МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РЕАЛІЗАЦІЯ ОБЧИСЛЕННЯ СХОЖОСТІ НАЗВ КОМПАНІЙ»**

Виконав: студент 4 курсу, групи 6.1220
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)
освітньої програми Комп'ютерні науки
(назва освітньої програми)
В.О. Копилов
(ініціали та прізвище)
Керівник доцент кафедри комп'ютерних наук, к.т.н. Добровольський Г. А.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент професор кафедри програмної інженерії, к.ф.-м.н., доцент, Кудін О.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти бакалавр

Спеціальність 122 Комп'ютерні науки
(шифр і назва)

Освітня програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук,
д.т.н., професор

_____ Шило Г.М.
(підпис)

" 25 " грудня 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Копилову Вадимові Олександровичу
(прізвище, ім'я та по-батькові)

1. Тема роботи Реалізація обчислення схожості назв компаній

керівник роботи Добровольський Геннадій Анатолійович, к.т.н.
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 21 » грудня 2023 року № 2180-с

2. Строк подання студентом роботи 05.06.2024

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Експлуатація та тестування моделі.

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

6. Консультанти розділів роботи

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

7. Дата видачі завдання 25.12.2023

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	24.01.2024	
2.	Збір вихідних даних.	25.02.2024	
3.	Обробка методичних та теоретичних	21.03.2024	
	джерел.		
4.	Розробка першого та другого розділу.	28.04.2024	
5.	Розробка третього розділу.	23.05.2024	
6.	Оформлення та нормоконтроль	05.06.2024	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	22.06.2024	

Студент _____ В.О. Копилов
(підпис) (ініціали та прізвище)

Керівник роботи _____ Г.А. Добровольський
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г. Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Реалізація обчислення схожості назв компаній»: 57 с., 30 рис., 15 джерел, 1 додатків**.**

АНАЛІЗ ДАНИХ, КОСИНУСНА СХОЖІСТЬ, МАШИННЕ НАВЧАННЯ, ОБРОБКА ПРИРОДНОЇ МОВИ, СХОЖІСТЬ ТЕКСТІВ, BERT.

Об’єкт дослідження – задача порівняння схожості назв компаній за допомогою методів обробки природної мови.

Мета роботи: визначення ефективності застосування моделі BERT для обчислення схожості між правильними та зіпсованими назвами компаній з використанням косинусної схожості.

Метод дослідження – аналітичний.

Щоб досягти поставленої мети, був розроблений метод порівняння назв компаній за схожістю, що базується на використанні моделі BERT для отримання векторних представлень тексту та обчислення косинусної схожості між ними. Було визначено підхід для генерації зіпсованих назв компаній шляхом додавання загальних слів до правильних назв.

Метод перевірено на реальних даних з різними форматами назв компаній. Отримані результати свідчать про високу ефективність використання моделі BERT для обчислення схожості між назвами компаній. Застосування цього методу дозволяє точно визначати схожість навіть у випадках, коли назви компаній містять загальні слова або незначні помилки.

Створений підхід до обчислення схожості між назвами компаній може бути використаний для автоматизації процесів обробки текстової інформації, покращення якості пошукових систем та систем перевірки достовірності інформації. Застосування цього підходу є особливо актуальним у сферах обробки природної мови, де важливо враховувати нюанси та варіації тексту для точного розпізнавання та порівняння інформації.

SUMMARY

Bachelor's Qualifying Theses «Implementation of calculating the similarity of company names»: 57 pages, 30 figures, 15 references, 1 supplements.

DATA ANALYSIS, COSINE SIMILARITY, MACHINE LEARNING, NATURAL LANGUAGE PROCESSING, TEXT SIMILARITY, BERT.

Object of the study – the task of comparing the similarity of company names using natural language processing methods.

Aim of the study: to determine the effectiveness of using the BERT model to calculate the similarity between correct and spoofed company names using cosine similarity.

Method of research – analytical.

To achieve this goal, a method for comparing company names by similarity was developed, based on the use of the BERT model to obtain vector representations of text and calculate the cosine similarity between them. An approach was defined to generate corrupted company names by adding common words to the correct names.

The method is tested on real data with different company name formats. The results obtained show the high efficiency of using the BERT model to calculate the similarity between company names. The application of this method allows to accurately determine the similarity even in cases where company names contain common words or minor errors.

The developed approach to calculating the similarity between company names can be used to automate text information processing, improve the quality of search engines and information verification systems. This approach is especially relevant in natural language processing, where it is important to take into account the nuances and variations of text for accurate recognition and comparison of information. paper

ЗМІСТ

ВСТУП

В епоху великих даних та автоматизованої обробки тексту, порівняння словосполучень та власних назв є однією з важливих задач, що стоїть перед сучасними інформаційними системами. Порівняння таких текстових елементів знаходить застосування в різноманітних сферах, від пошукових систем до аналізу конкурентного середовища. Особливо складними для порівняння є назви компаній, адже вони можуть містити як унікальні слова, так і загальні компоненти, які утруднюють точне визначення схожості.

Назви компаній можуть включати як специфічні терміни, що мають велике значення для ідентифікації, так і загальні слова, які можуть бути спільними для багатьох назв. Наприклад, назви "Tech Innovations" та "Innovative Technologies" містять як важливі ключові слова, так і загальні терміни. Традиційні методи порівняння текстів не завжди здатні коректно врахувати цю різницю, що призводить до неточних результатів. Таким чином, виникає потреба в розробці більш ефективних підходів для порівняння таких текстових одиниць.

Метою роботи є розробка методу для обчислення схожості назв компаній на основі моделі BERT. Це сучасна технологія в області обробки природної мови, забезпечує контекстуальне розуміння тексту, що дозволяє більш точно визначати семантичну схожість між словосполученнями. Використання BERT у цій задачі дозволить враховувати не лише поверхневу лексичну схожість, а й глибинні семантичні зв’язки між словами, що сприятиме підвищенню точності порівняння назв компаній.

Робота включає аналіз існуючих методів обчислення схожості назв, опис архітектури BERT, та розробку спеціальних алгоритмів і підходів для обробки назв компаній. Результати матимуть практичне застосування у сфері бізнес-аналітики, маркетингу та інформаційних технологій, забезпечуючи точніше та ефективніше порівняння назв компаній.

1 МЕТОД ПОРІВНЯННЯ СЛОВОСПОЛУЧЕНЬ

Метод порівняння словосполучень в обробці природної мови використовується для аналізу та зіставлення фраз або речень з метою виявлення їхньої схожості або різниці. Цей процес включає в себе наступні етапи:

Обробка тексту. Перед порівнянням необхідно провести попередню обробку тексту, таку як токенізація (розділення тексту на окремі слова або токени), видалення стоп-слів (слів, які не несуть значущої інформації, наприклад, "і", "у", "на", "за" тощо), стемінг або лематизація (приведення слів до їхньої базової форми).

Представлення тексту. Слова або фрази потрібно перетворити у векторний формат, щоб комп’ютер міг їх аналізувати. Це може виконуватися за допомогою методів векторизації, таких як Bag of Words (Мішок слів), TF-IDF (частота та обернена частота документів) або векторизація на основі векторних представлень слів, таких як Word2Vec, GloVe тощо.

Визначення схожості. Після того, як тексти були представлені у векторному форматі, можна використовувати різні метрики схожості, такі як косинусна схожість, щоб порівняти їхні вектори. Чим більше значення метрики схожості, тим більш схожі тексти вважаються.

Використання моделей глибокого навчання. Останнім часом методи глибокого навчання, такі як BERT або GPT, стали популярними для порівняння текстів у NLP завдяки їхній здатності до вивчення складних залежностей між словами та фразами.

Врахування контексту. В деяких випадках важливо враховувати контекст при порівнянні текстів. Наприклад, моделі контекстуальних векторних представлень, такі як ELMo або GPT, можуть допомогти у врахуванні значень слів в залежності від їхнього контексту.

Машинне навчання. Застосування методів машинного навчання, таких як класифікаційні або регресійні моделі, для визначення схожості між

словосполученнями. Моделі можуть навчатися на парах текстів, що є схожими або несхожими, та використовувати ці знання для прогнозування схожості нових текстів.

Векторне представлення слів. Використання векторних представлень слів для знаходження семантичної схожості між словами або фразами. Моделі, такі як Word2Vec, GloVe, або FastText, створюють вектори для слів таким чином, що схожі за значенням слова мають близькі вектори.

Механізми уваги. Використання механізмів уваги у моделях глибокого навчання, які дозволяють враховувати важливість різних частин вхідного тексту при порівнянні. Це дозволяє моделям зосередитися на ключових словах або фразах під час порівняння текстів.

Метрики відстані. Використання різних метрик відстані, таких як відстань Левенштейна, що вимірює "відстань" між двома текстовими рядками шляхом підрахунку мінімальної кількості редагувань (вставок, видалень, замін), необхідних для перетворення одного рядка в інший.

Метод порівняння словосполучень використовується для багатьох завдань, таких як виявлення плагіату, класифікація текстів, підсумовування, машинний переклад тощо. Це допомагає виявити схожість або різницю між текстовими даними, що є важливим для різних аспектів аналізу тексту. Зокрема, використання моделей глибокого навчання, таких як BERT або GPT, дозволяє отримати більш точні результати за рахунок їхньої здатності до усвідомлення контексту та складних залежностей між словами і фразами. Такі методи відкривають нові можливості для розвитку різних застосувань у сфері обробки природної мови.

1.1 Компоненти методу

Обробка природної мови складається з двох основних компонентів. Це розуміння природної мови (NLU) і генерація природної мови (NLG).

Розуміння природної мови (Natural Language Understanding) – це процес, коли комп’ютерна система аналізує та розуміє людську мову, розпізнаючи інтонацію, смислові зв’язки, синоніми та інші мовні аспекти. NLU використовується в різних сферах, включаючи відповіді голосом, автоматизовані чат-боти, аналіз текстів тощо.

Генерація природної мови (Natural Language Generation) – це процес, за допомогою якого комп’ютерна система створює текст або мовлення, яке звучить природно для людини. NLG застосовується в системах автоматичного написання тексту, генерації звітів, створення діалогів для чат-ботів та інших задач, де потрібно створювати мовленнєвий вихід на основі даних чи інструкцій.

Розуміння та генерація природної мови спільно створюють основу для багатьох застосувань штучного інтелекту, які взаємодіють з людьми через мову. Вони допомагають покращити ефективність та доступність багатьох сервісів та продуктів, роблячи їх більш природними та зручними у використанні.

NLU базується на алгоритмах машинного навчання, які аналізують великі обсяги текстової інформації для вивчення мовних закономірностей і контексту. Такі системи використовують методи обробки природної мови, такі як синтаксичний аналіз, семантичне моделювання, векторне представлення слів і тематичне моделювання, для досягнення більш високого рівня розуміння тексту

З іншого боку, NLG використовується для створення тексту зрозумілого для людини на основі внутрішніх моделей та шаблонів. Системи NLG можуть використовувати правила, статистичні моделі або глибокі нейронні мережі для генерації тексту з урахуванням контексту, мовного стилю та інших факторів.

Обидва компоненти, NLU і NLG, є ключовими для розвитку розумних систем, які можуть ефективно взаємодіяти з людьми через природну мову. Вони застосовуються в різних галузях, від веб-пошуку та особистих асистентів до медичних систем і робототехніки, сприяючи поліпшенню якості обслуговування та зручності користування технологіями.

Наприклад, у сфері веб-пошуку системи з розумінням природної мови (NLU) допомагають інтерпретувати запити користувачів і забезпечують більш

точні та релевантні результати пошуку. Вони можуть розрізняти синоніми, розуміти контекст і навіть враховувати мовні відмінності для користувачів різних країн або регіонів.

У робототехніці системи з NLG використовуються для створення голосових інструкцій для роботів, які працюють у виробничих середовищах. Вони можуть надавати звукові команди та пояснення для роботів, що допомагає підвищити ефективність та безпеку виробничих процесів.

Загалом, обробка природної мови відіграє ключову роль у багатьох аспектах сучасного життя, полегшуючи взаємодію з технологіями та забезпечуючи нові можливості для автоматизації інформаційних процесів в різних галузях.

1.1.1 Попередня обробка тексту

Попередня обробка тексту – це важливий крок в обробці природної мови та машинному навчанні, коли необроблені текстові дані перетворюються у формат, який може бути легко зрозумілим і проаналізованим машинами. Вона включає в себе очищення, перетворення та збагачення вихідних текстових даних для підвищення точності та ефективності алгоритмів машинного навчання.

Текстові дані часто містять шум у вигляді небажаних символів, спеціальних символів, HTML-тегів і неузгоджених форматів. Попередня обробка текстових даних допомагає видалити ці шумові елементи, стандартизувати текстовий формат і виокремити значущі ознаки, які можна використовувати для побудови моделей машинного навчання.

Обробка тексту включає кілька етапів, таких як переведення в нижній регістр, токенізація, видалення стоп-слів, розбиття на частини, лематизація, перевірка та виправлення орфографії, позначення частин мови, розпізнавання іменованих об’єктів, векторизація та вилучення ознак.

Належна попередня обробка текстових даних може призвести до створення більш точних і ефективних моделей машинного навчання, особливо в таких

завданнях, як аналіз настроїв, класифікація текстів, узагальнення текстів і мовний переклад.

Нижче наведені загальні кроки, пов’язані з попередньою обробкою тексту:

Токенізація. Процес перетворення вихідного тексту на послідовність токенів (слів, фраз, символів тощо) називається токенізацією.

Токенізація необхідна при обробці тексту, щоб розбити великий текст на менші частини, які називаються токенами, і які легше піддаються аналізу. Токенізація важлива, оскільки більшість алгоритмів НЛП вимагають, щоб вхідні дані були у вигляді токенів, а не повного блоку тексту.

Наприклад, розглянемо речення: "The quick brown fox jumped over the lazy dog." Токенізація цього речення розбиває його на окремі токени, такі як "The", "quick", "brown", "fox", "jumped", "over", "the", "lazy", і "dog".

Після того, як текст токенізовано, над цими токенами можна виконувати різні завдання НЛП, такі як маркування частин мови, розпізнавання іменованих сутностей, аналіз настрою тощо.

Окрім того, що токенізація є необхідним кроком для більшості алгоритмів НЛП, вона також може підвищити точність аналізу, зменшуючи неоднозначність тексту. Наприклад, розглянемо слово "бігти". Залежно від контексту, його можна інтерпретувати як дієслово або іменник. Розбиваючи текст на токени, алгоритм може легше визначити передбачуване значення кожного слова на основі навколишнього контексту.

Видалення стоп-слів. Стоп-слова – це загальновживані слова в мові, такі як "the," "and," "a," тощо, які не додають особливого сенсу тексту. Видалення цих слів допомагає зменшити шум у текстових даних.

Видалення стоп-слів необхідне для попередньої обробки тексту, оскільки стоп-слова – це слова, які дуже часто зустрічаються в мові, але не передають жодного конкретного значення в контексті речення. Ці слова займають цінне місце в пам’яті і можуть сповільнити аналіз. Тому їх видалення з тексту може підвищити ефективність і точність аналізу.

Наприклад, у реченні "The cat sat on the mat", стоп-словами є "the" і "on". Видалення цих стоп-слів призведе до того, що слова, які залишилися, матимуть більше значення в аналізі: "ca", "sat", і "mat".

Стеммінг. Стеммінг – це процес скорочення слова до його основи або кореневої форми. Наприклад, слова "jumping", "jumps" і "jumped" за допомогою алгоритму стеммінгу будуть скорочені до "jump". Основна мета стеммінгу – звести різні форми слова до спільної базової форми, що може допомогти в таких завданнях, як класифікація тексту, аналіз настроїв та пошук інформації.

Стеммінг важливий для обробки текстів, оскільки він допомагає зменшити кількість унікальних слів у текстовому корпусі, що полегшує обробку та аналіз даних. Приводячи слова до їхньої базової форми, ми можемо групувати різні варіації одного і того ж слова і розглядати їх як єдиний термін. Це може підвищити точність та ефективність багатьох завдань з обробки природної мови.

Існує кілька популярних алгоритмів стемінгу, які використовуються в обробці текстів, зокрема алгоритм Портера та алгоритм "снігової кулі". Ці алгоритми працюють, застосовуючи набір правил для видалення афіксів зі слів і приведення їх до базової форми.

Лемматизація. Лематизація – це процес приведення слів до їхньої базової або словникової форми (відомої як лема), щоб їх можна було аналізувати як єдине ціле, а не як кілька різних форм. Наприклад, слово "running" можна звести до його базової форми "run" за допомогою лематизації.

Лематизація необхідна для попередньої обробки тексту, щоб зменшити варіації слів і згрупувати схожі слова разом, що може допомогти в аналізі та розумінні тексту. Приведення слів до їхньої базової форми полегшує підрахунок і аналіз входжень слів, а також виявлення зв’язків між словами в тексті. Це особливо корисно в задачах обробки природної мови (NLP), таких як аналіз настроїв, моделювання тем і класифікація текстів.

Стеммінг і лематизація – це два поширені методи, які використовуються в обробці природної мови для скорочення слів до їхніх базових або кореневих форм. Основна відмінність між стеммінгом і лематизацією полягає в тому, що

стеммінг – це грубий процес видалення суфіксів зі слів для отримання їх кореневих форм, тоді як лематизація – це більш складний процес зіставлення слів з їх базовими формами з використанням словника і морфологічного аналізу слів.

Стеммінг відсікає кінець слова за допомогою простих правил та евристик, не беручи до уваги контекст або значення слова. Наприклад, слово "running" буде перетворено на "run", "runningly" – на "running", і так далі. Стеммінг швидший і менш ресурсомісткий, ніж лематизація, але він може спричинити неточності через надмірне або недостатнє розчленування слів.

З іншого боку, лематизація використовує словниковий і морфологічний аналіз слів, щоб зіставити слова з їхніми базовими формами, або лемами. Цей процес враховує контекст і значення слова і може дати точніші результати, ніж словотвір. Наприклад, слово "running" буде лематизовано на "run", а "ran" також буде лематизовано на "run". Однак лематизація є більш трудомістким процесом, ніж стеммінг, і вимагає більше ресурсів, таких як словник або тезаурус.

Позначення частин мови (POS). Позначення частин мови – це процес ідентифікації та позначення частини мови кожного слова в реченні, наприклад, іменника, дієслова, прикметника, прислівника тощо. POS-тегування корисне в різних завданнях обробки природних мов, таких як аналіз настрою, класифікація текстів, вилучення інформації та машинний переклад.

Розпізнавання іменованих об’єктів (NER). Розпізнавання іменованих об’єктів (NER) – це метод обробки природної мови, який використовується для ідентифікації та вилучення іменованих об’єктів із заданого тексту. Іменовані сутності можуть бути будь-якими, наприклад, особами, організаціями, місцезнаходженням, продуктами тощо.

NER є важливим кроком у попередній обробці тексту, оскільки допомагає витягти важливу інформацію з тексту, яка може бути використана для різних завдань, таких як аналіз настроїв, класифікація текстів, пошук інформації тощо.

Перевірка та виправлення орфографії. Перевірка та виправлення орфографії – це процес виявлення та виправлення орфографічних помилок у тексті. Це важливий етап попередньої обробки тексту, оскільки він може

підвищити точність алгоритмів обробки природної мови, які застосовуються до текстових даних.

Видалення HTML-тегів, розділових знаків і спеціальних символів. Видалення HTML-тегів, розділових знаків і спеціальних символів необхідне для попередньої обробки тексту, щоб очистити текстові дані і підготувати їх до подальшої обробки. HTML-теги, розділові знаки та спеціальні символи не впливають на зміст тексту і можуть спричинити проблеми під час аналізу тексту.

Перетворення в нижній регістр. Перетворення тексту в нижній регістр є поширеним кроком попередньої обробки в обробці природної мови (NLP), щоб зробити текстові дані узгодженими і легшими для аналізу. Цей крок передбачає перетворення всіх літер у тексті на малі, щоб слова, які відрізняються лише регістром, сприймалися як одне й те саме слово.

Перетворення тексту на малі літери допомагає зменшити кількість унікальних слів у тексті, що, в свою чергу, допомагає зменшити розрідженість даних. Це важливо, оскільки більшість алгоритмів НЛП працюють краще, коли дані менш розріджені. Крім того, це може допомогти в таких завданнях, як підбір слів і пошук, оскільки усуває чутливість тексту до регістру.

Векторизація тексту. Векторизація тексту – це процес перетворення необробленого тексту в числове представлення, яке може бути використане алгоритмами машинного навчання. Це важливий крок у попередній обробці тексту, оскільки більшість алгоритмів машинного навчання працюють з числовими даними. Існує кілька способів векторизації тексту, зокрема мішок слів (Bag of Words, BoW), частота терміна – зворотна частота документа (TF-IDF) і вставки слів (Word Embeddings).

BoW представляє текст як набір унікальних слів, ігноруючи їхній порядок і контекст. Воно створює словник унікальних слів, присутніх у текстовому корпусі, а потім генерує матрицю, де кожен рядок представляє документ, а кожен стовпець – слово зі словника. Значення в кожній комірці матриці представляє частоту слова у відповідному документі.

Представлення TF-IDF схоже на BoW, але воно враховує важливість слова в документі та в усьому корпусі. Кожному слову присвоюється вага на основі його частоти в документі та зворотної частоти в усьому корпусі.

Вставлені слова представляють слова у вигляді щільних векторів у високорозмірному просторі, де відстань між векторами відображає семантичну схожість між словами. Вкладення слів створюються шляхом навчання нейронної мережі на великому масиві тексту, а потім вилучення вивчених векторних представлень слів.

1.1.2 Представлення слів у RNN: Word2vec, FastText, GloVe

Представлення слів є критичним аспектом для розуміння та обробки текстових даних. Деякі з найпопулярніших підходів до векторного представлення слів включають Word2Vec, FastText та GloVe. Кожен з цих методів має свої особливості та переваги.

Вбудовування є важливим компонентом конвеєрів обробки природної мови. Вони відносяться до векторного представлення текстових даних. Ви можете думати про вбудовування як про перетворення тексту, що читається людиною, в числа або вектори, що читаються комп’ютером, як показано на (рис. 1). Ці вбудовування можна використовувати в будь-якій задачі машинного навчання, яка приймає текст як вхідні дані, наприклад, відповіді на запитання, класифікація, генерація тексту.

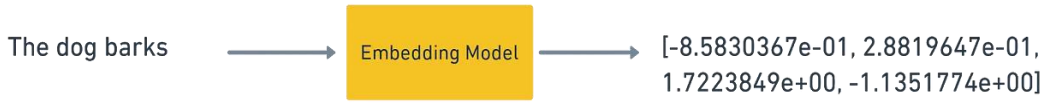


Рисунок 1 – Спрощена ілюстрація моделі вбудовування

Різні методи вбудовування відрізняються за своєю складністю та можливостями. Наприклад, найпростіша форма вбудовування слів може бути представлена за допомогою однохотових кодувань, коли кожному слову в

корпусі розміром V ставиться у відповідність унікальний індекс у векторі того ж розміру. Це дає нам вектор з усіх нулів, окрім одного елемента, який вказує на слово. Наприклад, припустимо, що словник складається лише з трьох слів: "the", "dog" і "barks" (рис. 2). Ми можемо закодувати кожне слово вектором довжиною 3, де для кожного слова окремий індекс має значення одиниці, а решта – нулі, як показано нижче.

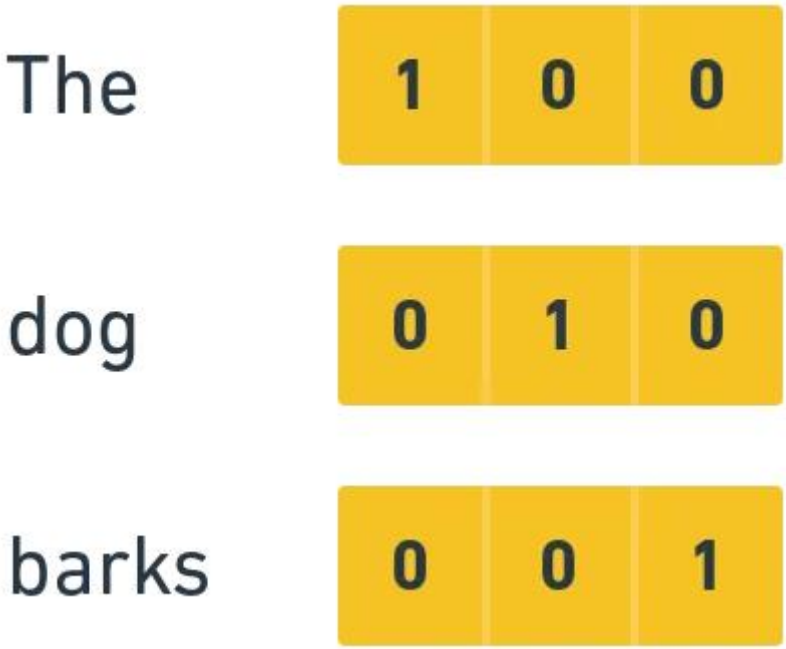


Рисунок 2 – Однозначні кодування всіх трьох слів у корпусі

Обмеження цього підходу досить очевидні. Наприклад, векторні представлення є дуже розрідженими і можуть бути великими для великого корпусу; крім того, зв’язок між словами не може бути врахований. Тому необхідна певна форма зменшення розмірності, яка б дозволила вловити латентну семантику тексту.

Здатність моделі вбудовування ефективно представляти текстові дані безпосередньо пов’язана із загальною продуктивністю завдання машинного навчання, будь то аналіз настроїв, автозаповнення додатків або виявлення побічних ефектів лікарських засобів.

Як і будь-яка інша модель машинного навчання, продуктивність моделей вбудовування сильно залежить від навчальних даних. Структура, стиль і лексика, що використовуються в текстових даних, можуть відрізнятися залежно від галузі. Наприклад, слова, що використовуються в науково-дослідній роботі, дуже відрізняються від лексики та правопису, що використовуються в твітах. Тому, залежно від застосування моделей вбудовування, потрібно вибирати відповідні дані.

Текстові дані можуть бути дуже зашумленими, тому перед навчанням моделі необхідна надійна процедура попередньої обробки. Як і на етапі збору даних, цей етап також дуже специфічний для конкретної предметної області. Наприклад, у клінічному тексті перетворення всіх символів на малі літери може спричинити неоднозначність деяких абревіатур, наприклад, ADD (attention deficit disorder) буде перетворено на дієслово add.

Після того, як дані зібрані та попередньо оброблені, на них накладається модель. Залежно від архітектури моделі, існує ряд гіперпараметрів, які потрібно налаштувати для оптимізації роботи моделі. Найпоширенішими гіперпараметрами для вбудовування моделей є розмір вікна, який визначає кількість сусідніх слів, мінімальну частоту слів для включення їх до словника, а також розмірність вихідних векторів.

Продуктивність моделі вбудовування можна оцінити двома методами: внутрішнім і зовнішнім.

Внутрішня оцінка фокусується на здатності моделі вивчати семантичні та синтаксичні зв’язки між словами. Іншими словами, на її розумінні мови предметної області або контексту.

Зовнішнє оцінювання – це міра продуктивності вбудованої моделі при виконанні наступних завдань, таких як розпізнавання іменованих об’єктів, класифікація текстів, відповіді на запитання тощо.

Word2Vec – одна з найперших моделей вбудовування слів, яка є досить простою і неглибокою (3 шари) нейронною мережею з двома режимами для навчання представлень слів з великих немаркованих даних. Ці два режими

навчання називаються Continuous Bag Of Words (CBOW) та Skip-gram. Як видно з (рис. 3), ці два методи по суті є протилежними один одному.

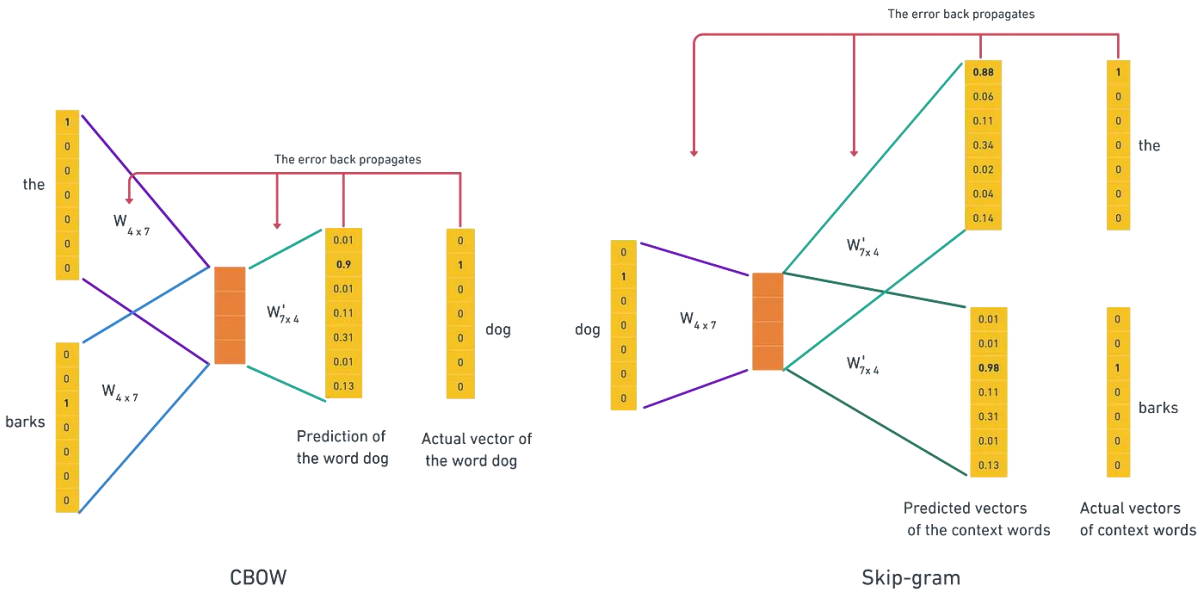


Рисунок 3 – Ілюстрація процесу навчання методів CBOW та Skip-gram

В обох методах всі слова в словнику кодуються одним символом, а розмір вікна визначається для розгляду фіксованої кількості слів; на рисунку нижче припустимо, що в словнику лише 7 слів, а розмір вікна дорівнює 3; тоді в CBOW мета полягає в тому, щоб передбачити центральне слово в кожному вікні на основі слів, що його оточують. Після того, як передбачення зроблено, різниця між передбаченим вектором і базовою істиною поширюється назад, щоб оновити дві вагові матриці. Ці ваги визначають вбудовування певного слова під час виведення.

З іншого боку, при використанні методу Skip-gram метою є передбачення контекстних слів за цільовим (центральним) словом, як показано на малюнку нижче.

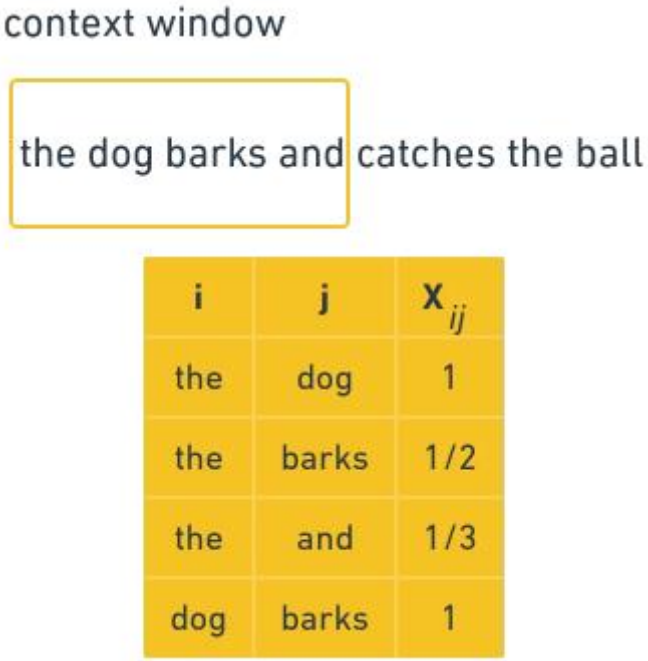
Будь-який із цих методів навчання можна використовувати для спеціальних даних. Як правило, Skip-gram краще підходить для більшого корпусу та повільніше, тоді як CBOW більше підходить, якщо корпус невеликий або швидкість є фактором у вашому навчальному конвеєрі.

Незважаючи на те, що Word2Vec чудово справляється з охопленням деяких тонкощів, він має свої обмеження; головним чином, він має єдине векторне представлення слова, незалежно від різних значень, які воно може мати в залежності від контексту. Ще одна проблема з цією моделлю полягає в тому, що вона не обробляє слова зі словникового запасу, оскільки вивчає лише представлення слів, які повторюються вище певного порогу в попередньо визначеному корпусі. Це може бути особливо проблематично для рідкісних слів або слів, які є специфічними для певної області, наприклад медицини.

GloVe (Global Vectors for Word Representation) – це метод, розроблений командою Стенфордського університету. Основна відмінність між GloVe та Word2Vec полягає в тому, що а) на відміну від Word2Vec, який є моделлю на основі прогнозів, Glove є методом на основі підрахунку та б) Word2Vec розглядає лише локальні властивості набору даних, тоді як GloVe розглядає глобальні властивості на додаток до місцеві.

GloVe створює зважену матрицю входжень на основі відстані, V × V, де V – розмір словника. З попередньо визначеним контекстним вікном (подібно до Word2Vec), кожен запис Xij у цій матриці відповідає слову i, що зустрічається на відстані від слова j. Якщо i і j зустрічаються разом і між ними немає інших слів, Xij дорівнює 1, якщо між і і j є одне слово, значення стає ½, якщо між ними є два слова, значення стає ⅓ і так далі, доки не буде досягнуто максимального контекстного вікна (рис. 4). Цей підрахунок на основі відстані дозволяє оцінювати слова, які зустрічаються послідовно, вище, ніж ті, що не зустрічаються.

Однак у великому корпусі очікується, що більшість клітинок у матриці входжень дорівнюватимуть 0, а ненульові клітинки матимуть великі значення, тобто розподіл матиме довгий хвіст. Для того, щоб пом’якшити цю проблему, до кожного запису додається 1, а потім обчислюється його логарифм.



i	j	X_{ij}
the	dog	1
the	barks	1/2
the	and	1/3
dog	barks	1

Рисунок 4 – Значення спільного розташування на основі відстані

Для подальшого кодування релевантності слів, що з'являються в одному контексті, кожен запис Xij має вагу на основі його значення, як показано на (рис. 5).

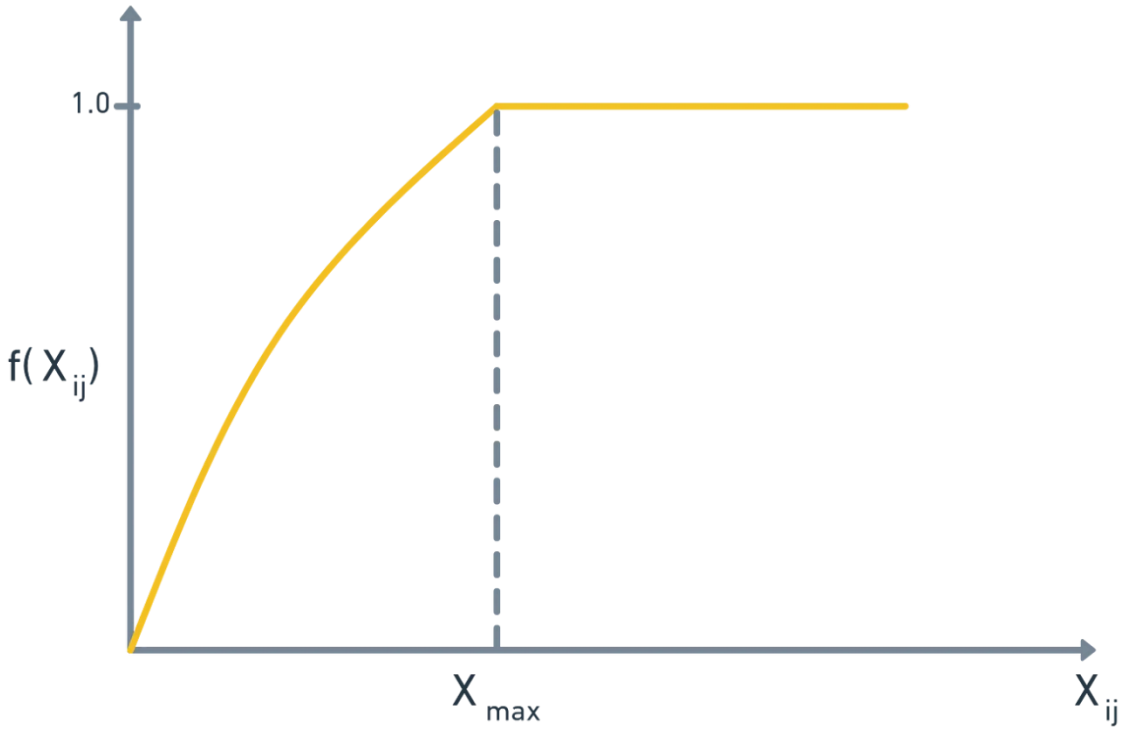


Рисунок 5 – Застосування вагової функції

Ця побудована матриця спільного проходження стає цільовою матрицею GloVe. Під час навчання факторизація матриці з градієнтним спуском використовується для прогнозування значень цієї матриці спільного входження. Як видно на (рис. 6), велика квадратна матриця спільного входження (R) у лівій частині рівняння може бути розбита на дві менші матриці, P і Q.

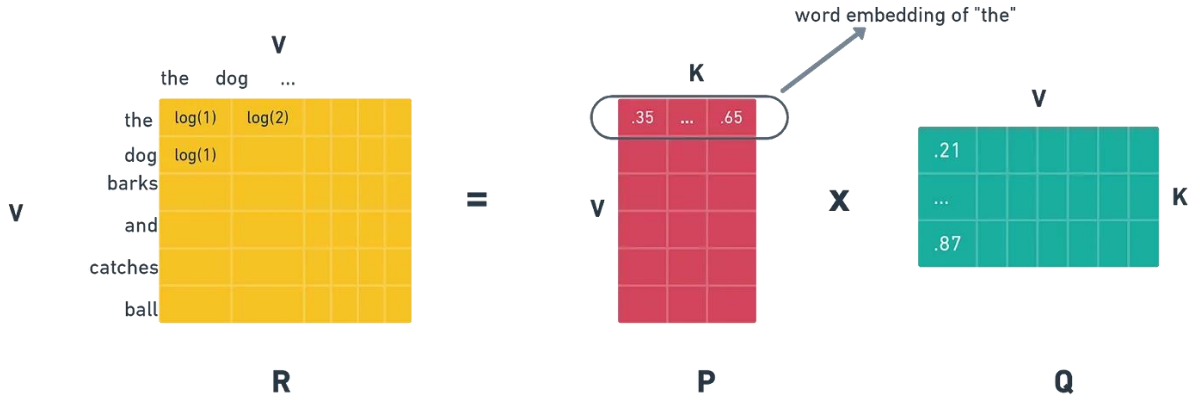


Рисунок 6 – Матриця спільного входження

Ці навчені матриці можна інтерпретувати як вкладення для кожного слова. Наприклад, можна взяти кожен рядок і матриці P або кожен стовпець і матриці Q як вкладення слова і . Крім того, середнє значення цих двох векторів можна використовувати як вкладення слова і. Причиною цього є те, що кожне співпоява Xij у R може бути апроксимовано рівнянням. 1, і після навчання кореляція між появою слова і з іншими словами проектується у кореляції векторів у і-му рядку та і-му стовпці матриць P та Q відповідно.

GloVe навчався на п’яти корпусах різного розміру: дампах Wikipedia 2010 і 2014 років, Gigaword 5, який містить 4,3 мільярда токенів, і веб-даних за допомогою Common Crawl, що складаються з 42 мільярдів токенів.

FastText – це покращення Word2Vec, розроблене Facebook AI Research (FAIR). На початку 2017 року Facebook AI Research опублікував статтю, яка представляє більш ефективний метод вбудовування FastText. Цей метод

побудовано на основі методу Skip-gram, але пом’якшує обмеження слів, які не містяться у словниковому запасі.

Щоб генерувати вбудовування слів за межами навченого словника, FastText розбиває слова на меншу послідовність символів, яка називається n-грамами. Наприклад, для n = 3, 3-грамами слова dog стають: "<do", "dog", "og>" і спеціальна послідовність "<dog>", що позначає все слово. Цей метод є ефективним, оскільки він вивчає представлення підслів, які є спільними для різних слів, і тому невидиме слово розбирається на його складові n-грами, які, швидше за все, були помічені під час навчання. Остаточне вкладення слова обчислюється як сума його складових вкладень n-грам.

Процес навчання FastText подібний до підходу Skip-gram у Word2Vec, за винятком того, що вхідне цільове слово розбивається на n-грами, і кожна n-грама представлена унікальним ідентифікатором. Для навчання використовувалися дампи Вікіпедії дев’ятьма різними мовами, арабською, чеською, німецькою, англійською, іспанською, французькою, італійською, румунською та російською.

За допомогою підходу FastText модель отримує більше інформації про морфологічні переходи слова, наприклад, як певний префікс може змінити значення будь-якого слова, яке йому передує. Він також більш стійкий до орфографічних помилок і може точніше представляти вкраплення рідкісних або невідомих слів. Незважаючи на те, що FastText має перевагу в тому, що він навчається багатьом мовам і пом’якшує проблему Word2Vec і GloVe; він має ту саму проблему генерування єдиного вбудовування для слова, незалежно від його контекстуального значення.

Всі ці методи представлення слів мають свої унікальні особливості та підходять для різних завдань. Word2Vec добре підходить для завдань, де важливий контекст слів. FastText є корисним у ситуаціях, коли потрібно враховувати морфологію та працювати з рідкісними словами. GloVe забезпечує високоякісні векторні представлення, ґрунтуючись на глобальній статистиці

тексту. Кожен з цих методів робить внесок у поліпшення обробки природної мови та розширює можливості рекурентних нейронних мереж.

1.1.3 BERT – загальний опис

BERT – це революційна модель обробки природної мови (NLP), розроблена компанією Google. Вона змінила ландшафт завдань розуміння мови, дозволивши машинам розуміти контекст і нюанси в мові.

У сфері обробки природної мови (NLP), що постійно розвивається, з’явилася революційна інновація під назвою BERT, яка змінила правила гри. BERT, що розшифровується як Bidirectional Encoder Representations from Transformers – це не просто ще одна абревіатура у величезному морі жаргону машинного навчання. Він відображає зміну в тому, як машини розуміють мову, дозволяючи їм розуміти складні нюанси і контекстуальні залежності, які роблять людське спілкування багатим і змістовним.

Уявіть собі речення: "Вона чудово грає на скрипці". Традиційні мовні моделі обробляють це речення зліва направо, упускаючи той важливий факт, що ідентичність інструменту ("скрипка") впливає на інтерпретацію всього речення. BERT, однак, розуміє, що контекстні зв’язки між словами відіграють вирішальну роль у виведенні значення. Він вловлює суть двонаправленості, дозволяючи розглядати повний контекст, що оточує кожне слово, революціонізуючи точність і глибину розуміння мови.

В основі BERT лежить потужна нейромережева архітектура, відома як Transformers. Ця архітектура включає механізм самоуваги, що дозволяє BERT зважувати значення кожного слова на основі його контексту, як попереднього, так і наступного. Це усвідомлення контексту надає BERT здатність генерувати контекстуалізовані вставки слів, які є представленнями слів з урахуванням їхніх значень у реченнях. Це схоже на те, як BERT читає і перечитує речення, щоб отримати глибоке розуміння ролі кожного слова.

Перш ніж BERT почне творити свою магію над текстом, його потрібно підготувати і структурувати так, щоб він був зрозумілим. У цій главі ми розглянемо найважливіші кроки попередньої обробки тексту для BERT, включаючи токенізацію, форматування вхідних даних і завдання Маскованої Мовної Моделі (MLM) (рис. 7).

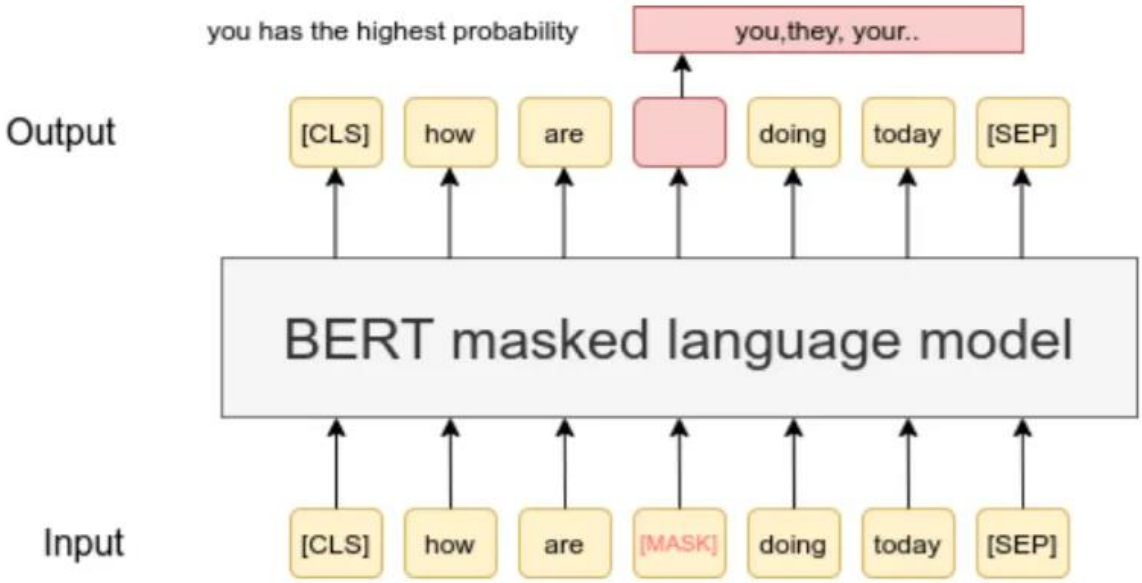


Рисунок 7 – Маскована мовна модель (MLM)

Токенізація: Розбиття тексту на значущі фрагменти. Уявіть, що ви вчите BERT читати книгу. Ви б не віддали всю книгу відразу; ви б розбили його на речення та абзаци. Подібним чином BERT потребує, щоб текст був розбитий на менші одиниці, які називаються токенами. Але ось нюанс: BERT використовує токенізацію WordPiece. Він розбиває слова на менші шматки, наприклад перетворюючи "running" на "run" та "ning". Це допомагає впоратися зі складними словами та гарантує, що BERT не загубиться в незнайомих словах.

Приклад: Оригінальний текст: "ChatGPT is fascinating. " Токени WordPiece: ["Chat", "##G", "##PT", "is", "fascinating", "."]

Форматування вхідних даних: Даємо BERT контекст. BERT любить контекст, і нам потрібно подати його йому на тарілочці. Для цього ми відформатуємо лексеми так, щоб BERT їх зрозумів. Ми додаємо спеціальні

лексеми, такі як [CLS] (позначає класифікацію) на початку та [SEP] (позначає розділення) між реченнями. Як показано на малюнку (модель машинної мови). Ми також призначаємо вставки сегментів, щоб сказати BERT, які лексеми належать до якого речення.

Приклад: Оригінальний текст: "ChatGPT is fascinating." Відформатовані токени: ["[CLS]", "Chat", "##G", "##PT", "is", "fascinating", ".", "[SEP]"]

Мета маскованої мовної моделі (MLM): Навчити BERT контексту. Секрет BERT полягає в його здатності розуміти двонаправлений контекст. Під час навчання деякі слова маскуються (замінюються на [MASK]) у реченнях, і BERT вчиться передбачати ці слова з контексту. Це допомагає BERT зрозуміти, як слова пов’язані одне з одним, як до, так і після. Як показано на малюнку (модель машинної мови)

Приклад: Оригінальне речення: "The cat is on the mat." Речення з маскою: "The [MASK] is on the mat."

Зрозумівши, як працює BERT, настав час застосувати його магію на практиці. У цій главі ми розглянемо, як налаштувати BERT для конкретних мовних завдань (рис. 8). Це передбачає адаптацію попередньо навченої моделі BERT для виконання таких завдань, як класифікація текстів.

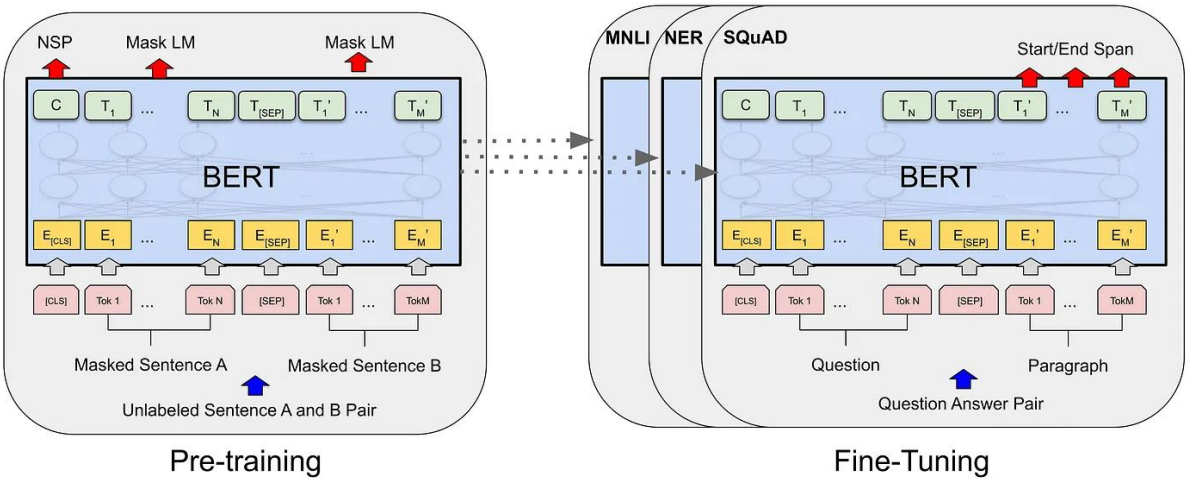


Рисунок 8 – Тонке налаштування BERT

BERT поставляється в різних варіантах, таких як BERT-base, BERT-large тощо. Варіації мають різні розміри та складність моделей. Вибір залежить від вимог вашої задачі та наявних ресурсів. Більші моделі можуть працювати краще, але вони також вимагають більшої обчислювальної потужності.

Уявіть собі BERT як мовного експерта, який вже прочитав тонну тексту. Замість того, щоб навчати його всьому з нуля, ми налаштовуємо його під конкретні завдання. У цьому і полягає магія трансферного навчання – використання вже наявних знань BERT та їх адаптація до конкретного завдання. Це як мати репетитора, який багато знає, але потребує лише деяких порад з конкретного предмету.

Завдання, для яких ми налаштовуємо BERT, називаються "подальшими завданнями". Приклади включають аналіз настроїв, розпізнавання іменованих об’єктів тощо. Точне налаштування передбачає оновлення ваг BERT з використанням даних для конкретного завдання. Це допомагає BERT спеціалізуватися на цих завданнях, не починаючи з нуля.

1.1.4 Механізм уваги

Механізм уваги є одним із останніх досягнень глибокого навчання, особливо для завдань обробки природної мови, таких як машинний переклад, підписи до зображень, створення діалогів тощо. Розроблений для підвищення продуктивності моделі RNN кодера декодера (seq2seq).

Увага пропонується як рішення обмеження моделі кодера-декодера, яка кодує вхідну послідовність в один вектор фіксованої довжини, з якого на кожному часовому кроці декодується вихідна послідовність. Вважається, що це питання є проблемою при декодуванні довгих послідовностей, оскільки нейронній мережі важко впоратися з довгими реченнями, особливо з тими, які довші за речення в навчальному корпусі.

Коли модель намагається передбачити наступне слово, вона шукає набір позицій у вихідному реченні, де сконцентрована найбільш релевантна

інформація. Потім модель прогнозує наступне слово на основі векторів контексту, пов’язаних з цими вихідними позиціями та всіма попередньо згенерованими цільовими словами.

Замість того, щоб кодувати вхідну послідовність в один фіксований вектор контексту, модель уваги створює вектор контексту, який фільтрується спеціально для кожного кроку виведення.

У багатьох завданнях, таких як машинний переклад або генерація діалогів, ми маємо послідовність слів на вході (наприклад, оригінальний текст англійською мовою) і хочемо згенерувати іншу послідовність слів на виході (наприклад, переклад на корейську мову). Нейронні мережі, особливо рекурентні (RNN), добре підходять для вирішення такої задачі.

Моделі нейронних мереж "від послідовності до послідовності" широко використовуються в NLP. Популярний тип цих моделей – "кодер-декодер" (рис. 9). У ній одна частина мережі – кодер – кодує вхідну послідовність у вектор контексту фіксованої довжини. Цей вектор є внутрішнім представленням тексту. Потім цей вектор контексту декодер декодує у вихідну послідовність.

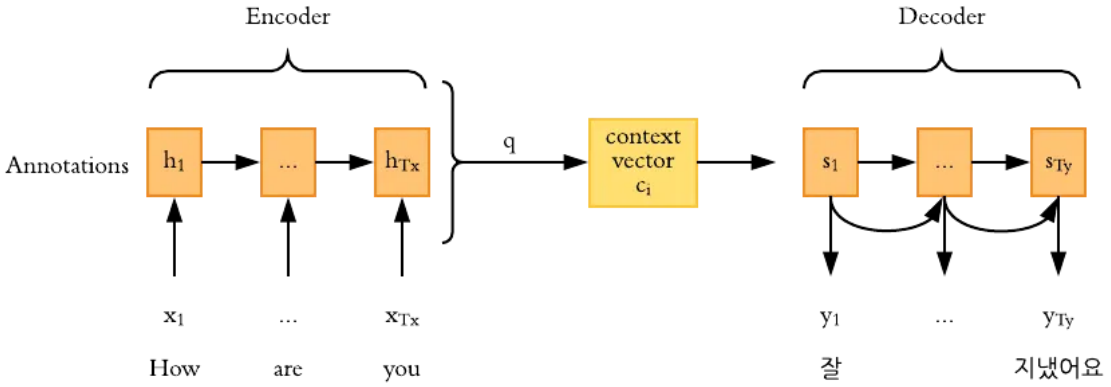


Рисунок 9 – Архітектура нейронної мережі кодер-декодер

Тут h позначає приховані стани кодера та s – декодера. Tx і Ty – довжини вхідної та вихідної послідовностей слів відповідно. q – функція, яка генерує вектор контексту з прихованих станів кодера. Це може бути, наприклад, просто

q({h_i}) = h_T. Отже, ми приймаємо останній прихований стан як внутрішнє представлення всього речення.

Однак, у загальному підході кодера-декодера є підступ: нейронна мережа стискає всю інформацію вхідного речення у вектор фіксованої довжини. Це призводить до зниження продуктивності при роботі з довгими реченнями.

Основна ідея: кожного разу, коли модель прогнозує вихідне слово, вона використовує лише ті частини вхідних даних, де сконцентрована найбільш релевантна інформація, а не ціле речення. Іншими словами, вона звертає увагу лише на деякі слова у вхідних даних.

Кодер працює так само, як і зазвичай, і різниця полягає лише у роботі декодера. Як ви можете побачити на малюнку, прихований стан декодера обчислюється за допомогою вектора контексту, попереднього виводу і попереднього прихованого стану (рис. 10). Але тепер ми використовуємо не один вектор контексту c, а окремий вектор контексту c_i для кожного цільового слова.

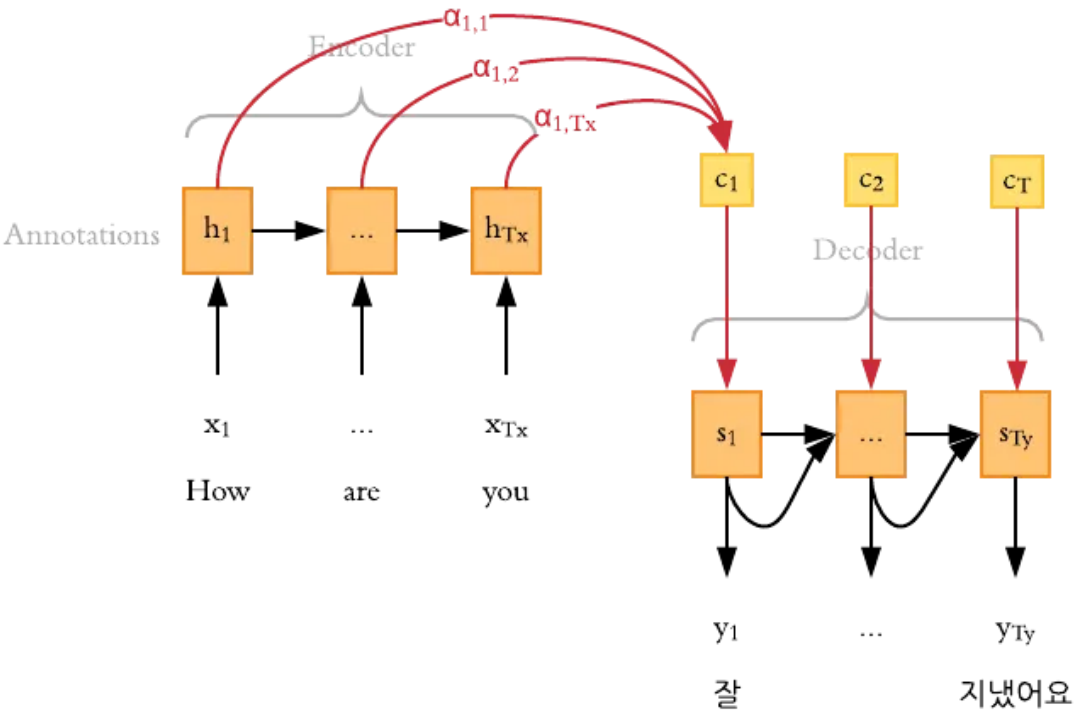


Рисунок 10 – Ілюстрація механізму уваги

Ці контекстні вектори обчислюються як зважена сума анотацій, згенерованих кодувальником. Вага кожної анотації обчислюється за допомогою моделі вирівнювання, яка оцінює, наскільки добре збігаються входи і виходи. Модель вирівнювання – це, наприклад, нейронна мережа прямого поширення. Загалом, це може бути будь-яка інша модель.

В результаті, альфа – ваги прихованих станів при обчисленні вектора контексту – показують, наскільки важлива дана анотація для прийняття рішення про наступний стан і генерації вихідного слова. Це і є показники уваги.

Механізм уваги можна розглядати як форму пам’яті. Вони поширюють цей механізм на налаштування кількох стрибків. Це означає, що мережа зчитує ту саму вхідну послідовність кілька разів, перш ніж створити вихід, і оновлює вміст пам’яті на кожному кроці. Інша модифікація полягає в тому, що модель працює з кількома вихідними реченнями замість одного.

Давайте подивимось на внутрішню роботу (рис. 11). Одношаровий випадок (a) реалізує одну операцію переходу по пам’яті. Весь вхідний набір речень перетворюється у вектори пам’яті m. Також вбудовано запит q для отримання внутрішнього стану u. Ми обчислюємо відповідність між u та кожною пам’яттю, беручи внутрішній добуток з наступним softmax. Таким чином ми отримуємо вектор ймовірності p над входами (це частина, що стосується уваги). Кожен вхід також має відповідний вихідний вектор. Ми використовуємо вагу p для зважування суми цих вихідних векторів. Ця сума є вектором відповіді o з пам’яті. Тепер у нас є вихідний вектор o і вхідний вкладиш u. Ми підсумовуємо їх, множимо на вагову матрицю W і застосовуємо softmax, щоб передбачити мітку.

Тепер ми можемо розширити модель для обробки K стрибкоподібних операцій (b). Шари пам’яті складені таким чином, що вхідні дані для шарів k + 1 є сумою вихідних даних і вхідних даних з шару k. Кожен шар має власні матриці вбудовування для вхідних даних.

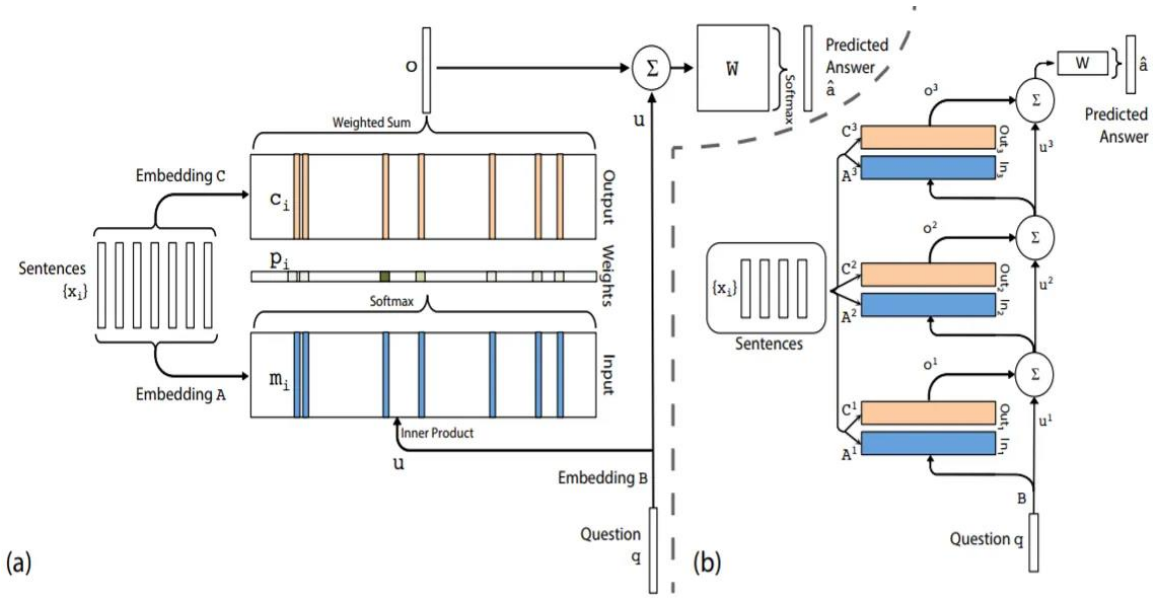


Рисунок 11 – Наскрізні мережі пам’яті

Коли вхідні та вихідні вбудовування однакові на різних шарах, пам’ять ідентична до механізму уваги. Різниця полягає в тому, що він робить кілька переходів через пам’ять (оскільки намагається інтегрувати інформацію з кількох речень).

Ідея глобальної уваги полягає у використанні всіх прихованих станів кодера при обчисленні кожного вектора контексту. Недоліком моделі глобальної уваги є те, що вона повинна враховувати всі слова на стороні джерела для кожного цільового слова, що вимагає значних обчислювальних витрат. Щоб подолати цю проблему, локальна увага спочатку обирає позицію у вихідному реченні. Ця позиція визначатиме вікно слів, на які звертатиме увагу модель.

Attention Sum Reader використовує увагу як вказівник на дискретні лексеми в тексті (рис. 12). Завдання полягає в тому, щоб вибрати відповідь на задане питання з контекстного абзацу. Відмінність від інших методів полягає в тому, що модель вибирає відповідь з контексту безпосередньо за допомогою обчисленої уваги, а не використовує показники уваги для зважування суми прихованих векторів.

Для прикладу розглянемо пару запитань. Нехай контекст буде "НЛО спостерігали над нашим містом у січні і знову в березні", а запитання буде

"Спостерігач помітив НЛО в ...". Січень і березень є однаково хорошими кандидатами, тому попередні моделі присвоять однакові бали уваги. Потім вони обчислюють вектор між представленнями цих двох слів і пропонують слово з найближчим значенням як відповідь. У той же час, Attention Sum Reader правильно запропонує січень або березень, тому що він вибирає слова безпосередньо з уривку.

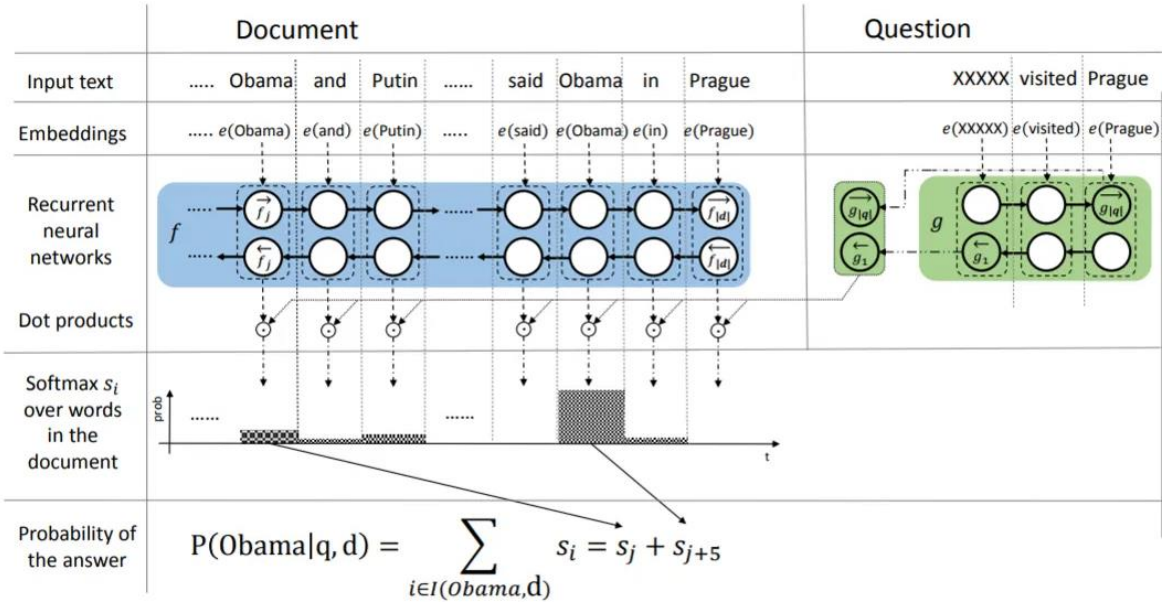


Рисунок 12 – Attention Sum Reader

Як ви могли помітити, в попередній моделі ми звертаємо увагу від джерела до цілі. Це має сенс у перекладі, але як щодо інших сфер? Наприклад, розглянемо текстовий підтекст. У нас є посилка "Якщо ви допомагаєте нужденним, Бог винагородить вас" і гіпотеза "Дати гроші бідній людині має хороші наслідки". Наше завдання – зрозуміти, чи тягне за собою передумова гіпотезу (в даному випадку тягне). Корисно звертати увагу не лише від гіпотези до тексту, а й навпаки.

Так виникає концепція двосторонньої уваги. Ідея полягає в тому, щоб використовувати ту саму модель для уваги як до передумови, так і до гіпотези. У найпростішій формі можна просто поміняти місцями дві послідовності. Таким чином, ми отримуємо два присутнісних представлення, які можна об’єднати.

Однак така модель не дозволить підкреслити важливіші результати поєднання. Наприклад, вирівнювання між стоп-словами є менш важливим, ніж між словами змісту. Крім того, модель все ще використовує один вектор для представлення передумови. Щоб подолати ці обмеження, розробили MatchLSTM (рис. 13). Щоб урахувати важливість зіставлення, вони додають спеціальний LSTM, який запам’ятовує важливі результати зіставлення, забуваючи інші. Цей додатковий LSTM також використовується для підвищення рівня деталізації. Тепер ми помножимо ваги уваги на кожен прихований стан. Він добре показав відповіді на запитання та текстові завдання.

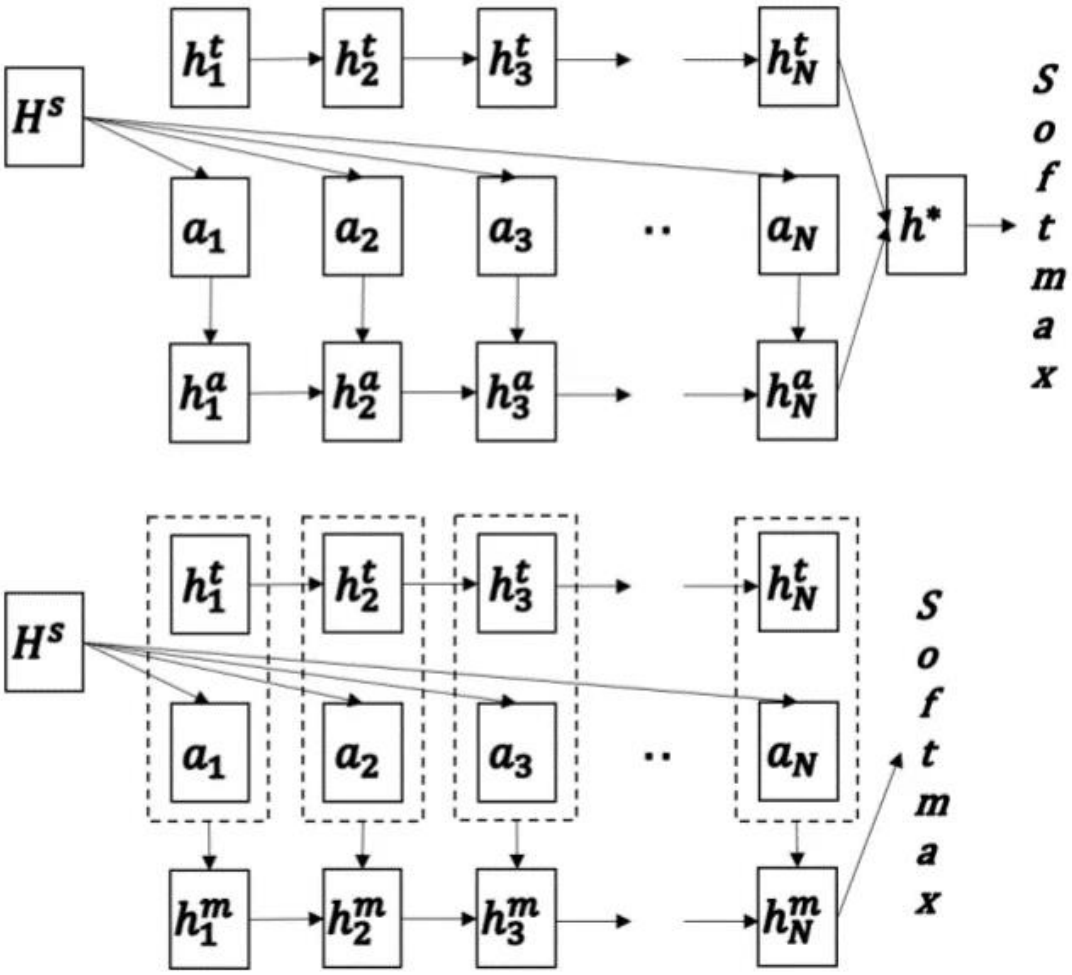


Рисунок 13 – Модель Rocktäschel та MatchLSTM

Завдання з відповідями на запитання породило ще більш просунуті способи поєднання обох сторін. Модель, яку ми бачили на початку, використовує

сумарний вектор запиту для врахування контексту. На відміну від неї, увага до контексту обчислюється як матриця вирівнювання для всіх пар слів контексту і запиту. Як приклад такого підходу розглянемо динамічні мережі уваги (рис. 14).

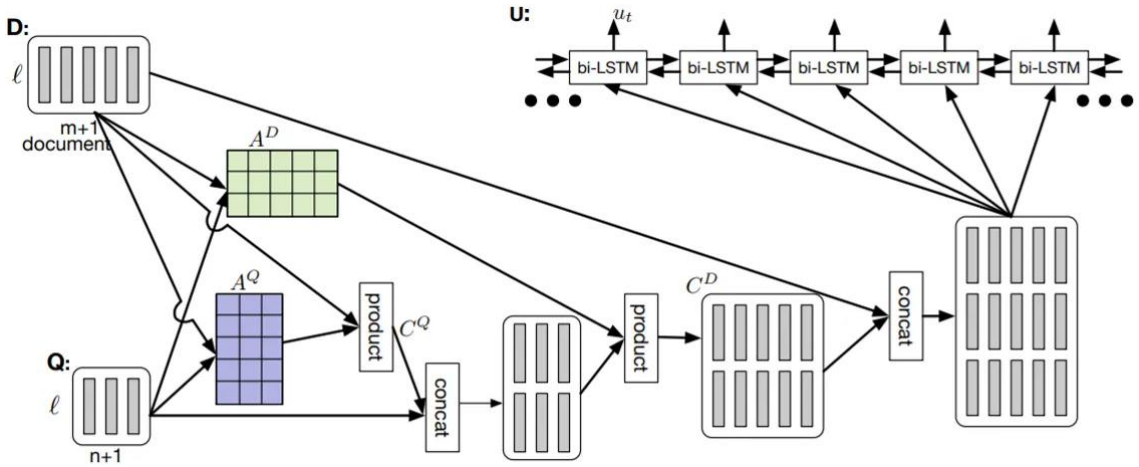


Рисунок 14 – Динамічні мережі уваги

Давайте розглянемо, що відбувається на малюнку. Спочатку ми обчислюємо матрицю спорідненості для всіх пар слів документа та запитання. Потім ми отримуємо ваги уваги AQ по всьому документу для кожного слова в запитанні і AD – навпаки. Далі обчислюється резюме або контекст уваги в документі в світлі кожного слова в запитанні. Таким же чином ми можемо обчислити його для питання в світлі кожного слова в документі. Нарешті, ми обчислюємо резюме попередніх контекстів уваги для кожного слова в документі. Отримані вектори об’єднуються у взаємозалежне представлення питання і документа. Це називається контекстом спільної уваги.

Інша проблема полягає в тому, що рекурентна мережа, незважаючи на свої теоретичні можливості, на практиці може запам’ятовувати лише обмежений контекст уривків. Наприклад, відповідаючи на запитання, один кандидат часто не знає про підказки в інших частинах абзацу.

Самоуважність успішно використовується в різних завданнях. Одним із випадків використання є аналіз настрою. Для таких задач стандартна увага не може бути застосована безпосередньо, тому що немає додаткової інформації:

модель отримує на вхід лише одне речення. Поширеним підходом є використання кінцевого прихованого стану або об’єднання.

У машинному перекладі самоуважність також сприяє досягненню вражаючих результатів. Наприклад, в статті з досить сміливою назвою "Attention Is All You Need" було представлено модель під назвою Transformer. Як можна здогадатися, ця модель покладається лише на самоуважність без використання RNN.

Але, мабуть, найбільш захоплюючою властивістю для лінгвістів було б те, що увага до себе, здається, вивчає складні синтаксичні моделі. Подивіться на приклад, коли мережі вчаться розрізняти анафору в реченні (рис. 15).

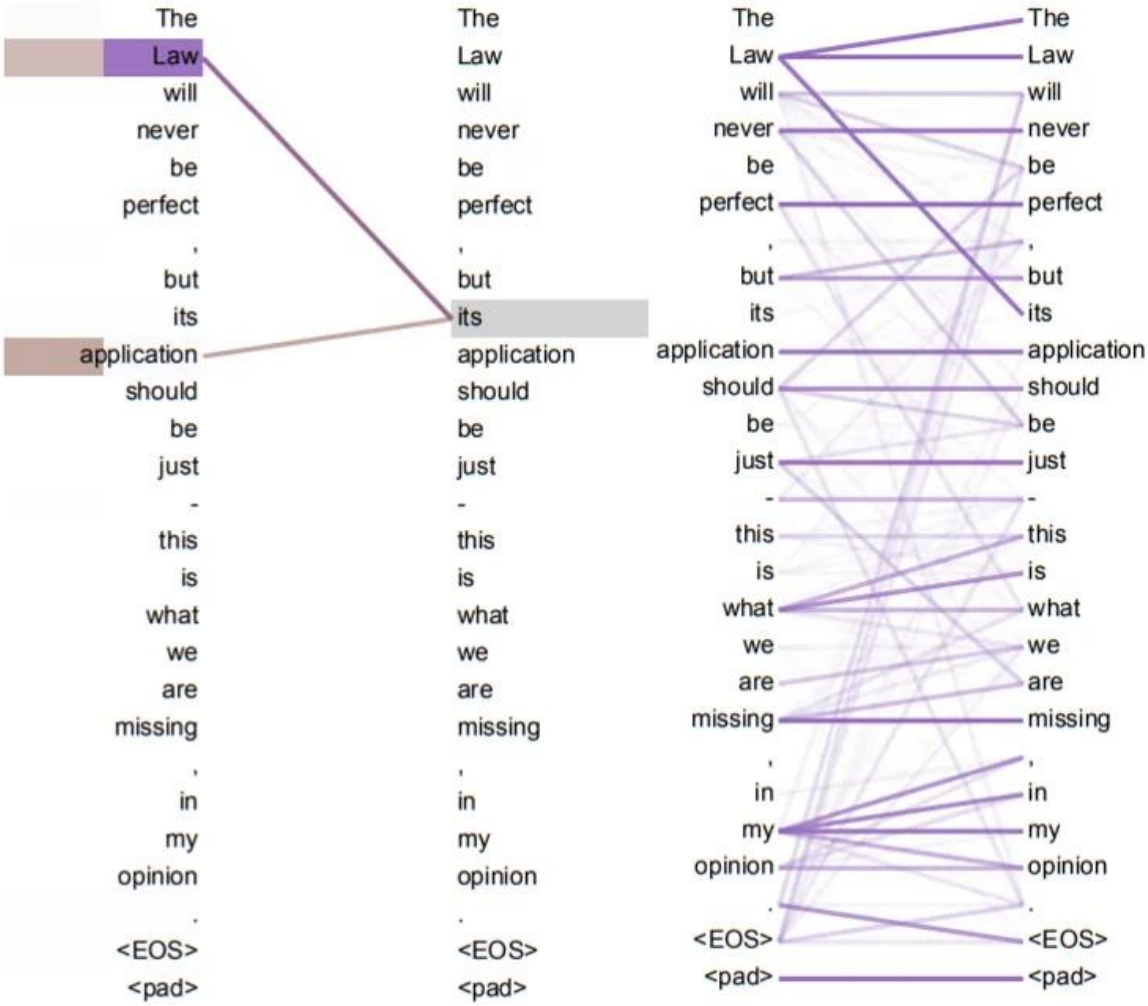


Рисунок 15 – Синтаксичні шаблони, вивчені Transformer

У розглянутих алгоритмах ми вимагаємо, щоб вихідні вектори одночасно зберігали інформацію для передбачення наступного слова, обчислення уваги та кодування вмісту, що стосується майбутніх кроків. Таке перевантажене використання вихідних представлень може зробити навчання невиправдано складним. Для боротьби з цією проблемою запропоновано ключ-значення (прогнозування) уваги. У частині ключ-значення ми розділяємо вихідні вектори на ключі для розрахунку уваги та значення для кодування розподілу наступного слова та представлення контексту.

Однак ми все ще використовуємо частину значень для двох цілей одночасно. Тому знову розділили її, і в підсумку модель виводить три вектори на кожному часовому кроці. Перший використовується для кодування розподілу наступних слів, другий слугує ключем для обчислення вектора уваги, а третій є значенням для механізму уваги (рис. 16).

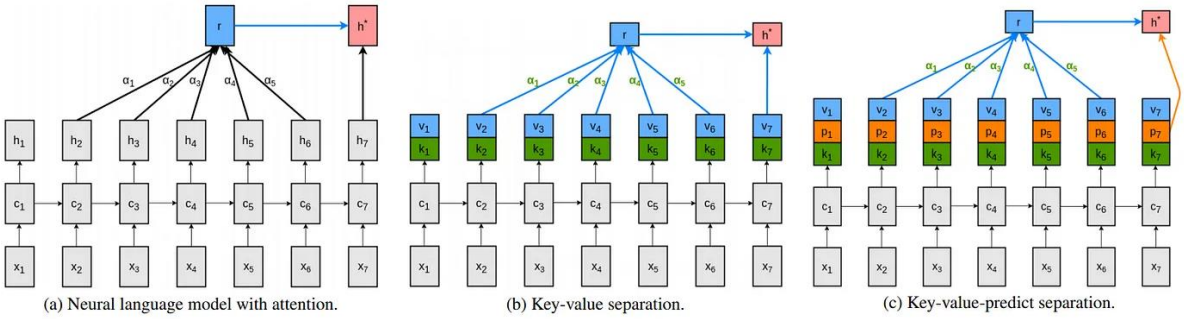


Рисунок 16 – Ключ-значення уваги

Тексти, як правило, мають ієрархічну структуру, а важливість слів і речень сильно залежить від контексту. Щоб врахувати це, ієрархічна модель, використовує два рівні уваги – на рівні слів і на рівні речень. Така архітектура також дозволяє покращити візуалізацію – високоінформативні компоненти документа виділяються. Подібна ідея була представлена в літературі для рівня слів і символів і адаптована до багатомовного середовища в роботі.

Інший погляд на це – Attention-over-Attention. Ідея полягає в тому, щоб розмістити іншу увагу над первинними увагами, щоб вказати на "важливість" кожної з них (рис. 17).

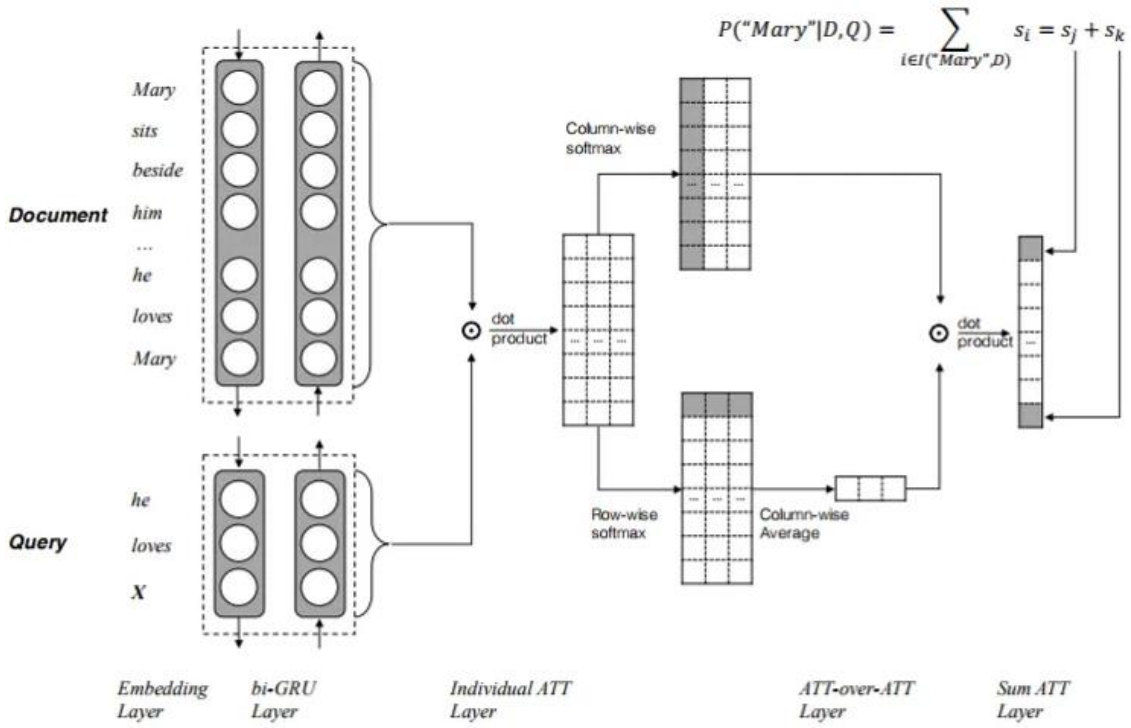


Рисунок 17 – Attention-over-Attention

Attention over-Attention мотивується Attention Sum Reader, який ми бачили раніше. Спочатку вони обчислюють матрицю попарного збігу M шляхом множення контекстних вкладень, створених LSTM для кожної пари слів. Потім вони застосовують softmax по стовпцях, щоб отримати вагові коефіцієнти уваги запиту до документа (альфа). Ваги представляють вирівнювання між усім документом і одним словом запиту. Після цього ваги уваги від документа до запиту отримують шляхом застосування softmax по рядках. Це знову ж форма двосторонньої уваги. Але тепер ми обчислюємо скалярний добуток цих ваг, щоб привернути увагу до уваги.

Остання з більш усталених концепцій, яку я хотів би згадати, це мережа потоку уваги. У такій мережі модель уваги відокремлена від RNN. Ідея була представлена в моделі BiDAF. Увага обчислюється для кожного кроку в часі, і

вектор, що ведеться, разом із представленнями з попередніх шарів може перейти до наступного рівня моделювання. Мета полягає в тому, щоб зменшити втрати інформації, викликані раннім підсумовуванням. Розширення з кількома переходами — це Ruminating Reader.

На зображенні ми бачимо шар потоку уваги (рис. 18). Він відповідає за зв’язування і злиття інформації з контексту і слів запиту за допомогою двосторонньої уваги. Вхідні дані шару – це контекстні векторні представлення контексту і запиту. На виході шару ми отримуємо контекстні векторні представлення слів контексту та контекстні вставки з урахуванням запиту. Зауважте, що ми не створюємо єдиний вектор, а дозволяємо інформації про увагу переходити до наступного шару на кожному часовому кроці.

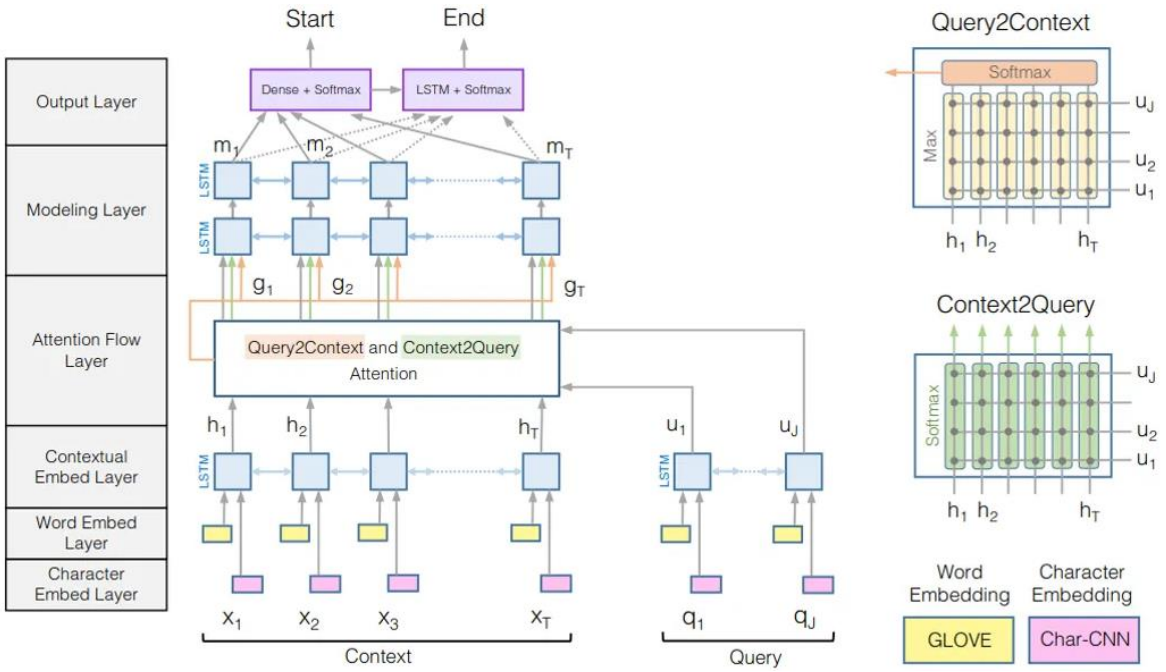


Рисунок 18 – BiDirectional Attention Flow Model

Механізм уваги є одним із найважливіших досягнень у галузі глибокого навчання, зокрема в обробці природної мови. Він значно підвищив продуктивність моделей "від послідовності до послідовності", таких як RNN кодер-декодер, що використовується для задач машинного перекладу, створення діалогів та генерації підписів до зображень. Основна перевага механізму уваги

полягає у здатності вирішувати обмеження традиційної моделі кодера-декодера, яка кодує всю вхідну послідовність у єдиний вектор фіксованої довжини. Це суттєво покращує декодування довгих послідовностей, що є особливо складним завданням для нейронних мереж, коли довжина речень перевищує довжину речень у навчальному корпусі.

Механізм уваги дозволяє моделі зосереджуватися на релевантних частинах вхідної послідовності під час прогнозування кожного наступного слова у вихідній послідовності. Це досягається шляхом створення контекстних векторів, які адаптуються для кожного кроку виведення, замість використання одного фіксованого контекстного вектора для всієї послідовності. Такий підхід забезпечує більш точне та контекстуально релевантне генерування вихідних даних.

1.1.5 Міри схожості

Міри схожості в обробці природної мови (NLP) грають важливу роль у визначенні ступеня схожості між текстовими фрагментами, словами або фразами. Ці міри допомагають в різних завданнях, таких як пошук інформації, класифікація документів, автоматичний переклад, порівняння документів тощо. Зазвичай вони використовуються для порівняння векторних представлень текстів, які створюються за допомогою методів векторизації, таких як Word2Vec, GloVe або BERT.

Для того, щоб комп’ютер міг працювати з текстовими даними, їх потрібно перетворити в певну математичну форму. Тому ми зазвичай представляємо текстові одиниці (символи, слова, речення, абзаци, документи) векторами.

Ідея полягає в тому, щоб представити ці текстові одиниці у вигляді векторів чисел, що уможливить багато операцій, таких як пошук інформації, кластеризація документів, мовний переклад або узагальнення тексту. Таким чином, модель векторного простору, яку ми обираємо для даної задачі, повинна дозволяти нам легко порівнювати документи, вимірюючи відстань між

відповідними векторами, тобто їхню схожість. Існує кілька способів обчислення векторів, які відображають структуру або семантику текстових одиниць.

Перш ніж заглибитися в різні методи представлення тексту, необхідно почати з обговорення деяких методів вимірювання подібності між текстовими одиницями.

Існує велика кількість метрик подібності, які можна використовувати для кількісної оцінки схожості між текстовими одиницями, але кожна з них не має однакового визначення того, що означає "подібний". Залежно від проблеми, з якою ми стикаємося, і залежно від типу подібності, яку ми хочемо виявити між одиницями, ми можемо вибрати одну метрику замість іншої.

Деякі з найпоширеніших способів виявлення схожості між текстовими одиницями такі: найдовший спільний підрядок (LCS), відстань редагування Левенштейна, відстань Гаммінга, косинусна подібність, відстань Жаккара, евклідова відстань.

Найдовший спільний підрядок (LCS) – це концепція, що використовується в обчислювальній теорії і ряді областей обробки текстів, таких як обробка природної мови (NLP) і біоінформатика. Визначається як найдовший спільний фрагмент тексту, який зустрічається у двох або більше послідовностях символів.

LCS є типовим прикладом вимірювання схожості на основі символів. Дано два рядки, s1 довжиною n1 і s2 довжиною n2, він просто враховує довжину найдовшого рядка (або рядків), який є підрядком як s1, так і s2.

Приклад: LCS рядків "Лідія" та "Лідоко" просто повертає 3 (рис. 19).

Відстань Левенштейна є ще одним поширеним прикладом символьної міри подібності. Вона кількісно визначає, наскільки дві текстові одиниці відрізняються одна від одної, обчислюючи мінімальну кількість односимвольних редагувань (операцій заміни, видалення та вставки), необхідних для перетворення текстової одиниці 1 в текстову одиницю 2 (рис. 20).

LCS('Lydia', 'Lydoko') = 3

		L	Y	D	I	A
	0	0	0	0	0	0
L	0	1	0	0	0	0
Y	0	0	2	0	0	0
D	0	0	0	3	0	0
O	0	0	0	0	0	0
K	0	0	0	0	0	0
O	0	0	0	0	0	0

Рисунок 19 – LCS

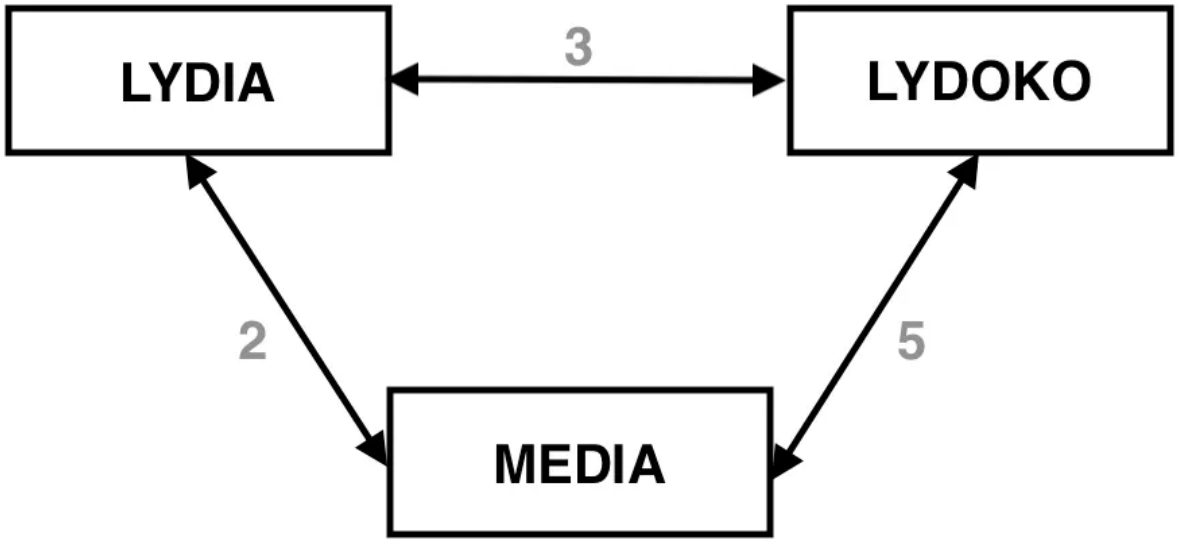


Рисунок 20 – Відстань Левенштейна

Наприклад щоб змінити "hubert" на "uber", знадобляться лише 2 операції, таким чином lev(‘hubert’, ’uber’) = 2.

Ця міра використовується в різних областях, таких як комп’ютерна лінгвістика, біоінформатика та інформаційний пошук. Наприклад, в пошукових системах вона може використовуватися для виправлення орфографічних помилок або автозаповнення запитів. У біоінформатиці вона може визначати

генетичну подібність між послідовностями ДНК або білка. Також корисна і в алгоритмах для автоматичної обробки тексту, таких як переклад або розпізнавання мови.

Іншим показником схожості символів є відстань Хеммінга. Відстань Хеммінга між двома рядками однакового розміру вимірює мінімальну кількість замін, необхідних для зміни одного рядка на інший.

Приклад: відстань Хеммінга між рядками "Лідія" та "Медіа" дорівнює 2 (рис. 21).

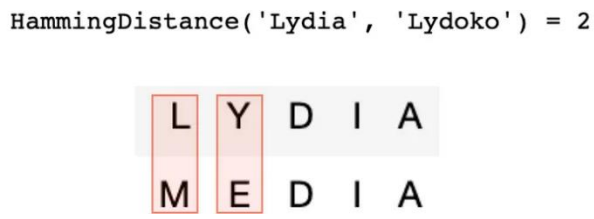


Рисунок 21 – Відстань Хеммінга

Відстань Хеммінга є важливим поняттям у теорії кодування і використовується в телекомунікаціях, комп’ютерних науках, криптографії та інших областях. Вона допомагає визначати якість передачі даних через шумні канали зв’язку та дозволяє виявити та виправити помилки в передачі інформації. Також вона широко використовується в генетиці для порівняння геномних послідовностей та в інших областях, де потрібно оцінювати схожість між послідовностями символів однакової довжини.

Косинусна подібність – це міра схожості між двома векторами у просторі. У контексті аналізу тексту, кожен текст може бути представлений як вектор, де кожне слово представлене як окрема компонента. Зазвичай, значення компоненти вектора відповідає частоті зустрічання слова у тексті.

Щоб обчислити косинусну подібність між двома текстами, спочатку представляють кожен текст у вигляді вектора. Потім обчислюють кут між цими векторами за допомогою косинуса відповідного кута. Чим ближче кут між векторами до 0 градусів (або косинус ближче до 1), тим більша подібність між

текстами. Коли значення близько до 1, дві одиниці знаходяться поруч у вибраному векторному просторі, коли близько до -1, дві одиниці знаходяться далеко одна від одної.

Тому ця міра є лише оцінкою орієнтації, а не величини: два вектори з однаковою орієнтацією мають косинусну подібність 1, два вектори, орієнтовані під кутом 90° один до одного, мають подібність 0, а два діаметрально протилежні вектори мають подібність -1 (рис. 22).

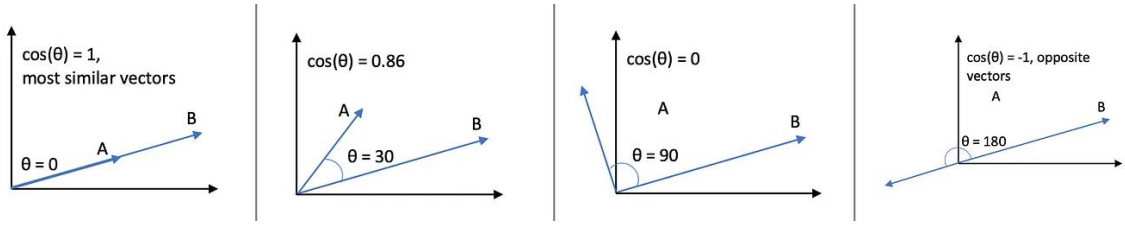


Рисунок 22 – Подібність косинусів у 2-вимірному просторі

Косинусна подібність широко використовується в різних задачах обробки тексту, таких як пошукові системи, рекомендаційні системи, класифікація тексту та інші. Вона дозволяє визначити, наскільки текстові документи схожі за їх змістом незалежно від їх розміру або внутрішньої структури.

Відстань Жаккара – це міра схожості між двома множинами, що вимірює ступінь спільності між ними. У контексті аналізу тексту, відстань Жаккара використовується для визначення схожості між двома множинами слів або термінів, які зустрічаються у текстах.

Щоб обчислити відстань Жаккара для двох текстів, спочатку визначають множини термінів (слів), що входять у кожен текст. Потім визначають кількість спільних термінів між двома текстами та загальну кількість унікальних термінів у обох текстах. Обчислюється як відношення кількості спільних термінів до загальної кількості унікальних термінів у двох текстах.

Відстань Жаккара, що використовується для порівняння текстових одиниць, представлених у вигляді BoW, зазвичай має певні недоліки: зі збільшенням розміру документа кількість спільних слів має тенденцію до

збільшення, навіть якщо документи стосуються різних тем. Крім того, ця міра не зможе відобразити схожість між різними текстовими одиницями, які мають однакове значення, але написані по-різному (це питання більше стосується представлення тексту, але оскільки відстань Жаккара особливо добре підходить для стратегії BoW, воно все одно викликає занепокоєння). Евклідова відстань – це міра відстані між двома точками у n-вимірному просторі. Кожен текст може бути представлений як вектор у n-вимірному просторі, де кожна компонента вектора відповідає певній визначеній характеристиці тексту, такі як частота вживання певних слів, тематичні ознаки тощо.

Щоб обчислити евклідову відстань між двома текстами, спочатку представляють кожен текст у вигляді вектора. Потім обчислюють різницю між векторами кожного тексту та обчислюють евклідову відстань між цими різницями.

Евклідова відстань може використовуватися для порівняння текстів у векторному просторі, де кожен текст представлений як вектор ознак. Це може бути корисно для задач, таких як кластеризація текстів, пошук схожих документів тощо. Однак вона може бути менш ефективною для текстів у великому просторі ознак через проблему розрідженості та вимірювальних шкал.

Отже, міри схожості в обробці природної мови є ключовим інструментом для аналізу та порівняння текстових даних. Вони допомагають в різних завданнях NLP, таких як пошук інформації, класифікація документів, автоматичний переклад і багато інших. Перетворення тексту в векторний формат дозволяє використовувати різні методи для оцінки схожості між текстовими одиницями. Метрики подібності, такі як LCS, відстань Левенштейна, косинусна подібність, відстань Жаккара та евклідова відстань, надають різні підходи до вимірювання подібності, що дозволяє вибирати найбільш підходящий метод залежно від конкретної задачі.. Розроблений для підвищення продуктивності моделі RNN кодера декодера (seq2seq).

2 ЗАГАЛЬНИЙ ОПИС МЕТОДУ

Метод призначений для оцінки схожості між правильними назвами компаній та їх варіантами, до яких додані загальні слова (такі як "Inc", "Ltd", "Corporation", "Company"). Використовується модель BERT для створення векторних представлень текстів і обчислення косинусної схожості між ними. Нижче наведено опис кожного кроку процесу:

а) імпорт необхідних бібліотек:

1) pandas: використовується для роботи з табличними даними, зокрема для завантаження та збереження даних у форматі Excel;

2) transformers: бібліотека, яка містить попередньо навчені моделі, такі як BERT, для обробки природної мови;

3) torch: бібліотека для роботи з тензорами та обчислень на GPU, яка використовується для обчислення ембеддингів та косинусної схожості;

б) ініціалізація токенізатора та моделі BERT:

1) Завантаження попередньо навчених вагів моделі BERT і відповідного токенізатора (bert-base-uncased);

2) Перехід моделі в режим оцінки (eval), що відключає градієнти і деякі механізми, які використовуються під час тренування моделі, для підвищення ефективності обчислень;

в) функція для отримання ембеддингів тексту (рис. 23):

```
def get_text_embeddings(texts, batch_size=16):
    embeddings = []
    for i in range(0, len(texts), batch_size):
        batch_texts = texts[i:i+batch_size]
        encoded_input = tokenizer(batch_texts, return_tensors='pt', padding=True, truncation=True, max_length=128)
        with torch.no_grad():
            output = model(**encoded_input)
        batch_embeddings = output.last_hidden_state.mean(dim=1)
        embeddings.extend(batch_embeddings)
    return torch.stack(embeddings)
```

Рисунок 23 – Отримання векторних представлень тексту

1) функція приймає список текстів і розбиває їх на пакети (batch)

розміром batch_size (за замовчуванням 16);

2) для кожного пакета текстів здійснюється їх токенізація за допомогою BERT токенізатора, що перетворює тексти в числовий формат, придатний для обробки моделлю;

3) модель BERT обчислює вихідні приховані стани для кожного токенізованого тексту, які потім усереднюються по всіх токенах, щоб отримати одне векторне представлення для кожного тексту;

4) отримані ембеддинги додаються до загального списку і повертаються у вигляді тензора;

г) функція для додавання загальних слів до назви (рис. 24):

1) приймає назву компанії та список загальних слів;

2) повертає список нових назв, де до кожної назви додається кожне загальне слово зі списку;

```
def add_common_words(name, common_words):
    return [f"{name} {word}" for word in common_words]
```

Рисунок 24 – Додавання загальних слів

д) завантаження даних з Excel файлу (рис. 25):

1) завантажується файл Excel, що містить правильні назви компаній;

2) формується список правильних назв компаній, при цьому порожні значення відсіюються;

```
df = pd.read_excel("Назви_компаній.xlsx")
correct_names = df["Компанія"].dropna().tolist()
```

Рисунок 25 – Завантаження даних

е) визначення загальних слів і генерація варіантів назв (рис. 26):

1) створюється список загальних слів, таких як "Inc", "Ltd", "Corporation", "Company";

2) для кожної правильної назви генеруються всі можливі варіанти з

додаванням кожного загального слова, створюючи комбіновані назви;

```
common_words = ["Inc", "Ltd", "Corporation", "Company"]
all_variants = []
for name in correct_names:
    all_variants.extend(add_common_words(name, common_words))
```

Рисунок 26 – Генерація варіантів назв

ж) отримання ембеддингів для правильних назв і їх варіантів:

1) викликаються функції для отримання ембеддингів для списку правильних назв і списку зіпсованих варіантів;

з) функція для порівняння схожості (рис. 27):

1) приймає списки правильних назв, варіантів і їх ембеддингів;

2) обчислює косинусну схожість між кожною правильною назвою і кожним варіантом;

3) перевіряє, чи є варіант правильною назвою з доданим загальним словом;

4) повертає список зі значеннями схожості та булевим значенням, яке вказує, чи є варіант правильним;

```
def compare_company_names_similarity(correct_names, all_variants, correct_embeddings, variant_embeddings, common_words):
    similarities = []
    for i, correct_name in enumerate(correct_names):
        for j, variant in enumerate(all_variants):
            similarity = cosine_similarity(correct_embeddings[i], variant_embeddings[j], dim=0).item()
            is_correct = any(correct_name + " " + word == variant for word in common_words)
            similarities.append((correct_name, variant, similarity, is_correct))
    return similarities
```

Рисунок 27 – Порівняння схожості

и) створення DataFrame і збереження результатів (рис. 28):

1) викликається функція для порівняння схожості, і результати зберігаються у DataFrame;

2) дані сортуються за спаданням значення схожості;

3) додається колонка з рангом, яка визначає порядок варіантів за схожістю для кожної правильної назви;

```
similarities = compare_company_names_similarity(correct_names, all_variants, correct_embeddings, variant_embeddings, common_words)
results_df = pd.DataFrame(similarities, columns=["Назва 1", "Назва 2", "Схожість", "Відповідь"])
results_df = results_df.sort_values(by="Схожість", ascending=False)
results_df['Ранг'] = results_df.groupby('Назва 1').cumcount() + 1
results_df["Відповідь"] = results_df["Відповідь"].apply(lambda x: "True" if x else "False")
results_df.to_excel("Результати_схожості_компаній.xlsx", index=False)
```

Рисунок 28 – DataFrame і збереження результатів

4) булеві значення "Відповідь" конвертуються у строки "True" і "False";

5) результати зберігаються у файл Excel;

і) створення рангових результатів і збереження у Excel (рис. 29):

1) визначаються пари правильних і зіпсованих назв, де "Відповідь" є "True";

2) для кожної правильної назви обчислюється середній ранг правильних варіантів;

3) створюється DataFrame з ранговими результатами і зберігається у файл Excel.

```
rank_results = []
for correct_name in correct_names:
    correct_pairs = results_df[(results_df['Назва 1'] == correct_name) & (results_df['Відповідь'] == "True")]
    if not correct_pairs.empty:
        avg_rank = correct_pairs['Ранг'].mean()
        for _, row in correct_pairs.iterrows():
            rank_results.append((correct_name, row['Назва 2'], row['Ранг'], avg_rank))
rank_df = pd.DataFrame(rank_results, columns=["Назва 1", "Назва 2", "Ранг", "Середній ранг"])
rank_df.to_excel("Ранги_схожості_компаній.xlsx", index=False)
```

Рисунок 29 – Визначення рангу

Цей метод забезпечує ефективну оцінку схожості між назвами компаній за допомогою моделі BERT. Векторні представлення текстів і обчислення косинусної схожості дозволяють точно порівнювати правильні назви з їх варіантами, до яких додані загальні слова. Результати зберігаються у вигляді таблиць Excel, що містять значення схожості та ранги для кожної пари назв. Це корисно для нормалізації та очищення даних, що особливо важливо при роботі з великими обсягами текстової інформації.

3 РЕАЛІЗАЦІЯ МЕТОДУ

Для оцінки ефективності моделі BERT у виявленні схожості між оригінальними назвами компаній та їх варіантами з доданими загальними словами було проведено детальний аналіз результатів порівнянь. Цей аналіз допомагає зрозуміти, як добре модель справляється з завданням ідентифікації схожих назв. Розглянемо детальніше приклади порівнянь, які були здійснені в результатах програми (рис. 30). Отже, беремо деякі приклади із початкового набору даних:

	Назва 1	Назва 2	Схожість	Відповідь	Ранг
2	Cognizant	Cognizant	0,970578	True	1
3	Cognizant	Cognizant	0,951982	True	2
4	Cognizant	Cognizant	0,949147	True	3
5	Cognizant	Cognizant	0,932224	True	4
6	United Sta	United Sta	0,929932	True	1
7	Walmart	Walmart I	0,924324	True	1
8	Delta Air L	Delta Air L	0,923718	True	1
9	Bank of Ar	Bank of Ar	0,92214	True	1
10	Delta Air L	Delta Air L	0,919111	True	2

Рисунок 30 – Приклади порівнянь

а) Cognizant Technology Solutions:

1) Cognizant Technology Solutions Company: Найвищий показник схожості – 0.970578. Це означає, що додавання слова "Company" до правильної назви призвело до створення варіанта, який дуже схожий на оригінал;

2) Cognizant Technology Solutions Inc: Другий за схожістю варіант (0.951982). Це також свідчить про високу схожість, що вказує на те, що додавання "Inc" незначно змінює сенс або структуру тексту з точки зору BERT;

3) Cognizant Technology Solutions Corporation: Показник схожості – 0.949147. Знову ж таки, висока схожість;

4) Cognizant Technology Solutions Ltd: Найнижчий серед цих варіантів показник – 0.932224. Хоча цей показник трохи нижчий, він все одно свідчить про високу схожість;

б) United States Air Force:

1) United States Air Force Inc: Показник схожості – 0.929932. Це свідчить про те, що додавання "Inc" до назви "United States Air Force" також зберігає значну схожість з оригіналом;

в) Walmart:

1) Walmart Inc: Показник схожості – 0.924324. Цей високий показник підтверджує, що додавання "Inc" до коротшої назви, як "Walmart", також призводить до високої схожості;

г) Delta Air Lines:

1) Delta Air Lines Company: Показник схожості – 0.923718. Додавання слова "Company" створює варіант, який дуже схожий на оригінал;

2) Delta Air Lines Inc: Показник схожості – 0.919111. Так само, як і в попередньому випадку, додавання "Inc" майже не змінює загальної схожості;

д) Bank of America:

1) Bank of America Inc: Показник схожості – 0.922140. Це вказує на те, що варіант з "Inc" дуже схожий на оригінальну назву.

Під час оцінки якості результатів порівняння назв компаній та їх варіантів з доданими загальними словами використовуються кілька критеріїв, які вказують на ефективність моделі та правильність її роботи. Основні аспекти оцінки якості включають:

а) схожість: Високі показники схожості (близькі до 1) свідчать про те, що модель успішно ідентифікує схожість між оригінальною назвою компанії та її варіантами з додаванням загальних слів. Наприклад, всі варіанти для "Cognizant Technology Solutions" мають показники схожості вище 0.93, що є ознакою відмінної якості моделі для цієї задачі;

б) відповідь: Всі варіанти з додаванням загальних слів позначені як "True" у колонці "Відповідь", що свідчить про те, що модель правильно розпізнає ці варіанти як істинні доповнення до оригінальної назви;

в) ранг: Висока схожість та правильне сортування варіантів за спаданням схожості підтверджують, що модель надає пріоритет найбільш схожим варіантам. Наприклад, варіанти для "Cognizant Technology Solutions" мають ранги від 1 до 4, що показує чітке ранжування за схожістю.

Загальний аналіз результатів демонструє високу якість роботи моделі. BERT успішно визначає схожість між оригінальними назвами компаній та їх варіантами з доданими загальними словами. Це показано високими показниками схожості для більшості порівнянь. Додавання загальних слів, таких як "Inc", "Ltd", "Corporation" та "Company", незначно впливає на схожість, що свідчить про те, що ці слова не сильно змінюють контекст назви з точки зору моделі BERT. Модель ефективно сортує варіанти за рівнем схожості, що дозволяє легко ідентифікувати найбільш схожі варіанти для кожної правильної назви. Це підтверджується рангами, які присвоєні кожній парі. Розподіл показників схожості вказує на те, що більшість варіантів мають високу схожість з оригіналами, що свідчить про стабільність та надійність моделі.

Щоб досягти ще кращих результатів, можна розглянути кілька шляхів покращення. По-перше, можна провести тонке налаштування моделі, зокрема додаткове тренування (fine-tuning) моделі BERT на специфічному наборі даних з назвами компаній, щоб підвищити її точність для цієї конкретної задачі. Також важливо розширити набір даних, додаючи більше прикладів з різними варіантами назв компаній, що може допомогти покращити загальну продуктивність моделі. Крім того, варто досліджувати інші моделі трансформерів, такі як RoBERTa або GPT, які можуть показати кращу продуктивність для даної задачі. Нарешті, необхідно провести детальний аналіз помилок, щоб зрозуміти причини низької схожості у деяких випадках та внести відповідні корективи.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було розроблено нейромережеву систему для порівняння схожості назв компаній, яка може бути використана для автоматизації процесу виявлення правильних та неправильних назв компаній. Ця система використовує модель BERT для отримання векторних представлень тексту та функцію косинусної схожості для оцінки схожості між назвами.

У процесі виконання кваліфікаційної роботи було реалізовано основні цілі, що були сформульовані на етапі початкового планування. Вони включають: завантаження та обробку даних із файлу Excel; визначення загальних слів, які слід додати до назв компаній; генерацію всіх можливих зіпсованих назв; отримання векторних представлень для всіх назв за допомогою моделі BERT; порівняння схожості між правильними та зіпсованими назвами; створення таблиці результатів та впорядкування її за значенням схожості; визначення рангів для кожної правильної назви та відповідної зіпсованої назви.

Оцінка якості є одним із ключових завдань для будь-якої моделі машинного навчання. В даному випадку була використана функція косинусної схожості для оцінки подібності між правильними та зіпсованими назвами компаній. В результаті було досягнуто високої точності у виявленні правильних назв з доданими загальними словами.

Розроблений продукт успішно виконує всі поставлені завдання. Результати кваліфікаційної роботи повністю відповідають поставленим цілям, а створена система може ефективно використовуватися для автоматизації процесу виявлення правильних назв компаній та покращення якості даних у відповідних застосунках.

ПЕРЕЛІК ПОСИЛАНЬ

1. Devlin J., Chang M.-W., Lee K., Toutanova K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. URL : https://arxiv.org/abs/1810.04805 (дата звернення: 02.02.2024).

2. Mikolov T., Chen K., Corrado G., Dean J. Efficient Estimation of Word Representations in Vector Space. URL : https://arxiv.org/abs/1301.3781 (дата звернення: 10.02.2024).

3. Bojanowski P., Grave E., Joulin A., Mikolov T. Enriching Word Vectors with Subword Information. Transactions of the Association for Computational Linguistics. URL : https://arxiv.org/abs/1607.04606 (дата звернення: 12.02.2024).

4. Pennington J., Socher R., Manning C. D. GloVe: Global Vectors for Word Representation. URL : https://www.aclweb.org/anthology/D14-1162/ (дата звернення: 25.02.2024).

5. Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A. N., Polosukhin I. Attention is All you Need. URL : https://arxiv.org/abs/1706.03762 (дата звернення: 07.03.2024).

6. Wolf T., Debut L., Sanh V., Chaumond J., Delangue C., Moi A., Rush A. M. Transformers: State-of-the-Art Natural Language Processing. URL : https://www.aclweb.org/anthology/2020.emnlp-demos.6/ (дата звернення: 23.03.2024).

7. Pedregosa F., Varoquaux G., Gramfort A., Michel V., Thirion B., Grisel O., Duchesnay É. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research. URL : http://jmlr.org/papers/v12/pedregosa11a.html (дата звернення: 05.04.2024).

8. Jurafsky D., Martin J. H. Speech and Language Processing. Third Edition draft. January 7, 2023. 628 pp. URL : https://web.stanford.edu/~jurafsky/slp3/ (дата звернення: 15.04.2024).

9. Manning C. D., Raghavan P., Schütze H. Introduction to Information Ret rieval. Cambridge University Press. URL : https://nlp.stanford.edu/IR-book/ (дата звернення: 19.04.2024).

10. Bird S., Klein E., Loper E. Natural Language Processing with Python. O'Reilly Media. Printing History. June 2009. 479 pp.

11. Murphy K. P. Machine Learning: A Probabilistic Perspective. MIT Press. 2012. 1104 pp.

12. Alpaydin E. Introduction to Machine Learning. Fourth Edition. MIT Press. 2020. 712 pp.

13. Raschka S., Mirjalili V. Python Machine Learning. Third Edition. Packt Publishing. 2019. 770 pp.

14. Brownlee J. Deep Learning for Natural Language Processing. Machine Learning Mastery. 2017. 414 pp.

15. Rogers A., Kovaleva O., Rumshisky A. A Primer in BERTology: What We Know About How BERT Works. 2020. URL : https://aclanthology.org/202 0.tacl-1.54/ (дата звернення: 15.05.2024).

ДОДАТОК А

Повний текст програми

main.py

```
import pandas as pd
from transformers import BertTokenizer, BertModel
import torch
from torch.nn.functional import cosine_similarity

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased',
output_hidden_states=True)
model.eval()

def get_text_embeddings(texts, batch_size=16):
    embeddings = []
    for i in range(0, len(texts), batch_size):
        batch_texts = texts[i:i+batch_size]
        encoded_input = tokenizer(batch_texts, return_tensors='pt',
padding=True, truncation=True, max_length=128)
        with torch.no_grad():
            output = model(**encoded_input)
        batch_embeddings = output.last_hidden_state.mean(dim=1)
        embeddings.extend(batch_embeddings)
    return torch.stack(embeddings)

def add_common_words(name, common_words):
    return [f"{name} {word}" for word in common_words]
```

```
df = pd.read_excel("Назви_компаній.xlsx")

correct_names = df["Компанія"].dropna().tolist()

common_words = ["Inc", "Ltd", "Corporation", "Company"]

all_variants = []
for name in correct_names:
    all_variants.extend(add_common_words(name, common_words))

correct_embeddings = get_text_embeddings(correct_names)
variant_embeddings = get_text_embeddings(all_variants)

def compare_company_names_similarity(correct_names, all_variants,
correct_embeddings, variant_embeddings, common_words):
    similarities = []
    for i, correct_name in enumerate(correct_names):
        for j, variant in enumerate(all_variants):
            similarity = cosine_similarity(correct_embeddings[i],
variant_embeddings[j], dim=0).item()
            is_correct = any(correct_name + " " + word == variant for word in
common_words)
            similarities.append((correct_name, variant, similarity, is_correct))
    return similarities

similarities = compare_company_names_similarity(correct_names,
all_variants, correct_embeddings, variant_embeddings, common_words)

results_df = pd.DataFrame(similarities, columns=["Назва 1", "Назва 2",
```

```
"Схожість", "Відповідь"])

results_df = results_df.sort_values(by="Схожість", ascending=False)

results_df[‘Ранг’] = results_df.groupby(‘Назва 1’).cumcount() + 1

results_df["Відповідь"] = results_df["Відповідь"].apply(lambda x: "True" if
x else "False")

results_df.to_excel("Результати_схожості_компаній.xlsx", index=False)

rank_results = []
for correct_name in correct_names:
    correct_pairs = results_df[(results_df[‘Назва 1’] == correct_name) &
(results_df[‘Відповідь’] == "True")]
    if not correct_pairs.empty:
        avg_rank = correct_pairs[‘Ранг’].mean()
        for _, row in correct_pairs.iterrows():
            rank_results.append((correct_name, row[‘Назва 2’], row[‘Ранг’],
avg_rank))

rank_df = pd.DataFrame(rank_results, columns=["Назва 1", "Назва 2",
"Ранг", "Середній ранг"])

rank_df.to_excel("Ранги_схожості_компаній.xlsx", index=False)
```