пришвидшити процес тренування, зберігаючи і навіть покращуючи її точність.

Результати багатьох досліджень показали, що велика за розмірами модель здатна досягти небачених раніше результатів. Звичною практикою стало тренування таких моделей з подальшим переданням її знань меншим за розміром моделями для розв’язання реальних проблем. Тому виникає наступне питання:

Чи для отримання більш ефективної моделі в галузі обробки природної мови достатньо лише збільшити її розмір?

Проблемою для відповіді на це питання є обмеження розміру пам’яті сучасного апаратного забезпечення. Найкращі мовні моделі часто використовують сотні мільйонів та навіть мільярдів параметрів, що призводить до виходу за ці обмеження при спробі збільшити їх розмір. Час навчання теж може зазнати суттєвих обмежень при дистрибутивному тренуванні, адже витрати на комунікування прямо пропорційно залежать від кількості параметрів. Запропонована авторами [9] архітектура ALBERT (A Little BERT) містить значно менше параметрів, що дозволяє ефективно боротись з описаними вище проблемами.

ALBERT пропонує дві техніки зменшення кількості параметрів. Перша техніка полягає у факторизації матриці ваг вкладень. Розкладаючи велику матрицю вкладень словника на добуток двох менших матриць, ми відділяємо розмір внутрішнього шару моделі від розміру вкладення у словнику. Таке розділення допомагає збільшити розмір прихованих шарів без значного збільшення розміру вкладень словника. Друга техніка полягає у використанні тих самих параметрів у різних шарах. Це дозволяє збільшувати глибину моделі, не додаючи нових параметрів. Обидві техніки значно зменшують кількість параметрів моделі BERT, зберігаючи її точність та підвищуючи ефективність кожного параметра. Також дані техніки відіграють роль додаткової регуляризації, яка стабілізує тренування та сприяє узагальненню.

Автори також пропонують нову цільову функцію SOP (Sentence-Order Prediction), яка головним чином зосереджується на узгоджуваності між реченнями. Дана цільова функція була спроектована на противагу неефективній цільовій функції NSP, яка була запропонована в оригінальній BERT.

Розглянемо ключові елементи ALBERT більш детально.

В BERT та наступних моделях на її основі вкладення кожного токену в словнику мають розмір E, і цей розмір прив'язаний до розміру прихованого шару H, тобто $E = H$. Дане рішення є не до кінця оптимальним для моделювання та практичного застосування.

З точки зору моделювання, вкладення зі словника мають на меті представляти незалежні від контексту подання слів. На противагу цьому, вкладення з прихованих шарів представляють залежні від контексту подання слів. Дослідження продемонстрували, що потужність BERT полягає саме у використанні контексту, тобто побудові контекстуальних вкладень. Таким чином, при $H \gg E$ ми зможемо більш ефективно використовувати параметри моделі.

З точки зору практичного застосування, обробка природної мови вимагає, щоб розмір словника V був достатньо великим, для BERT $V = 30000$. Якщо $E = H$, то збільшення H збільшує розмір матриці вкладень словника, яка має розмір $V \times E$. Це може призвести до великої кількості параметрів, які будуть рідко оновлюватись під час тренування.

Враховуючи це, ALBERT використовує факторизацію матриці вкладень, розкладаючи її на дві менші. Спочатку проектуємо одиничний вектор розмірності V у простір меншої розмірності E, а потім проектуємо отриманий вектор в простір розмірності H. Завдяки такій декомпозиції, ми можемо зменшити кількість параметрів матриці вкладень з $O(V \times H)$ до $O(V \times E + E \times H)$. Зменшення кількості параметрів особливо суттєве при $H \gg E$.

Іншим способом підвищення ефективності використання параметрів для ALBERT є використання тих самих параметрів у різних шарах – так званий обмін параметрів між відповідними шарами. Такий обмін може бути здійснений багатьма способами, наприклад, лише для параметрів мережі прямого поширення між усіма шарами, або лише для блоків з механізмами уваги. Автори ALBERT використовують обмін усіх параметрів між усіма шарами.

Запропонований підхід також стабілізує параметри моделі, забезпечуючи більш гладкий перехід між шарами.

Як уже було сказано раніше, оригінальна BERT використовує додаткову цільову функцію NSP, яка має на меті покращити ефективність моделі на деяких конкретних задачах. Проте подальші дослідження, наприклад, авторами RoBERTa, продемонстрували неефективність NSP. В свою чергу, автори ALBERTA припускають, що ця неефективність обумовлена відсутністю достатньої складності, порівнюючи з MLM. NSP, в тому вигляді як вона сформульована, об'єднує два завдання в одне: передбачення теми та передбачення порядку послідовності сегментів тексту, бо негативні пари складаються з сегментів з різних документів, які мають різні теми. Передбачення теми є значно простішим у порівнянні з передбачення порядку сегментів, і також перетинається з тим, що може бути вивчене моделлю з допомогою MLM.

Для ALBERT пропонується нова цільова функція SOP (Sentence-Order Prediction), яка базується саме на порядку сегментів. Позитивні пари будуються так само, як і в BERT (два послідовні сегменти з одного документу), а негативні пари використовують ті самі послідовні пари, проте в оберненому порядку. Такий підхід допомагає моделі краще розуміти порядок речень, а це в свою чергу покращує точність моделі на деяких конкретних задачах.

Підсумовуючи, запропоновані авторами ALBERT техніки дозволили значно підвищити ефективність оригінальної моделі BERT, зменшивши кількість параметрів та покращивши результати на багатьох задачах в галузі обробки природної мови.

РОЗДІЛ ІІ

МОДЕЛЮВАННЯ СЕМАНТИКИ РЕЧЕННЯ

2.1. Постановка завдання

Під моделюванням семантики речення розглядатимемо побудову скінченновимірного векторного простору $S \subseteq R^n$, в якому кожне речення представлено елементом даного простору. Основною властивістю отриманого простору є те, що два семантично подібні речення знаходяться близько один від одного в термінах певної міри подібності. Таким чином, для визначення семантичної подібності нам буде достатньо отримати векторне подання кожного речення і порівняти їх за допомогою визначеної міри подібності. Ключовим для практичного застосування даного підходу є відсутність обчислювальної складності при використанні обраної міри подібності.

Для побудови даного простору ми використовуватимемо контекстуальні подання слів кожного речення, отримані за допомогою глибоких мовних моделей на основі Transformer, а саме – BERT, RoBERTa та ALBERT. В якості міри подібності використовуватимемо косинус подібності, а також запропонуємо деякі інші, які базуватимуться на нейронних мережах. Крім того, для отримання бажаної властивості простору, яка полягає в тому, що семантично близькі речення знаходяться поруч, ми розглянемо та запропонуємо певні стратегії навчання моделей, які сприятимуть даному дискримінативному ефекту.

Загальну схему процесу побудови векторних подань двох речень з їх подальшим порівнянням можна побачити на рисунку 5.

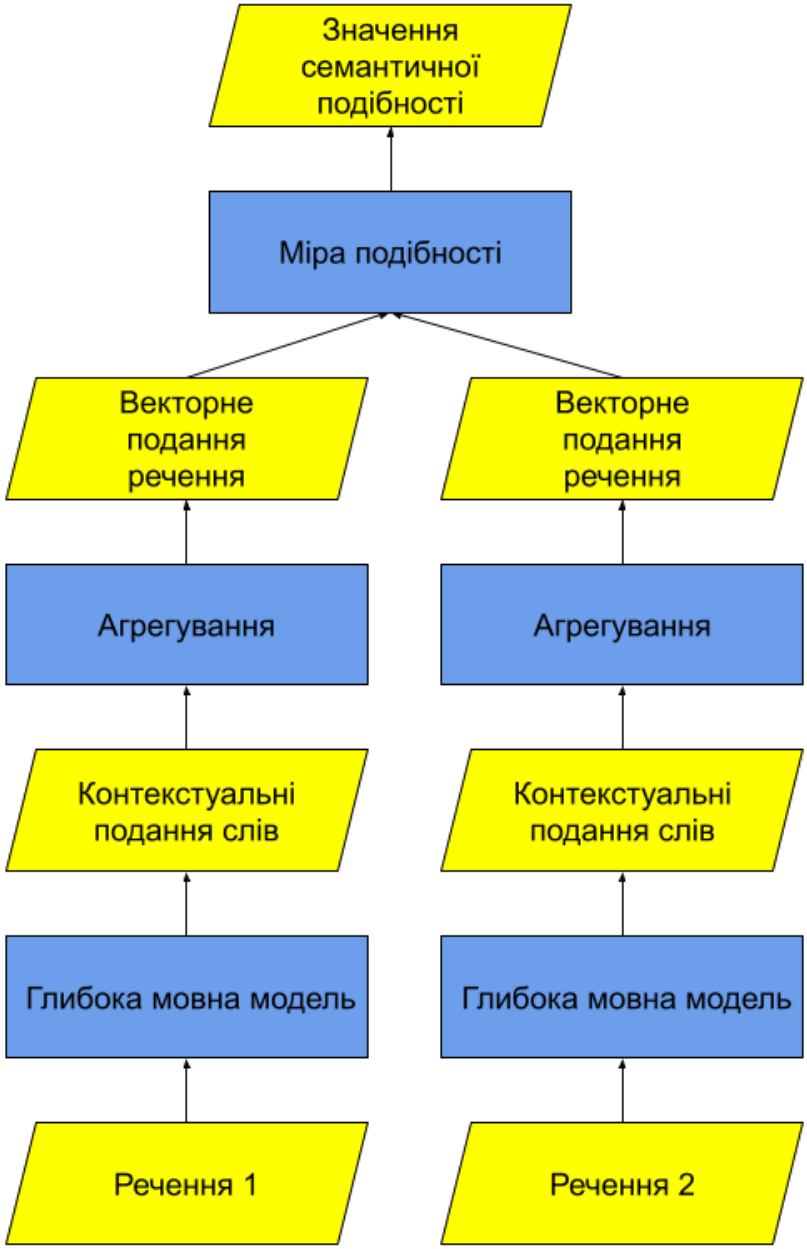

Рисунок 5 - Схема визначення семантичної подібності речень

2.2. Агрегування виходів глибоких мовних моделей

В розділі I ми розглянули глибокі мовні моделі на основі архітектури Transformer, які будуть базовим елементом в моделюванні семантики речення. Кожна подібна модель приймає на вхід послідовність слів, які утворюють

речення. Механізм уваги, який є ключовим в такій моделі, дозволяє побудувати контекстуальне подання кожного слова. З одного боку, таке подання інкапсулює в собі статистичні особливості даного слова в межах певної мови, а, з іншого боку, враховує його роль в контексті даного речення. На відміну від звичайних рекурентних моделей, блок енкодера моделі Transformer, який є основою BERT, RoBERTa та ALBERT, дозволяє нам враховувати контекст кожного слова як зліва, так і справа. Причому, завдяки механізму уваги відстань між словами у реченні не має ніякого значення для ефективного визначення важливості конкретного слова для побудови контексту поточного слова.

Отже, глибока мовна модель перетворює вхідну послідовність слів у послідовність контекстуальних подань відповідних слів. Для отримання векторного подання речення нам потрібно якимось чином агрегувати цю послідовність векторних подань слів. Очевидно, що загалом речення мають різні довжини, тому важливим є те, щоб векторне подання кожного окремого речення мало певний фіксований розмір, який і буде визначати розмірність нашого векторного простору речень.

Найбільш очевидним є усереднення усіх векторних контекстуальних подань слів. У такому випадку розмірність простору речень збігається з розмірністю простору слів. При такому способі агрегування кожне слово є однаково важливим для побудови подання речення, іншими словами, ми враховуємо кожне контекстуальне подання з ваговим коефіцієнтом $1/l$, де l – це довжина речення. Назвемо такий спосіб агрегування MEAN (англ. середнє).

В курсовій роботі було розглянуто зв’язки між словами та їхню граматичну роль у реченні. Було проаналізовано важливість кожного слова для фінального подання речення залежно від частини мови, до якої вони належать. Отримані результати експериментів показали, що здебільшого для найефективнішого подання речення важливими є усі слова в реченні, не залежно від їхньої функціональної ролі. Тому при усередненні усіх векторних контекстуальних подань слів будемо враховувати усі слова з однаковим ваговим коефіцієнтом.

Ще одним методом усереднення може виступати функція максимуму. Іншими слова, ми переглядаємо покомпонентно кожен вектор і вибираємо найбільше значення. Такий спосіб агрегування зарекомендував себе в інших сферах штучного інтелекту, особливо в галузі комп'ютерного зору. Назвемо даний спосіб агрегування MAX.

В оригінальній статті BERT було введено спеціальний токен CLS. Даний токен використовувався при навчанні моделі в контексті цільової функції NSP. Цей токен завжди додається на початок вхідної послідовності. Векторне подання, що відповідає токену CLS, виступає в ролі представника усього речення. Таким чином, саме вихід цього токена використовується для додаткових цільових функцій, які є задачами класифікації, при тренуванні мовних моделей. У випадку BERT – це NSP, а у випадку ALBERT – SOP. В оригінальній статті BERT та інших при подальшому тонкому налаштовуванні цих моделей на конкретні задачі, саме токен CLS використовують в якості представника речення, тобто якраз як векторне подання речення. Ми також розглянемо даний метод агрегування.

2.3. Косинусна міра подібності

Косинус подібності – це міра подібності між двома ненульовими векторам у просторі з визначеним скалярним добутком. Вона дорівнює косинусу кута між ними, що є аналогічним скалярному добутку нормалізованих векторів. З визначення косинуса випливає, що важливим є саме орієнтація векторів, а не їхні абсолютні величини. Однаково напрямлені вектори, тобто кут між ними дорівнює $0°$, мають міру косинусної подібності 1, ортогональні між собою вектори мають міру подібності 0, а протилежно напрямлені вектори мають міру подібності -1, незалежно від їхніх довжин. Зазвичай міра косинусної подібності застосовується в додатному просторі, де значення обмежуються відрізком $[0; 1]$.

В такому випадку, вектори найбільш подібні, якщо вони паралельні, і найменш схожі, якщо вони між собою ортогональні.

Косинус кута між двома векторами можна отримати з визначення скалярного добутку:

$$\mathbf{A} \cdot \mathbf{B} = \|\mathbf{A}\|\|\mathbf{B}\|\cos(\theta)$$

З цього отримуємо визначення міри косинусної подібності:

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

Основною властивістю міри косинусної подібності є те, що вона відображає відносне, а не абсолютне, порівняння між окремими компонентами вектора. Іншими словами, для сталої a та вектора V, вектори V та aV є максимально подібними.

Ще однією важливою перевагою косинусної подібності є її низька обчислювальна складність, особливо для розріджених векторів, бо розглядаються лише ненульові елементи векторів.

Міра косинусної подібності пов’язана з евклідовою відстанню наступним чином. Позначимо евклідову відстань як зазвичай $\|A - B\|$ і розпишемо:

$$\|A - B\|^2 = (A - B)^T(A - B) = \|A\|^2 + \|B\|^2 - 2A^TB$$

Якщо ми нормалізуємо вектори A та B до одиничної довжини, тобто $\|A\|^2 = \|B\|^2 = 1$, то отриманий вираз дорівнюватиме $2\big(1 - \cos(A, B)\big)$.

Дуже часто також розглядають поняття косинусної відстані, яка задається рівністю $D_C(A, B) = 1 - S_C(A, B)$, де S_C – міра косинусної подібності. Надалі в даній роботі косинусна відстань буде використовуватись нарівні з косинусною подібністю. Варто зауважити, що дана відстань не є метрикою, бо не виконується

нерівність трикутника. Для того, щоб ця властивість виконувалась, потрібно розглядати кутову відстань: $arccos(S_C)/\pi$.

Загалом міра косинусної подібності широко застосовується в інформаційному пошуку та інтелектуальному аналізі тексту, де кожен документ подається вектором, кожна координата якого дорівнює кількості входжень певного слова в цьому документі. Таке подання дозволяє ефективно порівнювати документи на основі їхньої тематики. Ще однією галуззю штучного інтелекту, в якій міра косинусної подібності є дуже популярною, – це розпізнавання облич і людей загалом. Кожній людині ставиться у відповідність вектор, і порівнюючи вектори за допомогою косинусної подібності, ми можемо виконати верифікацію особи або визначити чи є певна людина у галереї.

2.4. Міра подібності на основі нейронної мережі

Незважаючи на широку популярність косинусної міри подібності саме в галузі обробки природної мови, варто також розглянути інші способи визначення міри подібності двох речень. Сьогодні нейронні мережі визнані потужним та ефективним інструментом у багатьох прикладних задачах, тому в даній роботі також розглянемо та побудуємо методи визначення міри подібності на основі нейронних мереж.

Загалом визначення міри подібності двох речень можна переформулювати в задачу регресії або класифікації. Задача регресії полягає у визначенні дійсного значення подібності, а задача класифікації розглядається у випадку, коли парі речень ми приписуємо певний клас, наприклад, в задачі визначення перефразувань, де в нас є два класи. Під задачею бінарної класифікації дуже часто мається на увазі задача логістичної регресії. Загалом задачу класифікації можна розглядати як задачу регресії з подальшим введенням певних порогових значень, тобто дискретизацією вихідних значень.

Таким чином, маючи два векторних подання речень ми будемо розв’язувати задачу регресії або класифікації для визначення їхньої семантичної подібності. У даній роботі розглядаються лише випадки логістичної регресії та класифікації. Оскільки, логістична регресія є окремим випадком класифікації, то назвемо нейронну мережу, яка буде обчислювати міру подібності, класифікатором. Розглянемо властивості класифікатора, необхідні для практичного застосування.

Перш за все, кількість параметрів класифікатора повинна бути набагато меншою, ніж кількість параметрів мовної моделі. Також важливим є час обчислення і кількість необхідної для цього пам’яті. В протилежному випадку, важкий і повільний класифікатор буде неефективним в практичному застосуванні.

Загалом процес класифікації можна розбити на три етапи:

1. Побудова вхідного векторного подання, яке є певною комбінацією уже агрегованих подань двох речень, отриманих з глибокої мовної моделі.
2. Обчислення нейронної мережі на побудованому векторному поданні.
3. Інтерпретація результату нейронної мережі.

Розглянемо кожен з цих етапів трішки детальніше.

На вхід класифікатору ми отримуємо два векторні подання речень, позначимо їх a та b. Наступним кроком буде перетворення цих векторів у єдине подання, яке подаватиметься на нейронну мережу. В [10] було запропоновано використовувати об’єднаний вектор самих вхідних подань a і b разом з результатами операцій над ними, наприклад покомпонентне додавання, віднімання, множення тощо. Іншими словами, одним з вхідних векторів може виступати $(a, b, |a - b|, a * b)$. В даній роботі розглянемо ефективність різноманітних подібних подань.

Отримане комбіноване векторне подання речень після цього йде на вхід штучній нейронній мережі. Розглянемо класичну логістичну регресію з одним шаром, а також розглянемо архітектури з різною кількістю параметрів, кількістю шарів та функцій активацій. Для навчання ваг розглянемо декілька стратегій тренування.

У випадку класичної класифікації результатом обчислення нейронної мережі буде вектор, довжина якого дорівнює кількості класів. Кожен елемент цього вектору відповідає за кількісну міру належності вхідних даних певному класу. В такому випадку до цього вектору застосовуватимемо функцію $softmax$, щоб отримати ймовірнісний розподіл передбачення мережі по класах, і функцію $argmax$ для визначення остаточного класу.

У випадку логістичної регресії виходом нейронної мережі завжди буде одне число, тобто одновимірний вектор. Для отримання міри подібності використовуватимемо сигмоїду, яка переводить свій аргумент в число з інтервалу $(0; 1)$. Для визначення класу у випадку бінарної класифікації використовуватимемо певне порогове значення.

Отже, побудований таким чином класифікатор після тренування можна буде використовувати для визначення семантичної подібності речень на основі їхніх векторних подань, отриманих з глибоких мовних мереж.

2.5. Метричне навчання. Сіамські та триплетні мережі

Тренування нейронних мереж полягає в оптимізації певної цільової функції, яка ще називається функцією втрат. Зазвичай йдеться про мінімізацію похибки передбачення мережі відносно правильної мітки вхідного об’єкту. В галузі штучного інтелекту така стратегія тренування називається навчанням з учителем (supervised learning). Популярними функціями втрат є перехресна

ентропія (cross-entropy) для класифікації і логістичної регресії та середня квадратична помилка (mean squared error) для регресії. Такі цільові функції використовуються коли потрібно напряму передбачити мітку, значення або набір значень для вхідного об’єкту. У випадку визначення семантичної подібності нам потрібно передбачити відносну відстань або подібність між двома вхідними реченнями. Така стратегія тренування широко поширена і в інших сферах штучного інтелекту, де потрібно якимось чином порівнювати об’єкти між собою. Ця концепція називається *метричним навчанням*, а цільові функції, які використовуються в такому випадку, називаються *контрастними функціями втрат* [11].

Задача визначення семантичної подібності речень якраз відповідає принципам метричного навчання. Ми спочатку отримуємо векторні подання речень завдяки глибокій мовній моделі. Потім визначаємо функцію порівняння, тобто міру подібності, наприклад, косинус подібності. Після цього ми тренуємо глибоку мовну модель генерувати подібні векторні подання для семантично схожих речень, і різні подання для несхожих. Загалом нам навіть неважливий вигляд і значення цих векторних подань, лише відстань між ними, проте метричне навчання дозволяє отримати ефективне подання для багатьох задач.

Контрастні функції втрат є досить гнучкими в термінах навчальних даних. Нам лише потрібна міра подібності між вхідними об’єктами, і ця міра подібності може бути бінарною, як у випадку визначення перефразувань, проте це все одно дозволяє отримати неперервну (в межах певного проміжку) міру подібності для тестових даних.

Загалом виділяють два основних підходи до побудови контрастної функції втрат:

- Використання пар тренувальних об’єктів – *парна функція втрат*.
- Використання трійок (триплетів) тренувальних об’єктів – *триплетна функція втрат*.

Парна функція втрат. Сіамські нейронні мережі

Розглянемо детальніше перший підхід. Він передбачає використання позитивних та негативних пар тренувальних об'єктів. Позитивні пари утворюються з анкерного об'єкту x_a та позитивного об'єкту x_p, який є схожим з x_a в термінах визначеної нами міри подібності, а негативні пари утворюються з анкерного об'єкту x_a та негативного об'єкту x_n, який не є схожим з x_a.

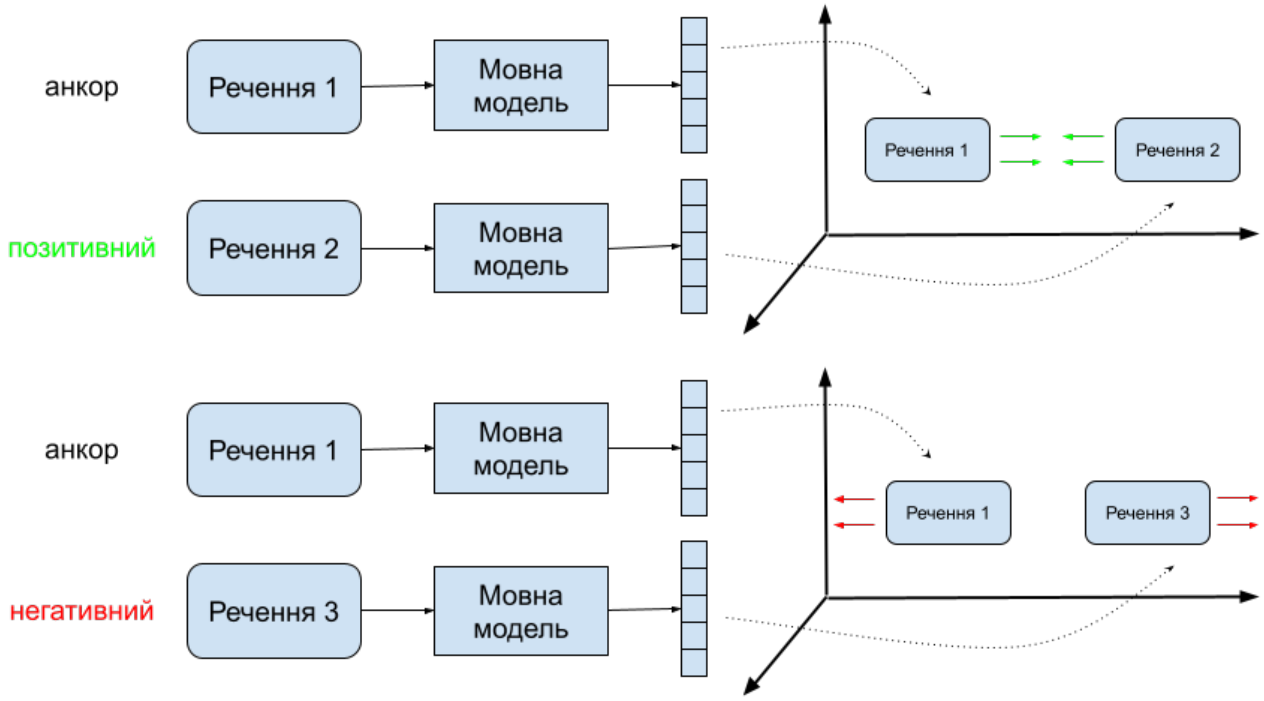

Рисунок 6 - Схема тренування з парною функцією втрат

Метою навчання є отримання таких векторних подань, відстань d між якими є якомога меншою для позитивних пар, та більшою за певну маржу m для негативних пар. Парна функція втрат змушує векторні подання мати нульову відстань для позитивних пар, та відстань більшу за певну маржу для негативних пар. Позначимо відповідні векторні подання вхідних об'єктів як r_a, r_p та r_n, тоді парну функцію втрат можна записати наступним чином:

$$L = \begin{cases} d(r_a, r_p) & if \quad PositivePair \\ max(0, m - d(r_a, r_n)) & if \quad NegativePair \end{cases}$$

Для позитивних пар функція втрат зростатиме (і відповідна величина оновлення ваг мережі) при збільшенні відстані між об’єктами. Якщо відстань між об’єктами рівна нулю, то і функція втрат рівна нулю.

Для негативних пар функція втрат рівна нулю лише тоді, коли відстань між об’єктами перевищує маржу m. Коли відстань є меншою за m, то функція втрат є додатною, що сприятиме оновленню ваг мережі з метою генерації якомога несхожих векторних подань. Найбільше значення функції втрат дорівнює m, коли відстань між r_a та r_n дорівнює нулю. Метою існування цієї маржі m є те, що, коли відстань для негативної пари є достатньою, ми не витрачаємо зусилля для оновлення ваг мережі і можемо зосередити тренування на більш складних парах.

Найчастіше парна функція втрат використовується тоді, коли векторні подання отримані за допомогою ідентичних глибоких нейронних мереж зі спільними вагами. У такому випадку говоримо, що тренування відбувається на основі *сіамських нейронних мереж.*

Триплетна функція втрат. Триплетні нейронні мережі

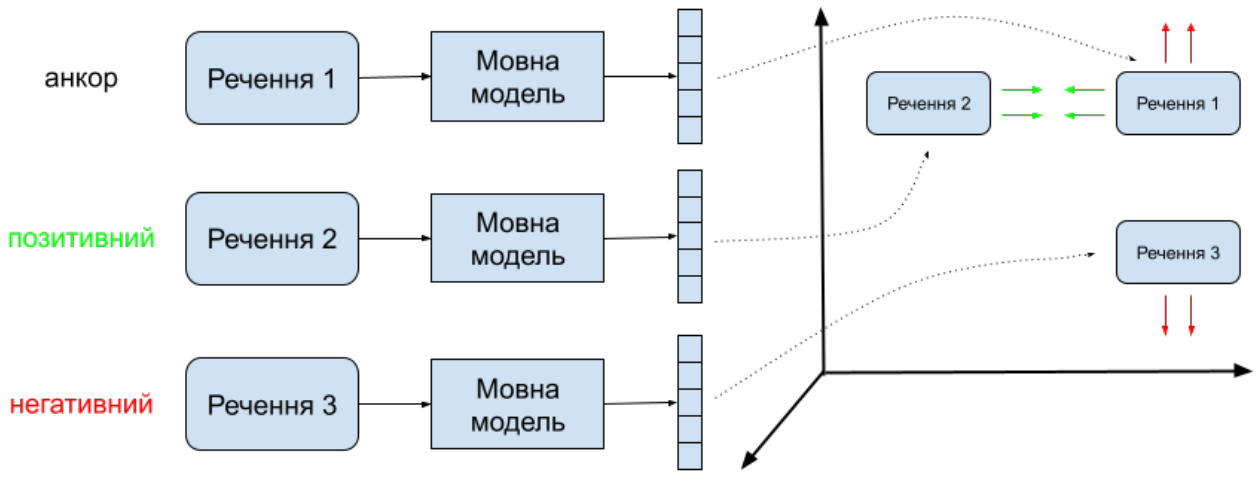

Рисунок 7 - Схема тренування з триплетною функцією втрат

Контрастна функція втрат на основі трійок (триплетів) тренувальних об’єктів в багатьох випадках демонструє кращий результат, ніж парна функція втрат. Триплети формуються анкором x_a, позитивним x_p та негативним x_n

об’єктами. Даний підхід базується на тому, що відстань між анкором та негативним об’єктом є більшою (з певною маржею m), ніж відстань між анкором і позитивним об’єктом. В загальному випадку, триплетну функцію втрат можна записати наступним чином:

$$L(r_a, r_p, r_n) = max(0, m + d(r_a, r_p) - d(r_a, r_n))$$

Розглянемо три можливі ситуації при використанні даної функції втрат.

- **Легкі триплети**: $d(r_a, r_n) > d(r_a, r_p) + m$. Негативний об’єкт уже достатньо далеко від анкора у порівнянні з позитивним об’єктом. У такому випадку функція втрат рівна нулю і ваги мережі не оновлюються.
- **Важкі триплети**: $d(r_a, r_n) < d(r_a, r_p)$. Негативний об’єкт знаходиться ближче до анкора, ніж позитивний. Функція втрат є додатною і більшою за m.
- **Напівважкі триплети**: $d(r_a, r_p) < d(r_a, r_n) < d(r_a, r_p) + m$. Негативний об’єкт знаходиться дальше від анкора, ніж позитивний, але відстань не перевищує маржу, тому функція втрат все одно є додатною, але меншою за m.

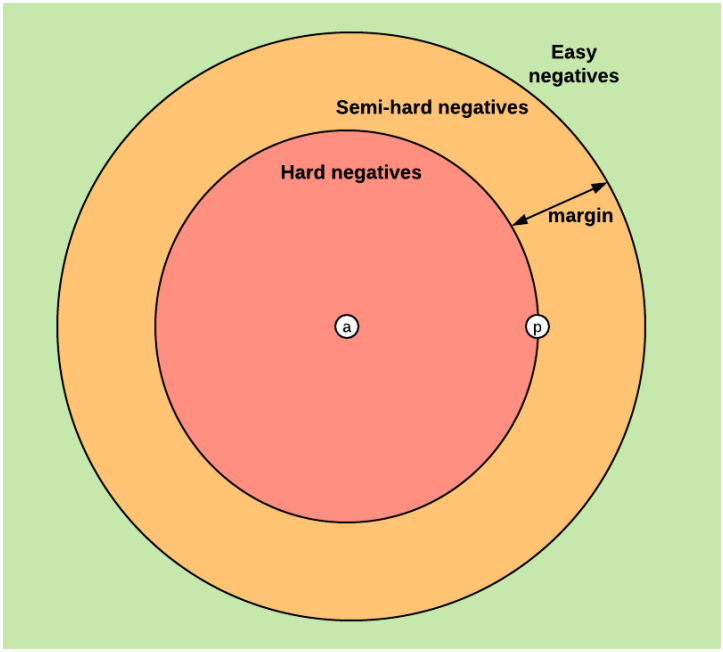

Рисунок 8 - Три типи негативних об'єктів для заданих анкора та позитивного об'єкту. Джерело: [12]

Важливим при тренуванні з триплетною функцією втрат є вибір негативних об’єктів, типи яких можна зобразити на рисунку 8. Вибрана стратегія відбору матимемо значний внесок у процес тренування та фінальний результат. Очевидним рішенням буде уникати легких триплетів, адже функція втрат уже рівна нуля. Одні стратегії базуються на виборі триплетів на початку тренування або на початку кожної епохи, інші стратегії формують триплети для кожного батчу на кожній ітерації тренування. Оптимальна стратегія відбору негативних триплетів дуже залежить від конкретної задачі.

Подібно до парної функції втрат, триплетна функція втрат найчастіше використовується тоді, коли векторні подання отримані за допомогою ідентичних глибоких нейронних мереж зі спільними вагами. У такому випадку говоримо, що тренування відбувається на основі *триплетних нейронних мереж.*

2.6. Тонке налаштовування BERT-подібних мовних моделей

Основним призначенням BERT є її подальше тонке налаштовування під конкретну задачу в галузі обробки природної мови. BERT є попередньо натренованою моделлю, яка здатна за допомогою численних шарів з механізмом уваги витягувати певні ознаки та властивості слів і моделювати зв’язки між ними в межах речення. І саме процес тонкого налаштовування дозволяє ефективно інтерпретувати ці ознаки і взаємозв’язки для розв’язання конкретних задач. Такий підхід на сьогоднішній день демонструє найкращі результати у багатьох задачах на різноманітних тестових корпусах. Це стосується і задачі визначення семантичної подібності, і задачі визначення перефразувань. Тому варто розглянути використання BERT для подальшого тонкого налаштування для задач, які мають справу з парами речень.

Перш за все, автори BERT передбачили використання своєї моделі для задач з парами речень, що було описано в розділі І. Коротко нагадаємо, що пара

речень об’єднується в єдину вхідну послідовність токенів, розділену службовим токеном. Окрім позиційного кодування, додатково розглядається треноване подання, яке відповідає за належність токена до одного з двох речень. Загалом приклад побудови вхідної послідовності для пари речень поданий на рисунку 9.

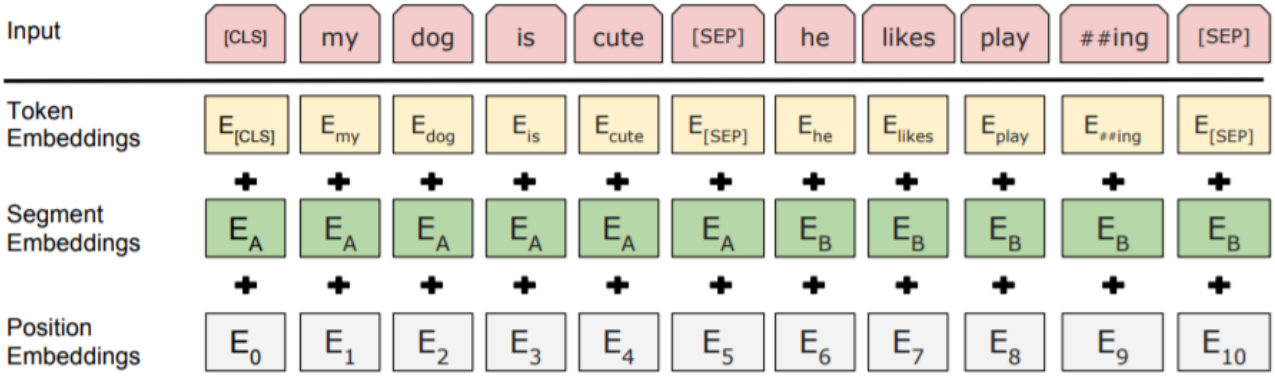

Рисунок 9 - Вхідна послідовність для пари речень

Окрім цього, автори BERT, RoBERTa, ALBERTA та інших подібних моделей використовували додаткову цільову функцію для попереднього тренування, щоб інкорпорувати в модель певне розуміння взаємозв’язків між реченнями.

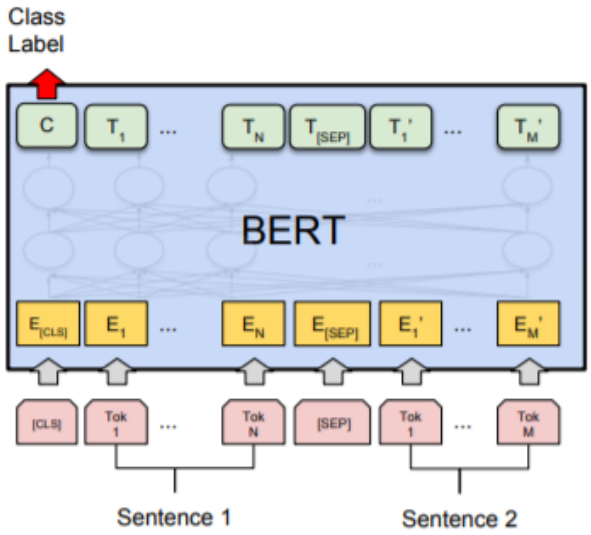

Рисунок 10 - Схема тонкого налаштування BERT для задач, які оперують парами речень

Процес подальшого тонкого налаштування моделі полягає в отриманні певного векторного подання пари речень з подальшим його використанням для класифікації чи регресії. Автори BERT в якості такого подання пропонують

використовувати фінальне приховане подання, яке відповідає токену [CLS], як це показано на рисунку 10. Це подання автори одразу подають на класифікаційний чи регресійний шар. Після цього відбувається кілька епох тренування для тонкого налаштування під конкретну задачу.

Незважаючи на те, що описаний підхід демонструє найкращі результати на багатьох тестових корпусах для різних задач, що працюють з парами вхідних речень, він вимагає, щоб обидва речення разом подавались на вхід моделі. В цьому випадку, ми отримуємо значні обчислювальні витрати. Автори [13] дослідили, що для пошуку найбільш семантично подібної пари серед $n = 10000$ речень, потрібно виконати $n \cdot (n-1)/2 = 49995000$ запусків моделі BERT. Для виконання такого завдання потрібно приблизно 65 годин на сучасній графічній карті V100. Як бачимо, такий тривалий час пошуку пари подібних речень є неадекватним для практичного застосування в епоху, коли кількість нової інформації зростає експоненційно. Крім того, такий підхід не дозволяє отримати явне векторне подання окремого речення.

Таким чином, незважаючи на високу точність моделі BERT, отриману внаслідок тонкого налаштування для конкретної задачі, даний підхід не може бути застосовний для вирішення реальних масштабних задач, таких як визначення семантичної подібності, кластеризації та пошуку інформації серед величезної кількості вхідних об’єктів. Саме тому, в даній роботі ми розглядаємо методи побудови семантичного подання кожного речення окремо, яке можна буде ефективно порівнювати з іншими. Проте, запропонований в BERT метод одночасного тренування з парою речень, в якості єдиної вхідної послідовності, може посприяти у покращенні отриманих результатів.

РОЗДІЛ ІІІ

РЕЗУЛЬТАТИ ЕКСПЕРИМЕНТІВ

3.1. Корпус текстових даних

Для тренування та тестування запропонованих підходів будемо використовувати класичний для задач визначення семантичної подібності речень корпус текстових даних Microsoft Research Paraphrase Corpus (MRPC). Даний корпус складається з пар речень $\{S_1, S_2\}$ з відповідними мітками, які вказують на те, чи є ці речення перефразуваннями (мітка “1”) чи ні (мітка “0”). MRPC був отриманий напівавтоматичним шляхом з новинних статей, розміщених в інтернеті. Автори корпусу спочатку застосували SVM класифікатор, передбачення якого були схильні до помилкового визначення перефразування, до 49375 пар речень. Після цього випадковим чином було обрано 5801 пару речень, які були надані людям для ручного визначення мітки. Оскільки чітке визначення перефразування полягає в ідентичному значенні двох речень, то майже завжди це зводиться до ідентичного вигляду. Проте, автори, бажаючи дослідити цікавіші та складніші залежності між реченнями, трішки послабили умову ідентичності значень. Настановою в визначенні перефразування для людей, які займались розміткою, було те, щоб значення речень були певною мірою ідентичними, але разом з тим описували ті ж події і зберігали усю важливу інформацію про них. Кожна пара речень отримувала мітку від двох людей, які у 83% випадків погоджувались між собою, а в інших випадках третя людина вирішували конфлікт. Таким чином, значення 83% можна вважати верхньою границею точності автоматичних систем визначення перефразувань [14].

Статистична характеристика MRPC наведена в таблиці 1.

Таблиця 1 - Статистика MRPC. Джерело: [15]

	Навчальна вибірка	Тестова вибірка
Позитивні	2753 (67,5%)	1147 (66,5%)
Негативні	1323 (32,5%)	578 (33,5%)
Разом	4076	1725
≤ 20 слів	2,40%	2,78%
20-50 слів	86,86%	86,03%
> 50 слів	10,77%	11,19%
Max довжина	60	63
Min довжина	14	12

Приклад речень, які є перефразуванням (мітка “1”):

- The increase reflects lower credit losses and favorable interest rates.
- The gain came as a result of fewer credit losses and lower interest rates.

Приклад речень, які не є перефразуванням (мітка “0”):

- By Sunday night, the fires had blackened 277,000 acres, hundreds of miles apart.
- Major fires had burned 264,000 acres by early last night.

3.2. Метрики оцінювання

Задача визначення перефразувань, як було сказано раніше, належить до задач бінарної класифікації. В цьому випадку найбільш популярним вибором метрик оцінювання якості моделі є точність (accuracy) та F1-score. Остання дозволяє отримати більш якісну оцінку моделі, коли тестовий корпус не є збалансованим, як у випадку MRPC.

Точність моделі визначається наступним чином:

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FN} + \text{FP} + \text{TN}},$$

де:

- TP (true positive) – кількість правильно класифікованих позитивних.
- TN (true negative) – кількість правильно класифікованих негативних.
- FP (false positive) – кількість хибно класифікованих позитивних.
- FN (false negative) – кількість хибно класифікованих негативних.

Описані вище класи об’єктів, отриманих в результаті класифікації, можна схематично зобразити на рисунку 11 (зліва).

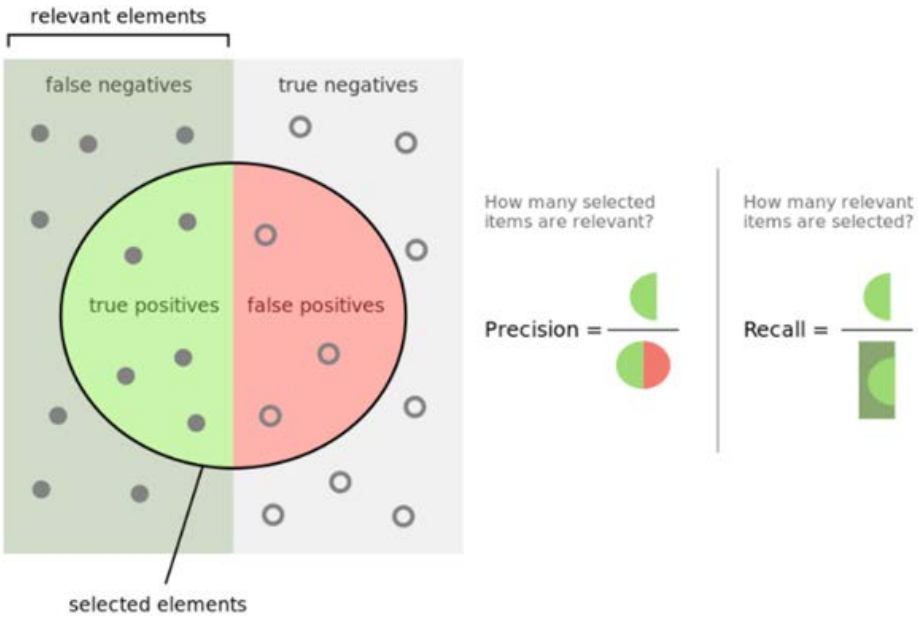

Рисунок 11 – Класи об’єктів при класифікації. Метрики. Джерело: [16]

Метрика F1-score є середнім гармонічним двох інших популярних метрик: precision та recall:

$$F1 = \frac{2 * \text{recall} * \text{precision}}{\text{recall} + \text{precision}},$$

де

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \qquad \text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}.$$

Ілюстративне пояснення значень цих двох метрик можна побачити на рисунку 11 (справа). Кожна з цих метрик може використовуватись окремо в залежності від мети, яку ставлять перед задачею класифікації. Якщо ціна хибно позитивного результату велика, то від класифікатора вимагають високі значення precision. На противагу цьому, у випадку високої ціни хибно негативного результату максимізують recall, також відому як сенситивність (sensitivity). Метрика F1-score використовуються ж тоді, коли нам потрібен баланс цих двох показників.

3.3. Перехресна валідація

У випадку метричного навчання та логістичної регресії для визначення перефразування нам потрібно підібрати порогове значення для класифікації. З метою справедливої оцінки роботи моделі на тестовому корпусі та для визначення її здатності до узагальнення, вибір порогового значення відбуватиметься з допомогою перехресної валідації (cross-validation). У загальному випадку, цей метод використовують, коли корпус даних не є достатньо великим для адекватного поділу на тестову та навчальну вибірки. У випадку з MRPC, нам уже надані готові вибірки для тренування та тестування. Саме тому, ми будемо використовувати перехресну валідацію лише на тестовій вибірці для отримання менш зміщеної оцінки порогового значення та відповідних метрик.

Отже, метод перехресної валідації, який ми будемо використовувати, полягає в наступному. Тестовий корпус ділиться на приблизно K рівних частин (folds). $K-1$ частина (умовно тренувальні дані) використовується для визначення найкращого, в термінах метрики точності, порогового значення, яке потім оцінюється на останній K-ій частині (умовно тестові дані). Такий процес повторюється K разів, після чого усі отримані порогові значення та оцінки

метрик усереднюються і подаються як остаточний результат тестування моделі. Важливо зауважити, що поділ на частини фіксується на початку і не змінюється в процесі роботи методу. Це дозволяє кожному об'єкту корпусу бути один раз у тестовій вибірці та $K-1$ разів у тренувальній. Також рекомендується випадковим чином перемішати корпус перед поділом на частини.

Вибір K є дуже важливим для адекватного оцінювання. Погано підібране значення K може призвести до хибної оцінки моделі з великою дисперсією або зі зміщенням. Найбільш популярним вибором є $K = 10$, хоча не існує якогось формального правила. При збільшенні K зменшується зміщення оцінки, проте зростає дисперсія.

3.4. Умови навчальних експериментів

Розробка програмної системи для проведення навчальних експериментів проводилась з використанням мови програмування Python та популярного фреймворку для глибокого машинного навчання PyTorch від компанії Facebook. Серед переваг даного фреймворку варто виділити простоту у використанні, динамічний обчислювальний граф та ефективний менеджмент пам'яті. Крім того, PyTorch тісно інтегрований з мовою Python, що сприяє більш ефективній та зручній розробці.

Усі експерименти відбувались на графічній карті Nvidia GeForce GTX 1060 з 6GB відеопам'яті. Для прискорення тренувань використовувалась бібліотека NVIDIA CUDA Deep Neural Network (cuDNN), яка надає можливості більш ефективного та швидкого виконання базових операцій в глибоких нейронних мережах на графічних картах.

Доступ до глибоких мовних моделей та відповідних ваг здійснювався з використанням бібліотеки HuggingFace [17], яка містить десятки сучасних

мовних моделей та надає уніфікований API. Через обмеження пам'яті для експериментів було обрано наступні версії моделей:

- $BERT_{base}$: 12 шарів, розмірність векторного подання – 768, 12 голів в мультиголовому механізмі самоуваги, 109 мільйонів параметрів.
- $RoBERTa_{base}$: 12 шарів, розмірність векторного подання – 768, 12 голів в мультиголовому механізмі самоуваги, 125 мільйонів параметрів.
- $ALBERT_{base}$: 12 однакових шарів, розмірність векторного подання – 768, 12 голів в мультиголовому механізмі самоуваги, 11 мільйонів параметрів.

Для усіх експериментів розмір батчу дорівнював 12. Кожне тренування здійснювалось протягом 30 епох, кожна з яких складалась з 80 ітерацій. У випадку класифікації та тонкого налаштовування, додатковий блок (голова) нейронної мережі тренувалась протягом перших 10 епох при заморожених вагах мовної моделі.

В якості оптимізатора було обрано стохастичний градієнтний спуск (Stochastic Gradient Descent (SGD)) з імпульсом (momentum) Нестерова з параметром 0,9. Початкове значення кроку оновлення ваг (learning rate) залежало від конкретної моделі та цільової функції, проте загальна стратегія зміни кроку полягала в його зменшенні на коефіцієнт 0,5 кожні 5 епох.

Для регуляризації моделі було обрано L2 регуляризацію з параметром 0,01.

У випадку сіамських та триплетних мереж значення маржі m дорівнювало 0,5 для усіх експериментів.

Код програмної системи для проведення навчальних експериментів знаходиться на https://github.com/i-onyshchenko/ParaphraseDetection.

3.5. Результати та аналіз навчальних експериментів

Стартові результати мовних моделей

Для розуміння ефективності тренування та використання певних цільових функцій в таблиці 2 наведено стартові результати моделей на попередньо тренованих вагах для різних типів агрегування виходів. В якості міри подібності було використано косинус подібності.

Таблиця 2 - Стартові результати (точність/ F1) мовних моделей

	MEAN	MAX	CLS
BERT	0.659/0.789	0.659/0.785	0.664/0.797
RoBERTa	0.681/0.791	0.661/0.788	0.664/0.797
ALBERT	0.690/0.788	0.683/0.792	0.665/0.798

Як бачимо, результати майже збігаються з випадковим класифікатором, проте можна сказати, що типи агрегування MEAN та CLS демонструють трішки кращі показники метрик.

Метричне навчання

Глибокі мовні моделі попередньо натреновані для виділення різноманітної кількості ознак з вхідної послідовності слів. Для розв'язання конкретної задачі потрібно правильно інтерпретувати ці ознаки, надавати більшу вагу одним та відкидати інші. Метричне навчання дозволяє моделі краще використовувати певні ознаки для визначення семантичної подібності. В таблиці 3 наведено результати тренування моделей в парадигмі сіамських мереж.

Таблиця 3 - Результати (точність/ F1) сіамських мереж

	MEAN	MAX	CLS
BERT	0.692/0.798	0.684/0.789	0.697/0.801
RoBERTa	0.693/0.797	0.686/0.790	0.701/0.799
ALBERT	0.691/0.799	0.684/0.787	0.702/0.811

Якщо порівняти з результатами з таблиці 2, то бачимо очевидне покращення для усіх моделей та типів агрегування.

В таблиці 4 наведемо результати тренування з використанням підходу на основі триплетних мереж.

Таблиця 4 - Результати (точність/F1) триплетних мереж

	MEAN	MAX	CLS
BERT	0.709/0.809	0.692/0.795	0.729/0.818
RoBERTa	0.709/0.807	0.691/0.799	0.720/0.812
ALBERT	0.694/0.793	0.676/0.797	0.715/0.795

Порівнюючи з таблицею 3, очевидно, що стратегія тренування на основі триплетних мереж демонструє значно кращі результати, ніж на основі сіамських мереж. Це підтверджується дослідженнями і в інших галузях штучного інтелекту.

Логістична регресія

Для визначення семантичної подібності за допомогою міри подібності на основі нейронної мережі було досліджено кілька типів об’єднань векторних подань речень, розглянуто варіанти класифікаційної мережі з різними кількостями шарів та параметрів. На основі результатів численних експериментів, які не продемонстрували суттєвих відмінностей у використанні багатьох внутрішніх шарів, було вирішено зупинитись на мережі з однією матрицею ваг, тобто одним вихідним шаром. Обираючи між softmax класифікацією та логістичною регресією, які продемонстрували схожі результати бінарної класифікації, вибір впав на логістичну регресію, яка дозволяє природніше інтерпретувати результат мережі, саме як міру подібності. В якості функції втрат було обрано бінарну перехресну ентропію.

Для вибору типу об’єднання векторних подань було проведено експерименти з моделлю BERT та типом агрегування CLS. Результати та типи

розглянутих об’єднань наведені в таблиці 5, де u, v – векторні подання речень, операція “ $*$ ” означає покомпонентне множення векторів.

Таблиця 5 - Результати (точність/F1) для різних типів об'єднань

Тип об’єднання	BERT (CLS)
(u, v)	0.671/0.782
$(\|u - v\|)$	0.689/0.791
$(u * v)$	0.698/0.795
$(u, v, \|u - v\|)$	0.722/0.812
$(u, v, u * v)$	0.719/0.809
$(u, v, \|u - v\|, u * v)$	0.727/0.814

Найкращі результати отримані для об’єднання $(u, v, |u - v|, u * v)$, яке ми і використовували для подальших експериментів. В таблиці 6 наведено результати тренування логістичної регресії для усіх моделей та типів агрегування.

Таблиця 6 - Результати (точність/F1) логістичної регресії

	MEAN	MAX	CLS
BERT	0.712/0.811	0.709/0.802	0.727/0.814
RoBERTa	0.782/0.839	0.762/0.829	0.809/0.860
ALBERT	0.770/0.837	0.751/0.823	0.769/0.843

Порівнюючи результати, отримані для триплетних мереж та логістичної регресії, очевидним є те, що використання міри подібності на основі нейронної мережі демонструє набагато кращі результати, особливо при використанні моделей RoBERTa та ALBERT. Також варто зауважити, що отримані векторні подання кожного речення, в результаті тренування логістичної регресії, теж можна ефективно порівнювати за допомогою косинуса подібності.

Висновки щодо типу агрегування

Загалом, аналізуючи отримані результати, приходимо до висновку, що тип агрегування MAX демонструє найгірші результати. Можна припустити, що функція максимуму є досить агресивною і відкидає важливі узагальнені властивості. MEAN та CLS демонструють досить схожі результати, які

мотивують використовувати один з цих типів агрегування для векторного подання речення. Слідуючи оригінальній статті BERT та беручи до уваги наведені вище результати, в наступних експериментах використовувався тип агрегування CLS.

Тонке налаштовування моделей

Як уже було сказано в попередніх розділах, BERT-подібні моделі мають певні попередньо натреновані ваги. Для кожної окремої задачі модель необхідно дотренувати, або іншим словами – тонко налаштувати (fine-tune). Це стосується і задачі визначення перефразувань на основі корпусу MRPC, коли два речення об'єднуються в єдину вхідну послідовність. Таке тонке налаштовування дозволяє отримати дуже хороші результати, але, як було проаналізовано в розділі II, даний підхід не може бути застосований на практиці через значні обчислювальні витрати. Тим не менш, в таблиці 7 наведемо результати власного тонкого налаштування та результати, наведені авторами в оригінальних статтях. Варто звернути увагу, що наші результати були отримані для base версій усіх моделей. Результати RoBERTa зі статті наведені для версії large, а для ALBERT – для версії xxlarge. Обидві ці версії мають значно більшу кількість параметрів, ніж версія base.

Таблиця 7 - Результати (точність/F1) тонкого налаштування

	Отримано	Стаття
BERT	0.864/0.898	-/0.889
RoBERTa	0.885/0.915	-/0.909 (large)
ALBERT	0.857/0.894	-/0.909 (xxlarge)

Комбінований підхід

Усі розглянуті моделі надають певний початковий набір ваг, які є відправною точкою для подальшого тренування. Незважаючи на відсутність можливості застосування підходу визначення семантичної подібності, який пропонується в оригінальному методі тонкого налаштування, тобто в об'єднанні двох речень в одну послідовність, яка подається на вхід мовній моделі, ми

можемо використати ваги, отримані в процесі такого налаштування, в якості стартової точки для розглянутого вище метричного навчання та логістичної регресії. Назвемо такий двокроковий метод тренування комбінованим підходом.

В таблиці 8 наведено результати навчання мовних моделей, які спочатку були тонко налаштовані (таблиця 7), з використанням розглянутих стратегій тренування та типу агрегування CLS, де Baseline - стартовий результат моделей з косинусом подібності, Siam – стратегія навчання сіамських мереж, Triplet – стратегія навчання триплетних мереж, LR – логістична регресія.

Таблиця 8 - Результати (точність/F1) на основі комбінованого підходу

	Baseline	Siam	Triplet	LR
BERT	0.708/0.790	0.751/0.828	0.743/0.825	0.796/0.851
RoBERTa	0.729/0.807	0.721/0.811	0.725/0.817	0.798/0.855
ALBERT	0.692/0.799	0.730/0.822	0.724/0.819	0.779/0.843

Аналізуючи результати, можна впевнено сказати, що попереднє тонке налаштування моделі сприяє отриманню кращих результатів при використанні подальших стратегій тренування. Таким чином, запропонований авторами оригінальної BERT підхід тонкого налаштування для задач, які оперують парама речень, може бути корисним у моделюванні семантики речення, тобто у побудові векторного простору речень, в якому семантично схожі речення, які подані точками цього простору, знаходяться близько в термінах певної міри подібності.

Загалом, отримані результати тренувань вказують на те, що звичайне використання виходів розглянутих мовних моделей для отримання векторного подання речення є малоефективним. Стратегія метричного навчання та логістичної регресії здатні допомогти отримати більш дискримінативні подання речень для точнішого визначення семантичної подібності.

ВИСНОВКИ

В даній магістерській роботі було запропоновано та досліджено ефективні підходи до аналізу семантичної подібності речень з використанням методів штучного інтелекту.

Найперше, було проведено аналіз сучасних глибоких мовних моделей. Базовими в даній роботі були глибокі нейронні мережі на основі Transformer, а саме BERT, RoBERTa та ALBERT. Механізм уваги, який використовується в даних моделях є потужним інструментом для моделювання та машинного розуміння природної мови. Контекстуальні подання слів, які ми отримуємо в результаті обчислень моделі над послідовністю вхідних даних, дозволяють будувати векторні подання речень для їх подальшого аналізу.

Розглянуті глибокі мовні моделі здатні отримати приховані ознаки слів та визначити їхні зв’язки у реченні, а також зв’язки між реченнями. Проте для моделювання семантики речення, тобто для побудови векторного простору, де семантично подібні речення, подані точками цього простору, знаходяться близько один від одного, нам потрібна відповідна інтерпретація ознак та зв’язків, які нам надаються мовною моделлю. Для цього були досліджені та застосовані стратегії тренування, які дозволили отримати більш дискримінативні ознаки. Метричне навчання на основі сіамських та триплетних мереж з косинусною мірою подібності дозволило покращити початкові результати розглянутих мовних моделей. Логістична регресія, як міра порівняння, показала, що такий підхід є дуже перспективним. Запропонований комбінований підхід з використанням оригінального тонкого налаштування BERT-подібних моделей продемонстрував значні покращення для попередньо розглянутих стратегій тренування. Загалом отримані результати свідчать про ефективність запропонованих та досліджених підходів.

Для проведення експериментів та навчання глибоких нейронних мереж було розроблено програмну систему, яка дозволяє будувати, тренувати та оцінювати різноманітні варіації архітектур моделей з різними цільовими функціями, параметрами, оптимізаторами тощо.

Отримані результати можна впровадити, зокрема, в системах виявлення плагіату або для пошуку схожих питань у базах Frequently Asked Questions (FAQ).

Проведені експерименти продемонстрували ефективність розглянутих підходів та доцільність подальших досліджень в галузі аналізу семантичної подібності речень. Перш за все, варто звернути увагу на інші сучасні глибокі мовні моделі, які кожного року демонструють все кращі результати у багатьох задачах в галузі обробки природної мови. Також доцільно провести експерименти з використанням більших текстових корпусів, які містять десятки тисяч різноманітних пар речень. Крім того, потрібно дослідити вплив різних тренувальних конфігурацій на остаточну точність моделі. Загалом галузь штучного інтелекту розвивається стрімкими темпами та демонструє неймовірні результати у багатьох сферах людського життя, тому дуже важливою є постійна адаптація нових методів для покращення уже готових та винайдення нових рішень для розв’язання багатьох прикладних задач, у тому числі, і аналізу семантичної подібності речень.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Sennrich, R., Haddow, B., & Birch, A. (2015). Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*.
2. Schuster, M., & Nakajima, K. (2012, March). Japanese and korean voice search. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 5149-5152). IEEE.
3. Kudo, T. (2018). Subword regularization: Improving neural network translation models with multiple subword candidates. *arXiv preprint arXiv:1804.10959*.
4. Kudo, T., & Richardson, J. (2018). Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*.
5. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *arXiv preprint arXiv:1706.03762*.
6. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).
7. Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.
8. Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., ... & Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.
9. Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., & Soricut, R. (2019). Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.

10. Conneau, A., & Kiela, D. (2018). Senteval: An evaluation toolkit for universal sentence representations. *arXiv preprint arXiv:1803.05449*.
11. https://gombru.github.io/2019/04/03/ranking_loss/
12. https://omoindrot.github.io/triplet-loss
13. Reimers, N., & Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.
14. Amaral, A. (2013). Paraphrase Identification and Applications in Finding Answers in FAQ Databases.
15. Kong, L., Han, Z., Han, Y., & Qi, H. (2020). A Deep Paraphrase Identification Model Interacting Semantics with Syntax. *Complexity, 2020*.
16. https://en.wikipedia.org/wiki/F-score
17. https://huggingface.co/