

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «магістр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Розробка та аналіз системи тестів англійської мови на основі нейронних мереж»

Виконав студент групи ІПЗм-22-1
ПРАНОВ Любомир Іванович

Керівник роботи:
ВЛАСЕНКО Олег Васильович

Рецензент:
КРАВЧЕНКО Світлана Миколаївна

Житомир – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

зав. кафедри інженерії
програмного забезпечення

Тетяна ВАКАЛЮК

«18» вересня 2023 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ПРАНОВ Любомир Іванович**

Керівник роботи: **ВЛАСЕНКО Олег Васильович**

Тема роботи: **«Розробка та аналіз системи тестів англійської мови на основі нейронних мереж»,**

затверджена Наказом закладу вищої освіти від **«18» вересня 2023 р. № 479/с**

Вихідні дані для роботи: Предметом дослідження є методи та алгоритми автогенерації запитань, правильних та неправильних відповідей на основі тексту англійської мови. Об'єктом дослідження є автогенерація запитань, правильних та неправильних відповідей на основі тексту англійської мови.

Консультанти з магістерської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Власенко О.В.	22.09.2023р.	22.09.2023р.
2	Власенко О.В.	22.09.2023р.	22.09.2023р.
3	Власенко О.В.	22.09.2023р.	22.09.2023р.

РЕФЕРАТ

Кваліфікаційна магістерська робота присвячена науковому вивченню генерації питань із англійського тексту з використанням моделей обробки природної мови, правильних та неправильних відповідей.

Ця робота складається з програмного пакета, що містить API сервера, та його описи. Пояснювальна записка складається зі списку літератури, включаючи записи, 3 розділи, висновки та 19 заголовків. Робота також містить 25 рисунків/діаграм та 4 таблиці.

Кваліфікаційна магістерська робота на тему: «Розробка та аналіз системи тестів англійської мови на основі нейронних мереж» складається з теоретичного та програмного комплексу системи та пояснювальної записки. Пояснювальна записка до кваліфікаційної роботи складається зі вступу, 3 частини та списку літератури з 33 найменувань.

У першому розділі визначено основні вимоги, проаналізовано проблему, вивчено методи та алгоритми її вирішення. У розділі 2 було проведено апробацію реалізації своєї написаної моделі для генерації завдань. У 3 розділі було охарактеризовано джерело даних та реалізовано практичну реалізацію алгоритму на передбаченій моделі GPT-2.

Пакет програмного забезпечення було протестовано.

Результати включають результати проведених досліджень.

КЛЮЧОВІ СЛОВА: МОДЕЛЬ, ТРАНСФОРМЕР, НАБІР ДАНИХ, ГЕНЕРАЦІЯ ПИТАНЬ, ПРАВИЛЬНІ ТА НЕПРАВИЛЬНІ ВІДПОВІДІ.

ABSTRACT

The qualified master's thesis is devoted to the scientific study of the generation of questions from the English text using natural language processing models, correct and incorrect answers.

This work consists of a software package containing the server API and its descriptions. The explanatory note consists of a list of references, including entries, 3 chapters, conclusions and 19 headings. The work also contains 25 figures/diagrams and 4 tables.

Qualified master's thesis on the topic: "Development and analysis of the system of English language tests based on neural networks" consist of the theoretical and software complex of the system and an explanatory note. The explanatory note to the qualification paper consists of an introduction, 3 parts and a list of 33 titles.

In the first chapter, the basic requirements are defined, the problem is analyzed, and the methods and algorithms for solving it are studied. In chapter 2, the implementation of the written model for task generation was tested. In the 3rd chapter, the data source was characterized and the practical implementation of the algorithm was realized on the predicted GPT-2 model.

The software package has been tested.

The results include the results of the conducted study.

KEYWORDS: MODEL, TRANSFORMER, DATASET, QUEST GENERATION, CORRECT AND INCORRECT ANSWERS.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

RNN - Recurrent Neural Networks

CNN - Convolutional Neural Networks

LSTM - Long Short-Term Memory

GRU - Gated Recurrent Unit

BERT - Bidirectional Encoder Representations from Transformers

GPT-2 - Generative Pre-trained Transformer 2

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	5
ВСТУП	7
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ	11
1.1. Генерація тексту за допомогою RNN	11
1.2. Генерація тексту за допомогою CNN	16
1.3. Генерація тексту за допомогою Transformer	18
1.4. Огляд BERT (Bidirectional Encoder Representations from Transformers)..	20
1.5. Порівняння готових програмних продуктів	21
РОЗДІЛ 2. ОПИС НАБОРУ ДАНИХ ДЛЯ ГЕНЕРАЦІЇ ТЕСТОВИХ ЗАВДАНЬ З АНГЛІЙСЬКОЇ МОВИ ДЛЯ НЕЙРОННОЇ МЕРЕЖІ	26
2.1. Опис набору даних.....	26
2.2. Застосування самонавчальної моделі нейронної мережі для генерації тестових завдань.....	27
2.3. Детальний аналіз Generative Pre-trained Transformer 2 (GPT-2).	35
2.4. Обґрунтування вибору Generative Pre-trained Transformer 2 (GPT-2).	41
РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ АЛГОРИТМІВ РОБОТИ З НЕЙРОННОЮ МЕРЕЖОЮ GPT-2.....	45
3.1. Підготовка даних і завантаження бібліотек.....	45
3.2. Підключення функцій генерації питань.....	47
ВИСНОВОК	54
ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ	56

ВСТУП

Актуальність теми. Сучасний світ все більше орієнтується на глобалізацію, що наголошує на важливості володіння англійською мовою як ключовим інструментом комунікації. У цьому контексті розробка ефективних методів оцінки мовних навичок стає питанням важливості. Нейронні мережі, як частина великої галузі штучного інтелекту, надають унікальні можливості для створення інноваційних та точних систем тестування з англійської мови.

Також актуальність визначається сучасними потребами та тенденціями у сфері освіти та технологій. Ось деякі аспекти, які підкреслюють актуальність цієї теми:

- **Оновлення освітніх підходів:** З урахуванням швидкого технологічного розвитку важливо впроваджувати інновації в оцінювання мовних навичок через використання нейронних мереж у тестових системах.
- **Збільшений інтерес до вивчення англійської мови:** Рістущий попит на вивчення англійської визначає потребу в ефективних тестових інструментах для об'єктивного вимірювання рівня знань.
- **Застосування машинного навчання:** Використання нейронних мереж у вивченні мов відкриває перспективи для поліпшення точності та об'єктивності процесу оцінювання мовних навичок.
- **Глобальна взаємодія:** В умовах зростання міжнародного спілкування важливо мати тести, які враховують різноманітність мовних відмінностей та відповідають високим стандартам.
- **Автоматизація та оптимізація:** Розробка систем тестів на базі нейронних мереж може автоматизувати процеси проведення тестів, що сприяє збереженню часу та ресурсів.

Таким чином, вивчення та розробка систем тестів англійської мови на основі нейронних мереж є важливим напрямком досліджень, спрямованим на вдосконалення методів оцінювання та підвищення ефективності мовного навчання.

Дана магістерська дисертація присвячена розробці та аналізу системи тестів з англійської мови, заснованої на принципах нейронних мереж. Використання сучасних методів машинного навчання та глибокого навчання дозволяє покращити точність та об'єктивність оцінки мовних навичок студентів та професіоналів.

У ході дослідження будуть розглянуті основні тенденції у галузі тестування з англійської мови, проблеми існуючих методів та потенційні переваги, які можуть бути досягнуті з використанням нейронних мереж. Аналіз результатів тестування та порівняння їх із традиційними підходами дозволить оцінити ефективність запропонованої системи.

Дана дисертація не лише надасть нового погляду на проблему тестування з англійської мови, а й створить фундамент для майбутніх досліджень у галузі застосування нейронних мереж в освітніх технологіях.

Нейронні мережі освіти. В останні десятиліття нейронні мережі стали потужним інструментом у галузі штучного інтелекту, виявляючи вражаючі результати у різних завданнях. Їхня здатність до навчання на великих обсягах даних та адаптації до мінливості мови робить їх привабливим засобом для розробки новаторських систем тестування.

Мета та завдання дослідження

Мета даної магістерської дисертації полягає у розробці та аналізі системи тестів з англійської мови, заснованої на нейронних мережах, з метою підвищення точності та об'єктивності оцінки мовних компетенцій.

Об'єктом дослідження є автогенерація запитань, правильних та неправильних відповідей на основі тексту англійської мови.

Предметом дослідження є методи та алгоритми автогенерації запитань, правильних та неправильних відповідей на основі тексту англійської мови.

Для досягнення цієї мети поставлено такі завдання:

- з'ясувати основні вимоги і бажані кінцеві результати;
- вивчити наявність існуючих нейронних моделей обробки природної мови, які можуть допомогти досягти поставлених цілей;

- вибрати відповідні моделі і методи їх навчання;
- прийняти обґрунтоване рішення щодо інструментів для навчання моделей;
- створити робочу модель для тестів і варіантів відповідей.

Наукова новизна. Автоматичне формування тестів: Система може використовувати нейронні мережі для створення тестових завдань, ґрунтуючись на заданих параметрах, таких як рівень складності або необхідний контент. Це дозволяє швидко формувати тести з великою кількістю питань і різних варіантів.

Адаптивність та персоналізація: нейронна мережа може аналізувати відповіді студента на попередні запитання та адаптувати наступні тести на основі цих даних відповідно до рівня знань та здібностей кожного конкретного учня. Це дозволяє оптимально використовувати час студента і забезпечити більш точне вимірювання їх успішності.

Автоматична оцінка та зворотній зв'язок: Система може використовувати нейронні мережі для автоматичної оцінки відповідей студента та надання миттєвого зворотного зв'язку. Це підвищує ефективність та швидкість оцінювання, а також допомагає студентам швидко зрозуміти свої помилки та покращити свої знання.

Передбачення успішності: нейронні мережі можуть використовуватися для аналізу даних про минулі результати студента і передбачення його майбутньої успішності. Це дозволяє викладачам виявляти слабкі місця студента і вживати відповідних заходів для підвищення їх успіху.

Значимість дослідження. Дане дослідження зробить внесок в область освіти та тестування, надаючи новий погляд використання нейронних мереж в оцінці мовних навичок. Результати роботи можуть бути потрібні освітніми установами, а також компаніями, які займаються мовним навчанням та оцінкою кваліфікацій персоналу.

Слід зазначити, що ця магістерська дисертація не лише вирішує конкретну проблему оцінки англійської мови, а й відкриває шлях для майбутніх досліджень у галузі застосування нейронних мереж в освітніх технологіях.

Матеріали дослідження з тематики кваліфікаційної роботи доповідалися на V Всеукраїнській науково-технічній конференції: «Комп'ютерні технології: інновації, проблеми, рішення» 01-02 грудня 2022 р. [1], а також на III Всеукраїнській науково-практичній інтернет-конференції здобувачів вищої освіти та молодих вчених: «Сучасні комп'ютерні та інформаційні системи і технології» 12-19 грудня 2022 р. [2]

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

В рамках випускної кваліфікаційної роботи необхідно знайти найбільш придатний для створення тексту тестових завдань вид нейронних мереж [3]. До найбільш популярних для даної задачі видів нейронних мереж належать:

- рекурентні нейронні мережі (RNN);
- згорткові нейронні мережі (CNN);
- трансформери (Transformer).

Для порівняння зазначених видів нейронних мереж можна використовувати такі критерії, як застосування до завдання генерації коротких тестових пропозицій та вимоги до обчислювальних ресурсів.

1.1. Генерація тексту за допомогою RNN

Рекурентні нейронні мережі (RNN) [4] - це тип нейронних мереж, які містять циклічні зв'язки та можуть зберігати інформацію з попередньої ітерації у наступній ітерації. Тому рекурентні нейронні мережі ефективні для обробки послідовностей.

Структура рекурентної нейронної мережі представлена на рисунку 1.1.

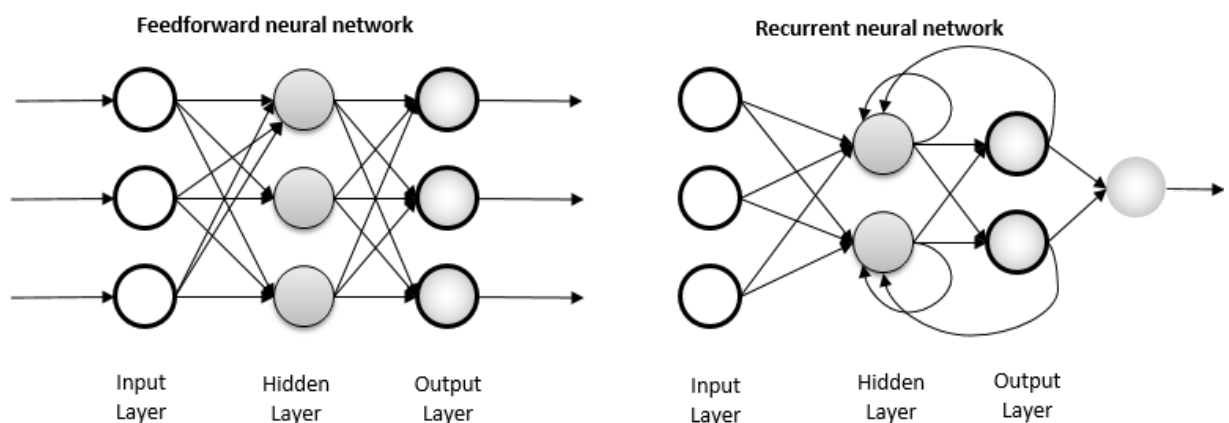


Рисунок 1.1 – Структура рекурентної нейронної мережі

У зв'язку з цим у них було внесено різні удосконалення. Найчастіше використовуваними нині є Gated Recurrent Unit (GRU) і Long Short-Term Memory (LSTM) [5].

Моделі LSTM були запропоновані для аналізу мови у 1997 році. Її перевага перед звичайними RNN полягає в тому, що вона менш чутлива до тимчасових розривів. Недоліком цієї моделі є її складність та додаткові обчислювальні витрати.

Модель GRU була запропонована у 2014 році як спрощена версія LSTM. Ця модель містить менше параметрів, ніж LSTM, але показує близькі результати деяких завдань. Найменша обчислювальна складність спрощує створення великих мереж і загалом прискорює навчання та роботу мережі.

Обидві моделі можуть бути використані на вирішення поставлених завдань; моделі LSTM, як правило, мають потужніші механізми пам'яті. Однак була обрана спрощена модель GRU, оскільки для створення однієї пропозиції довгострокові залежності менш важливі, ніж для створення тексту.

Існує кілька способів перетворення тексту в послідовності елементів для нейронних рекурентних мереж. Найбільш поширені з них – на рівні слів та символів.

Генерація тексту лише на рівні слів означає, що вихід нейронної мережі - це порядковий номер слова в словнику. Хоча цей метод можна використовувати для створення тексту, він має ряд недоліків. Моделі із невеликим обсягом словника зазвичай видають монотонний текст.

Для кодування десятків або сотень тисяч слів потрібний великий обсяг пам'яті. Крім того, моделі з великим обсягом лексики вимагають значно більшої кількості навчальних даних, щоб переконатися, що модель адекватно представляє використання кожного слова в лексиконі. Крім того, робота на рівні слів не застосовується для генерації послідовностей, що не складаються зі слів (наприклад, телефонних номерів, веб-адрес).

Завдання на рівні символів генерують набагато довші послідовності, які потребують складніших структур нейронних мереж. Тому моделі на рівні символів погано справляються з деякими завданнями. Однак їх структура робить їх більш цікавими як генеративні моделі, оскільки вони можуть генерувати послідовності, не включені в навчальні дані. Крім того, робота на рівні символів

дозволяє значно скоротити обсяг пам'яті, необхідної для роботи моделі, що знижує вимоги до апаратного забезпечення. З цих причин для вирішення поставленого завдання було обрано роботу на рівні символів [6].

Існує кілька способів перетворення символів у вхідні значення для нейронних рекурентних мереж. Найпростіший - присвоїти символу порядковий номер і використовувати цей номер як вхідне значення. Недоліком цього методу є те, що порядок цифр не стосується літер під цифрою, але може ввести модель в оману (наприклад, літери з вищими порядковими номерами можуть бути позначені як важливіші). Цей метод найкраще підходить для нумерації слів, коли між ними є зв'язок (наприклад, "маленький", "середній", "великий"). Інший варіант представлення символів - одноточкове кодування, при якому з кожним символом зв'язується вектор довжини, яка дорівнює кількості символів у словнику моделі. До кожного символу один елемент вектора, відповідний його номеру, дорівнює одиниці, інші ж рівні нулю.

Нуль. Цей метод вільний від попереднього недоліку, пов'язаного із символічними нерівностями, але вимагає більше пам'яті для зберігання розріджених векторів довжини, що дорівнює довжині словника моделі. Цей метод легко використовувати, коли довжина словника моделі мала, але зі збільшенням кількості елементів у словнику потреба у пам'яті швидко зростає.

Існує також варіант вбудовування, при якому символи подаються у вигляді векторів, що містять певну кількість дійсних чисел. За такого підходу кожне значення відбиває якусь особливість символу. Такий спосіб подання часто використовується і для слів, коли значення слова представляється у вигляді вектора дійсних чисел, кожне значення якого асоціюється з якоюсь властивістю слова. В даний час розробляються спеціалізовані методи для отримання значень векторів слів, такі як Word2Vec і GloVe [8], але, крім них, значення таких векторів можуть бути отримані за допомогою глибокого навчання на певних наборах даних, наприклад, Embedding Projector, зображений на рисунку 1.2. Існують програми, що дозволяють переглядати проєкції векторів різних слів. Як

і у випадку з One-Hot кодуванням, зі збільшенням розміру словника моделі збільшуються вимоги до пам'яті [9].

Модель, реалізована у цьому дослідженні, працює із символами, а чи не зі словами, але можна використовувати і векторні уявлення символів. У цьому випадку словник моделі складається з 95 символів (кількість символів, які можна надрукувати в кодуванні ASCII), тому вимоги до пам'яті нижче. У зв'язку з цим реалізації моделі було обрано варіант вбудовуваного уявлення символів.

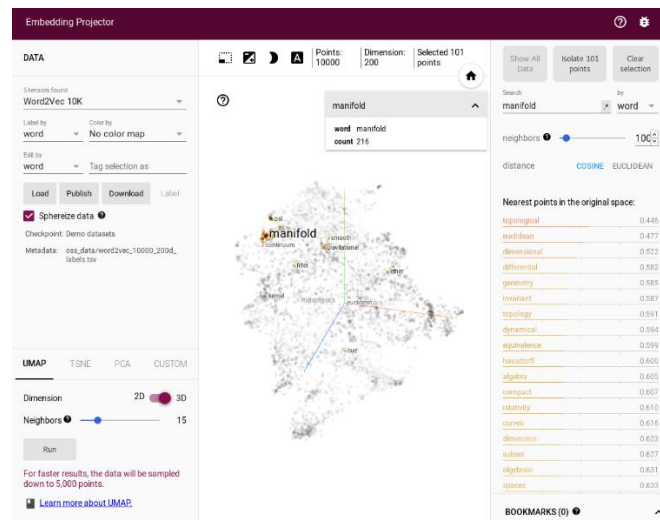


Рисунок 1.2 – Embedding Projector

Генерація послідовності за допомогою RNN також називається вибіркою: вихідне значення RNN є розподілом ймовірностей наступного елемента (в даному випадку символу), який може бути обраний різними способами. Після генерації чергового елемента послідовності ймовірність наступного елемента обчислюється на основі всіх попередніх елементів, включаючи ті, що були згенеровані на останньому кроці. Послідовності будь-якої довжини можуть бути згенеровані цим способом. Рядок, згенерований цим методом, може бути як окремою пропозицією, так і зв'язковим текстом; Схема генерації рядка "Think" за допомогою RNNs показана на рисунку 1.3.

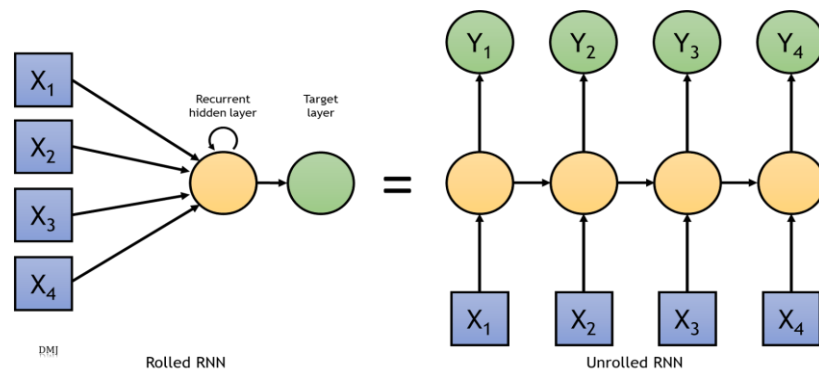


Рисунок 1.3 – Генерація послідовності за допомогою RNN

Існує кілька методів створення послідовностей за допомогою RNN.

RNN використовуються для створення послідовностей. Один із них - метод жадібного пошуку. У цьому методі кожному етапі генерації вибирається елемент із найбільшою ймовірністю. Недоліком цього методу є те, що модель завжди видає ту саму послідовність при одних і тих же вхідних даних, що робить його непридатним для отримання безлічі зразків речень. Крім того, при використанні цього методу немає можливості оцінити якість послідовностей до завершення генерації. Схема алгоритму жадібного пошуку для генерації слів показано на рисунку 1.4.

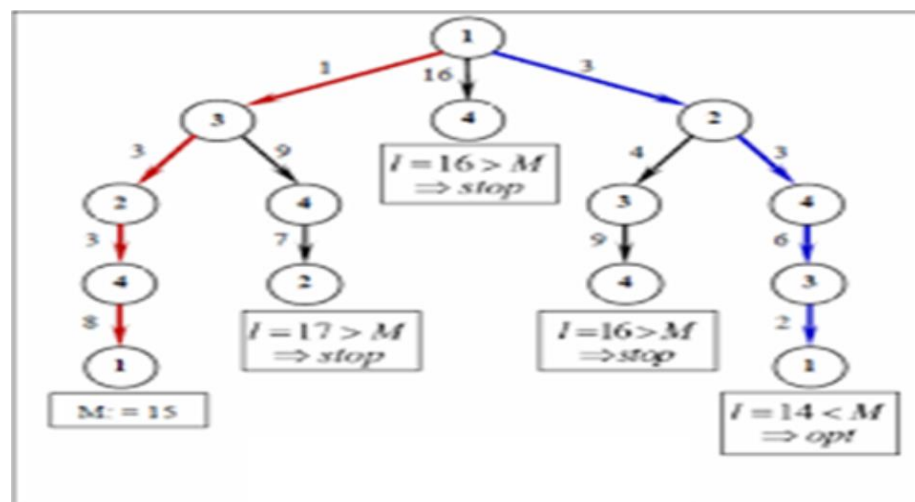


Рисунок 1.4 – Алгоритм жадібного пошуку

Іншим широко використовуваним методом є метод променевого пошуку. У цьому методі замість зберігання лише одного найкращого варіанта кожному етапі виробництва зберігається кілька варіантів. Таким чином, модель може генерувати кілька послідовностей та вибрати найкращу з них наприкінці роботи. Цей підхід широко використовується в таких завданнях, як машинний

переклад, але не дуже добре застосовується до проблеми генерації речень. Схема алгоритму променевого пошуку показано на рисунку 1.5.

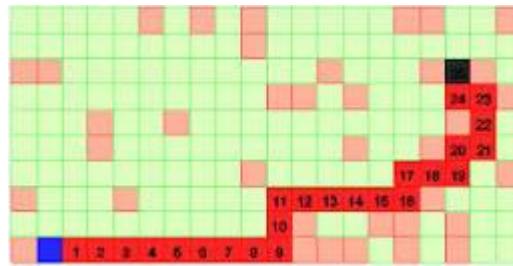


Рисунок 1.5 – Алгоритм променевого пошуку

Метод випадкової вибірки також успішно використовується у завданнях обробки текстів. Цей метод генерує послідовність на основі вибору наступного елемента в послідовності через розподіл ймовірності з усіх можливих елементів. Метод має гіперпараметр, званий температурою, який використовується для керування ймовірністю вибору наступного елемента. Цей метод найбільш зручний для вирішення поставленої задачі, оскільки він дозволяє генерувати велику послідовність та вводить додатковий гіперпараметр для контролю результату. Оскільки зміна температури не вимагає перенавчання моделі, цей параметр є комерційно доступним.

1.2. Генерація тексту за допомогою CNN

Однією із проблем рекурентних нейронних мереж є необхідність послідовної обробки вхідних даних. Через відсутність паралельної обробки необхідний час може значно збільшитися, особливо під час роботи з великими обсягами тексту.

Одним із рішень цієї проблеми є генерація тексту за допомогою надточної нейронної мережі (CNN) [10]. Головна перевага цього підходу – простота паралельної обробки. CNN також можуть виявляти локальні залежності у вхідних даних, а порядок відстаней між елементами вхідних даних є логарифмічним, на відміну від RNN, які є лінійними. У цьому відношенні надточні нейронні мережі краще справляються з далекими залежностями тексту. Однак вхідні дані надточних нейронних мереж мають кінцеву кінцеву довжину.

Тому такі мережі не можуть виявляти залежності між елементами, розташованими на відстані, що перевищують довжину оброблюваної ними послідовності вхідних даних.

Прикладами відомих архітектур нейронних мереж, що використовуються для генерації різних послідовностей, є Bytenet, призначена для машинного перекладу, та Wavenet, призначена для генерації аудіопослідовностей. На прикладі Wavenet обробка вхідних даних такою моделлю показано рисунку 1.6.

Як видно з рисунка, Wavenet є надточною нейронною мережею, і, завдяки її структурі, розмір вхідних даних експоненційно збільшується з глибиною нейронної мережі; за словами авторів архітектури, Wavenet, незважаючи на високі вимоги до обчислювальних ресурсів, цей підхід показав хороші результати у задачі генерації складних та реалістичних частотних звуків, таких як

За словами авторів, підхід показав хороші результати у завданнях генерації складних реалістичних звуків із частотами вищими за 16 кГц.

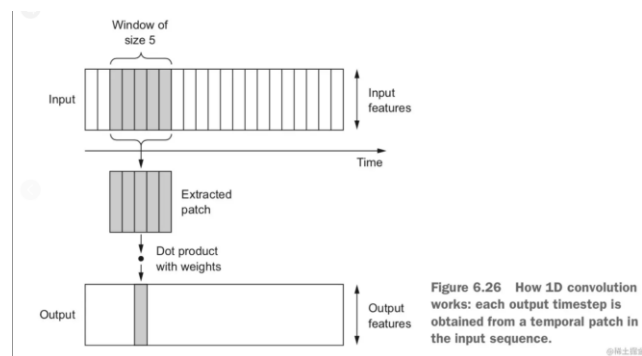


Рисунок 1.6 – Обробка послідовностей за допомогою CNN

Крім вищезгаданих переваг, підходи до генерації послідовностей на основі нейронних конволюційних мереж мають і деякі недоліки. Окрім високих вимог до обчислювальних ресурсів, конволюційні моделі обробки даних не завжди можуть розпізнати складні залежності між вхідними даними, які часто зустрічаються у завданнях обробки тексту.

1.3. Генерація тексту за допомогою Transformer

В даний час однією з найпопулярніших архітектур для обробки мови є архітектура Transformer. Вона заснована на поєднанні підходів, заснованих на CNN та увазі [11]. Підхід з урахуванням уваги ґрунтується на присвоєнні кожному елементу вхідних даних значення, що відбиває важливість цього елемента цьому кроці. Таким чином, на кожному кроці обробки послідовності модель певним чином надає "увагу" елементам вхідних даних. У міру обробки тексту модель може дізнатися, які елементи вхідних даних заслуговують на особливу увагу, що підвищує ефективність роботи моделі при вирішенні багатьох завдань. Одне із завдань, що найчастіше вирішуються за допомогою цього підходу, - машинний переклад.

Перетворювачі використовують самоувагу для прискорення роботи. Архітектура перетворювача складається з деякої кількості кодерів та декодерів зі схожою структурою. Кодер складається з механізму самонавіювання і нейронної мережі з прямолінійним рухом, а декодер включає ці два шари і механізм уваги кодера-декодера [12].

У разі роботи зі словами механізм самоуваги полягає у використанні додаткових векторів для опису зв'язків між вхідними даними. Цими векторами є вектори запитів, ключові вектори та вектори значень. Ці вектори створюються шляхом множення вхідних векторів, одержаних в результаті роботи механізму вбудовування, на три відповідні матриці, які змінюються внаслідок процесу навчання.

Наступним кроком у процесі саморозуміння є обчислення значень (балів) слів шляхом множення вектора запиту аналізованого слова на ключові вектори інших слів. На наступному етапі вектор значень кожного слова множиться на значення, отримане в результаті softmax операції. Таким чином, значущі ваги можуть бути присвоєні словам, які важливі для визначення значення, а менші ваги - словам, які не є корисними на даному етапі [13].

На останньому етапі шляхом підсумовування отриманих творів виходить остаточний результат саморозпізнавання процесу для даного слова. Процес показаний рисунку 1.7.

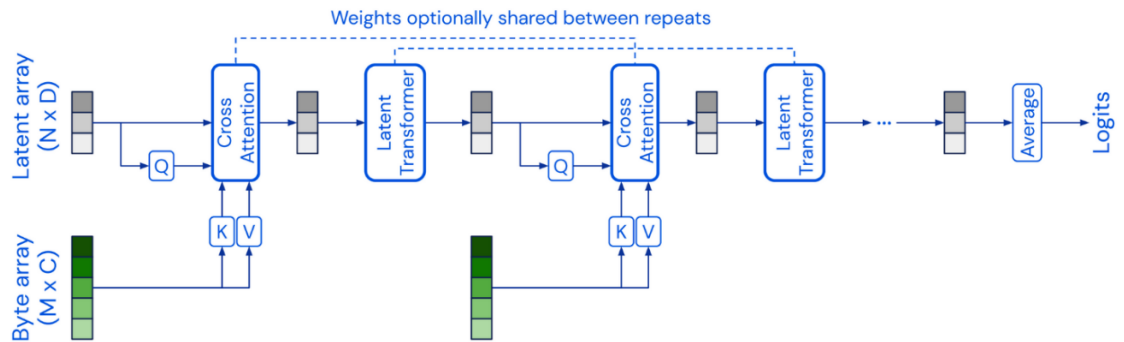


Рисунок 1.7 – Процес самоуваги

Це є основою продуктивності трансформатора. Існують інші механізми, які можуть поліпшити результати роботи трансформатора. До них відноситься механізм множинної уваги, коли створюється кілька наборів матриць запитів, ключів та значень. При такому підході кожен набір матриць може відповідати за кілька різних питань, які можуть бути задані в процесі обробки речення, як показано на рисунку 1.8. Іншим механізмом є кодування позицій, яке враховує порядок слів у вхідному реченні [14].

В даний час перетворювачі використовуються для вирішення різних завдань обробки тексту, таких як машинний переклад та генерація тексту. Відомим прикладом використання перетворювачів для генерації тексту є гра зі штучним інтелектом Dungeon. У цій грі гравці можуть писати різні історії, використовуючи автоматичну генерацію тексту разом із рукописним текстом. Перевага перетворювачів полягає в тому, що вони здатні обробляти зміст тексту більш точно, ніж інші архітектури, проте моделі перетворювачів, що використовуються в даний час, дуже ресурсомісткі, причому не тільки для навчання, але і для запуску навчених моделей.

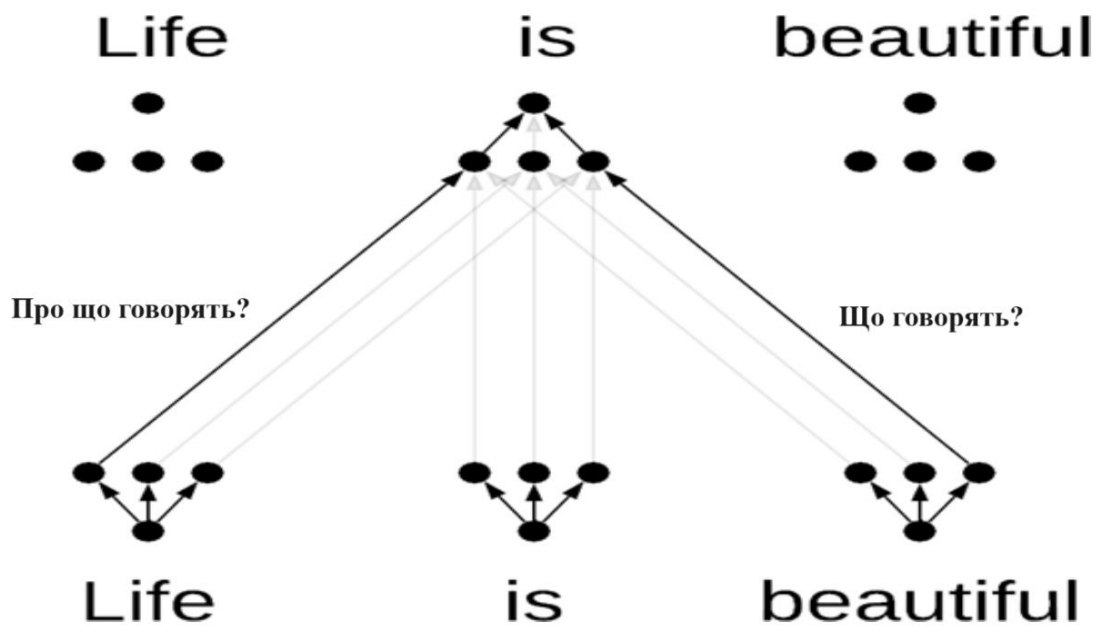


Рисунок 1.8 – Множинна увага

У ході вивчення літератури були прийняті до уваги типи нейронних мереж, що найчастіше використовуються для завдання генерації тексту. Після порівняння за різними критеріями було обрано найбільш підходящу модель для цієї задачі. Рекурентні нейронні мережі мають найбільш м'які вимоги до ресурсів і здатні генерувати точні пропозиції, які можуть бути використані для автоматичного складання тестового завдання з англійської мови.

1.4. Огляд BERT (Bidirectional Encoder Representations from Transformers)

BERT (Bidirectional Encoder Representations from Transformers) – це модель глибокого навчання, яка була розроблена та представлена у 2018 році дослідниками компанії Google. Основною метою BERT було створення моделі, здатної розуміти природну мову більш високому рівні, враховуючи контекст і семантику слів [15].

BERT заснований на архітектурі трансформерів, яка стала одним із ключових проривів у галузі обробки природної мови. Трансформери є модель, засновану на механізмі "уваги" (attention mechanism), що дозволяє моделі вчитися захоплювати залежності між словами в реченні.

BERT відрізняється від попередніх моделей обробки природної мови тим, що вона використовує контекстуалізовані ембедінги. Замість того, щоб представляти кожне слово у реченні незалежно, BERT враховує його контекст, що дозволяє краще вловлювати зміст слів та їх взаємозв'язок у реченні. Іншими словами, модель BERT враховує як контекст перед словом, так і контекст після нього при обробці.

Модель BERT навчається на великих нерозмічених текстових корпусах, таких як Вікіпедія, і в процесі навчання передбачає наступне слово в реченні. Це дозволяє моделі навчитися вловлювати сенс різних слів і виразів та будувати якісніші ембедінги речень.

BERT також пройшов подальшу попередню обробку даних, таку як токенізація та сегментація пропозицій, щоб подати їх у зручній формі для моделі. Експерименти показали, що BERT досягає видатних результатів на різних завданнях обробки природної мови, включаючи питання-відповіді системи, розпізнавання іменованих сутностей та інші.

В цілому, BERT є важливим кроком у галузі обробки природної мови і дозволяє моделі краще розуміти та обробляти складні структури та зміст речень.

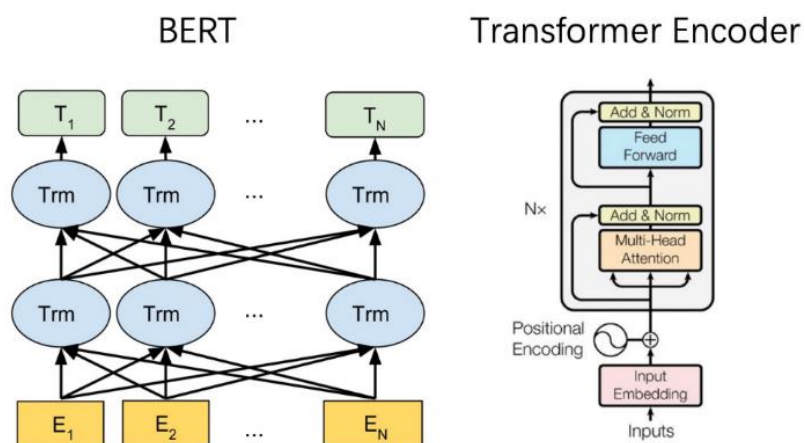


Рисунок 1.9 - Bidirectional Encoder Representations from Transformers

1.5. Порівняння готових програмних продуктів

У цьому підрозділі ми розглянемо 3 готових програмних продукту для генерації тестових завдань з англійської мови.

Таблиця 1.1 – Порівняльна характеристика

Характеристика	PrepAI [16]	Quillionz[17]	Questgen[18]
Тип продукту	Генератор контенту для навчання AI	Інструмент для створення питань і контенту	Інструмент для генерації питань і тестів
Мета використання	Підготовка даних для навчання моделей AI	Створення питань для навчання та тестування	Генерація завдань і тестів для освітніх цілей
Інтерфейс	Web-додаток з графічним інтерфейсом	Web-додаток з простим інтерфейсом	Командний рядок та API
Мовна підтримка	Обмежена, залежить від моделі мови AI	Багато мов підтримується	Широка мовна підтримка
Генерація контенту	Використовує потужні мовні моделі AI	Заснована на шаблонах і ключових словах	Заснована на правилах і шаблонах
Навчання моделі	Використовує заздалегідь навчені моделі	Не вимагає навчання, працює на основі даних користувача	Може бути дообучена на призначених для користувача даних
Гнучкість генерації	Висока гнучкість, але залежить від якості навчання моделі	Обмежена заздалегідь заданими шаблонами	Гнучкість залежить від налаштувань Користувача
Рівень складності	Складний для новачків, потрібне розуміння роботи з AI	Легкий у використанні, Підходить для широкого кола користувачів	Середній рівень складності, вимагає розуміння шаблонів і правил
Ціна	Залежить від тарифного плану, надаються безкоштовні опції	Безкоштовна версія і платні тарифи	Безкоштовна і платні версії, ціна залежить від використання API і додаткових функцій
Висновок	PrepAI підходить для тих, хто шукає потужний генератор вмісту за допомогою AI, але вимагає розуміння роботи з цими технологіями.	Quillionz підходить для швидкого створення питань і контенту без необхідності розуміння складних алгоритмів.	Questgen забезпечує більш гнучке налаштування генерації, підходить для тих, хто хоче більш тонкого налаштування і може захотіти деякого навчання для оптимальних результатів.

Кожен з продуктів має свої сильні і слабкі сторони, і вибір між ними залежить від конкретних потреб користувача і рівня його експертизи в області генерації контенту і використання штучного інтелекту.

І, як підсумок, розглянемо плюси і мінуси даних продуктів:

- **PrepAI**

Плюс:

Потужні мовні моделі: PrepAI використовує передові мовні моделі, що забезпечує високу гнучкість і якість генерації контенту.

Навчання моделей: може бути дообучен на призначених для користувача даних, що дозволяє адаптувати його до конкретних потреб.

Мінус:

Складність використання: вимагає розуміння роботи зі штучним інтелектом, що робить його менш доступним для новачків.

Залежність від якості навчання: якість генерації може залежати від якості попереднього навчання моделі.

- **Quillionz**

Плюс:

Простота використання: має простий інтерфейс і заснований на шаблонах, що робить його доступним для широкого кола користувачів.

Безкоштовна версія: надає безкоштовну версію з базовими функціональностями.

Мінус:

Обмежена гнучкість: заснований на шаблонах і ключових словах, що обмежує гнучкість у створенні різноманітного контенту.

Обмежені можливості навчання: не надає можливості навчання моделі під конкретні потреби користувача.

- **Questgen**

Плюс:

Широка мовна підтримка: підтримує багато мов, що робить його універсальним для різних мовних спільнот.

Гнучкість налаштування: дозволяє більш тонке регулювання генерації, що підходить для тих, хто шукає більш індивідуалізований контент.

Можливість донавчання: може бути дообучен на призначених для користувача даних для оптимізації результатів.

Мінус:

Середній рівень складності: вимагає розуміння шаблонів і правил, що може зробити його менш привабливим для новачків.

Командний рядок та API: інтерфейс базується на командному рядку та API, що може створити додаткові бар'єри для деяких користувачів.

Вибір між цими продуктами залежить від конкретних потреб користувача: PrepAI підходить для досвідчених користувачів, Quillionz - для тих, хто шукає простоту, а Questgen - для тих, хто цінує гнучкість і готовий вкласти зусилля в додаткове навчання і настройку.

Висновки з 1-го розділу.

У розділі 1 "АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ" були розглянуті різні методи генерації тексту, включаючи використання рекурентних нейронних мереж (RNN), згорткових нейронних мереж (CNN) та трансформерів.

У підрозділі 1.1 була проведена аналітична робота з опису та аналізу генерації тексту з використанням RNN. Були розглянуті основні принципи роботи RNN, їх переваги та недоліки, а також застосування цієї моделі для створення тексту.

Підрозділ 1.2 присвячено генерації тексту з використанням CNN. Тут був проведений аналіз роботи згорткових нейронних мереж, їх можливості у сфері генерації тексту, а також оцінка їхньої застосовності та ефективності.

В підрозділі 1.3 проведено аналіз генерації тексту з використанням трансформерів. Були розглянуті основні принципи роботи трансформерів, їх застосування для генерації тексту та порівняння з іншими моделями.

Підрозділ 1.4 присвячено моделі глибокого навчання BERT. Була розглянута архітектура та в цілому тип моделі та її історія.

В останньому підрозділі 1.5 було необхідно розглянути і порівняти вже наявні подібні готові продукти на ринку, що б мати повне розуміння до чого, в результаті потрібно прагнути що б уникнути помилок попередників.

Метою цього розділу було провести огляд предметної області та розглянути різні методи генерації тексту. Ми вивчили основні принципи роботи RNN, CNN і трансформерів, їх переваги та недоліки, а також показали їхню застосовність для генерації тексту.

Висновки з цього розділу будуть корисні при виборі оптимального методу генерації тексту для подальших досліджень та розробки нових моделей.

РОЗДІЛ 2. ОПИС НАБОРУ ДАНИХ ДЛЯ ГЕНЕРАЦІЇ ТЕСТОВИХ ЗАВДАНЬ З АНГЛІЙСЬКОЇ МОВИ ДЛЯ НЕЙРОННОЇ МЕРЕЖІ

2.1. Опис набору даних

Набір даних, що використовується в даній роботі, є набір англійських пропозицій з проекту Tatoeba [19], колекції пропозицій та перекладів з відкритим вихідним кодом. Набір містить близько 1,5 мільйона англійських пропозицій різної тематики та складності. Всі пропозиції можна завантажити з офіційного сайту проекту як файл `eng_sentences.tsv`. У таблиці 2.1 наведено деякі вибрані пропозиції з цього файлу.

Таблиця 2.1 – Пропозиції із файлу `eng_sentences.tsv`

Ідентифікатор	Код мови	Текст речення
1277	eng	I have to go to sleep.
1530	eng	Allen is a poet.
1771	eng	The wind calmed down.

На додаток до цього файлу використовувався файл `tags.csv`, що містить теги речень, додані користувачем. Теги вказують на різні характеристики речення, такі як тема або джерело речення, час дієслова в реченні (наприклад, час однини або минулий час). Тільки в невеликій кількості речень є теги, що вказують на час дієслова. Наприклад:

Файл `tags.csv` містить унікальний ідентифікатор речення та текст тега для кожного тега. У таблиці 2.2 наведено деякі з тегів у файлі `tags.csv`.

Таблиця 2.2 - теги у файлі `tags.csv`

Ідентифікатор	Мітка
1277	present simple
1277	less than 8 words
1771	past simple

Таким чином, набір даних Tatoeba містить велику кількість речень, але лише невелика частина речень має теги, що визначають граматичний час дієслова в реченні [20].

2.2. Застосування самонавчальної моделі нейронної мережі для генерації тестових завдань

Щоб навчити генератор речень, було вирішено класифікувати речення по граматичній формі дієслова, що використовується в реченні, наприклад, сьогодення або просте сьогодення. Таке обмеження обмежує граматичні форми використовуваних дієслів та підвищує якість генератора речень [21]. На першому етапі попередньої обробки даних було створено файл для навчання класифікатора, який можна було використовувати для вибору найбільш підходящих пропозицій позначення заданого часу дієслова. З даних, які у файлах `eng_sentences.tsv` і `tags.csv`, було відібрано речення з тегами, що вказують на граматичний час дієслова [22]. В результаті були отримані пропозиції, позначені як "present simple", "present continuous", "past simple" та "future simple",

Пропозиції, позначені як "просте минуле" і "просте майбутнє", а також пропозиції, позначені як "повне сьогодення" та "продовжене сьогодення", були відібрані та згруповані в кластери, позначені як "інші", через їх невелику кількість [23]. Оскільки в даний час не існує способу отримати найкращу структуру нейронної мережі для конкретного завдання для навчання архітектури, описаної в цьому підрозділі, використовувалися лише пропозиції, надані проектом Татоеба [24]. Текстові файли, використані як вхідні дані для моделі, склалися з пропозицій, класифікованих як "прості на даний момент", як описано в попередньому розділі [25]. Для реалізації нейронної мережі, описаної в цьому розділі, було використано бібліотеку Keras [26, 27].

Першим етапом роботи буде завантаження бібліотек

Імпорт `pandas` як `pd`: Імпортує бібліотеку `pandas` і надає їй псевдонім "pd" для зручності. `Pandas` - потужна бібліотека для маніпулювання даними та аналізу для `Python`.

`import numpy` як `np`: Імпортує бібліотеку `NumPy` та надає їй псевдонім "np". `NumPy` – це бібліотека для числових операцій на `Python`, що забезпечує підтримку великих багатовимірних масивів та матриць, а також математичних функцій.

`import string`: імпортує модуль `string`, який надає різні функції та константи, пов'язані з рядками.

`import operator`: імпортує модуль операційної системи, який надає спосіб використання функцій, що залежать від операційної системи, таких як читання або запис у файлову систему.

`import nltk`: Імпортує Natural Language Toolkit (`nltk`), бібліотеку для завдань обробки природної мови, таких як токенізація, стеммінг та позначка частин мови.

`import re`: Імпортує модуль регулярних виразів, який дозволяє використовувати регулярні вирази для зіставлення зі зразком у рядках.

`import random`: Імпортує модуль `random`, який надає функції для створення псевдовипадкових чисел.

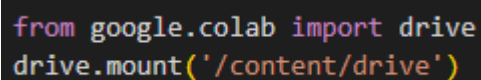
`import io`: Імпортує модуль вводу-виводу, який надає основні засоби Python для роботи з різними типами вводу-виводу (`input/output`), такими як читання з потоку або запис у потік.

`import sys`: імпортує модуль `sys`, який надає доступ до деяких змінних, що використовуються або підтримуються інтерпретатором, та функцій, які взаємодіють з інтерпретатором.

`import warnings`: імпортує модуль `warnings`, який надає функції для керування попередженнями, які видають інтерпретатор Python.

Попередження. `filterwarnings` ("ігнорувати"): Встановлює фільтр для ігнорування всіх попереджень. Цей рядок пригнічує попереджувальні повідомлення, які в іншому випадку можуть відображатися під час виконання коду. Приклади пропозицій, що згенеровані з їх допомогою, наведені в таблиці.

Далі підключаємо Google Disk, як показано на рисунку 2.1.



```
from google.colab import drive
drive.mount('/content/drive')
```

Рисунок 2.1 – Підключення Google Disk

Цей код використовується в середовищі Google Colab для встановлення диска Google Drive. Після його виконання з'явиться запит на автентифікацію, і після успішного входу до вашого облікового запису Google Drive, диск буде

примонтований в папку `"/content/drive"` на віртуальній машині Google Colab. Далі ви зможете звертатися до файлів на Google Drive зі свого ноутбука Google Colab.

Код з `google.colab import drive` імпортує функцію `drive` з бібліотеки `google.colab`, а `drive.mount('/content/drive')` викликає функцію `mount` з аргументом `"/content/drive"`, вказуючи шлях для монтування Google Drive.

Слідом запускаємо функцію читання речень, вона виконує кілька кроків:

```
pd.read_csv("/content/drive/MyDrive/eng_sentences.tsv", sep = '\t', names = ['id', 'lang', 'text'], encoding='utf-8')
```

 - цей фрагмент завантажує дані з файлу `TSV eng_sentences.tsv` та вказує, що поля розділені табуляціями (`sep = '\t'`).

Ці дані завантажуються в об'єкт типу `DataFrame` з бібліотеки `pandas`, із трьома стовпцями, названими `"id"`, `"lang"` і `"text"`. Потім до стовпця `"text"` застосовується індексація (`['text']`), що означає, що в новому `DataFrame` буде міститися тільки стовпець `"text"`.

```
data = pd.DataFrame(data, columns = ['text'])
```

 - цей фрагмент створює новий `DataFrame` з ім'ям `"data"`, використовуючи лише стовпець `"text"` з попереднього `DataFrame`.

Таким чином, в результаті виконання цього коду ми отримуємо новий `DataFrame` під назвою `"data"`, що містить лише стовпець `"text"` з вихідних даних.

Задаємо регулярний вираз, який використовується для пошуку та фільтрації тексту. Розберемо його докладніше:

`r` на початку рядка вказує на те, що це рядок із використанням "сирих" (raw) рядкових літералів у Python.

`[^az,. !?-'&:;]` - це шаблон, який використовується для пошуку в тексті. Він означає будь-який символ, який не знаходиться в діапазоні від `"a"` до `"z"` або не є одним із символів: `"", ".", "'", "!", "?", "-", " ", ":", ";"`. Символ `^` спочатку вказує, що ми шукаємо всі символи, крім перерахованих.

Таким чином, при використанні цього регулярного виразу у функціях роботи з рядками, таких як `re.findall()` або `re.sub()`, будуть знайдені та/або видалені всі символи, що не відповідають зазначеному шаблону.

Виконуємо обробку тексту в стовпці `"text"` `DataFrame "data"` таким чином:

`data['text'].apply()` - застосовує функцію до кожного елемента стовпця "text" у DataFrame "data".

`lambda x: re.sub(r'^az.,!?\\r ]+',", x.lower())` - це анонімна функція, яка приймає рядок `x` як вхідні дані і виконує наступні дії:

`x.lower()` - наводить всі символи в рядку `x` до нижнього регістру.

`re.sub(r'^az.,!?\\r ]+',", ...)` - замінює всі символи, що не відповідають регулярному виразу `r'^az.,!?\\r ]+'`, у рядку `x` на порожній рядок.

Таким чином, цей код видаляє всі символи, що не є літерами (az), комами, точками, знаками оклику і знаками оклику, і знаками перенесення рядка, з тексту в стовпці "text", приводячи всі літери до нижнього регістру.

Цей код корисний для очищення текстових даних та підготовки їх до аналізу.

Наступним етапом буде результат виконання коду створення рядка, що містить всі текстові значення зі стовпця "text", об'єднані в один рядок з використанням пробілів як роздільники.

`data['text'].values.tolist()` - цей фрагмент перетворює стовпець "text" з DataFrame "data" в масив NumPy (`.values`), а потім у список Python (`.tolist()`), що містить усі текстові значення з стовпця "text".

`' '.join(...)` - потім ці значення поєднуються в один рядок, розділений пробілами, за допомогою методу `join`.

Створюємо словник `char2idx`, де кожному символу `u` з `vocab` зіставляється його індекс `i`. Індеси починаються з 0 та збільшуються на одиницю для кожного символу.

Потім код створює масив `idx2char`, який містить елементи з `vocab`.

Далі код створює масив `text_as_int`, який містить індекси символів із рядка `text`, зіставлені ним у словнику `char2idx`.

І, нарешті, код генерує список рядків, де кожен рядок представляє символ з `char2idx` та його відповідний індекс `i`. У разі перераховані перші 20 символів зі словника. Підсумок генерації показаний рисунку – 2.2.

```
[ ' = 0',  
  '! = 1',  
  ', = 2',  
  '. = 3',  
  '? = 4',  
  'a = 5',  
  'b = 6',  
  'c = 7',  
  'd = 8',  
  'e = 9',  
  'f = 10',  
  'g = 11',  
  'h = 12',  
  'i = 13',  
  'j = 14',  
  'k = 15',  
  'l = 16',  
  'm = 17',  
  'n = 18',  
  'o = 19' ]
```

Рисунок 2.2 – Генерація списку рядків

Обробляємо текст для створення датасету.

Змінна `seq_length` встановлює довжину послідовності символів, яка використовуватиметься для навчання моделі. В даному випадку значення дорівнює 100.

Змінна `examples_per_epoch` обчислює кількість прикладів, які можна одержати з тексту шляхом поділу на послідовності. Для цього текст ділиться на довжину послідовності `seq_length+1` і результат зберігається у змінній.

Потім створюється `char_dataset`, який є TensorFlow датасет, що містить індекси кожного символу в тексті. Для цього `text_as_int`, яка передбачається вектором цілих чисел, перетворюється на `tf.data.Dataset.from_tensor_slices`.

Нарешті, останній оператор використовується для повернення символів, що відповідають першим п'ятьом індексам у датасеті `char_dataset`. `idx2char` – це словник для перетворення індексів назад у символи.

Створюємо пакети (batches) із послідовностей символів. Вхідні дані представлені як `char_dataset`, а кожна послідовність має довжину `seq_length+1`. `drop_remainder=True` означає, що останній неповний пакет буде відкинутий, якщо він існує. Результатом коду є список рядків, де кожен рядок є поєднанням символів з послідовностей, перетворених з числового подання в символічне. Кожен символ утворюється за допомогою індексу `idx2char[item.numpy()]`. Потім символи об'єднуються в рядок за допомогою `join()`.

```
[ ] def generate_text(model, start_string, temp, gen_chars):
    input_eval = [char2idx[s] for s in start_string]
    input_eval = tf.expand_dims(input_eval, 0)
    text_generated = []
    model.reset_states()
    for i in range(gen_chars):
        predictions = model(input_eval)
        predictions = tf.squeeze(predictions, 0)
        predictions = predictions / temp
        predicted_id = tf.random.categorical(predictions, num_samples=1)[-1,0].numpy()
        input_eval = tf.expand_dims([predicted_id], 0)
        text_generated.append(idx2char[predicted_id])
    return (start_string + ''.join(text_generated))
```

Рисунок 2.3 – Функція генерації

Далі після низки нескладних процедур: розподілу датасета на моделі, завдання її параметрів і структури, а також функцій звільнення пам'яті та кастомізації функцій втрати та компіляції моделі, навчання 10 епох, переходимо до запуску функцій безпосередньої генерації тексту:

Аргументи функції:

`model`: модель, яка використовуватиметься для генерації тексту.

`start_string`: рядок, з якого починається генерація тексту.

`temp`: параметр, який визначає "різноманітність" згенерованого тексту (що вище значення, тим випадковішим буде текст).

`gen_chars`: кількість символів, які потрібно згенерувати.

Опис алгоритму:

Створює список `input_eval`, в якому зберігаються числові уявлення символів із `start_string` (перетворення символів у числа відбувається за допомогою словників `char2idx`).

Розширює розмір `input_eval`, щоб його можна було використовувати в моделі.

Ініціалізує порожній список `text_generated`, який міститиме згенерований текст.

Скидає стан моделі.

Запускає цикл за кількістю символів, які потрібно згенерувати.

Отримує передбачення моделі для поточного `input_eval`.

Стискає розмірність передбачень та застосовує функцію `softmax` із параметром `temp` для отримання ймовірнісного розподілу.

Випадково вибирає індекс символу, використовуючи передбачені ймовірності.

Розширює вибраний індекс та надає його `input_eval`.

Додає відповідний символ `text_generated`.

Повертає вихідний рядок плюс згенерований текст, об'єднаний в один рядок.

```
[ ] embedding_dim = 256
    rnn_units_2 = 128

def loss(labels, logits):
    return tf.keras.losses.sparse_categorical_crossentropy(labels, logits, from_logits=True)

model1 = tf.keras.Sequential([
    Embedding(31, embedding_dim, batch_input_shape=[1, None]),
    GRU(rnn_units_2, return_sequences=True, stateful=True, recurrent_initializer='glorot_uniform'),
    Dropout(0.2),
    GRU(rnn_units_2, return_sequences=True, stateful=True, recurrent_initializer='glorot_uniform'),
    Dropout(0.2),
    Dense(31)
])
```

Рисунок 2.4 - модель нейронної мережі для обробки послідовності
вхідних даних

І, нарешті, приходимо до результатів генерації текстів, вказаних у таблиці
2.3.

Таблиця 2.3 – Результати генерації речень за допомогою нейронної мережі, створеної самотійно

Температура	Пропозиції
generate_text(model1, prom, 2.0, 200)	alice said i mople!ootbsh obe.hed into bbchesludnitezy. yuehompert timeshance hapbed. wh! you nothnownows sleg and yank! next tohlax impheaely? wuqpuctangd theh. hher! tldmedtlea?s myself.. our . htgr is.hy dlsmar
generate_text(model1, prom, 1.0, 200)	alice said more than should i could hope find the ments. ii hospehabe of this time xiston and here. said everything i wouldnt be late for hard to be like tom? where have tom has been for boston to. pan cuttors
generate_text(model1, prom, 0.5, 200)	alice said that it was a little of the car in boston. tom andming the boys and have to be happy to the park and that he was serious than that. tom didnt want to be a lot of boston anything to be a bit a boy and
generate_text(model1, prom, 0.2, 200)	alice said that he was the only one when the problem is the street. tom has a state of the street. tom was a book to boston in boston. tom is a few months are the only one who have a states to boston to boston t
generate_text(model1, prom, 0.1, 200)	alice said that tom is a book. tom is a book and see the street. tom is a states and seems to be a book. tom has book and see the park of the street. tom is a states and started to be a book. tom is a states a

Як видно з результатів, створена модель здатна видавати рядки, близькі до англійських пропозицій, але не всі рядки, що видаються, граматично правильні. Багато пропозицій некоректні і не відповідають формі теперішнього часу. Це може свідчити про те, що пропозиції з навчальної бази даних містять інші форми дієслів, а також недостатність даних для її навчання [28].

Легко помітити вплив температури на результати. За низької температури всі мережі видають однакові та неправильні пропозиції. З підвищенням температури пропозиції стають різноманітнішими, але частіше видають неправильні пропозиції. При найвищих значеннях температури іноді з'являються англійські слова, що не існують, що також впливає на якість пропозицій.

Таким чином, в ході експериментів з розмірністю нейромережевої моделі було встановлено, що збільшення складності структури нейронної мережі незначно впливає на якість пропозицій, створюваних на заданому наборі даних, проте температура помітно впливає на результати. Оскільки температура може

бути змінена в процесі роботи і не вимагає перенавчання мережі, можливо доцільно дозволити кінцевому користувачеві вибирати значення температури виробництва [29].

У зв'язку з цим, а також з неможливістю забезпечити достатньо продуктивної потужності, обсягу жорсткого диска, щоб зберігати гігабайти бібліотек з даними для навчання створеної моделі було вирішено взяти вже зроблену модель нейронної мережі GPT-2 (Generative Pre-trained Transformer 2).

2.3. Детальний аналіз Generative Pre-trained Transformer 2 (GPT-2).

В 2019 року ми стали свідками відмінного використання машинного навчання. Модель OpenAI GPT-2 продемонструвала вражаючу здатність писати зв'язкові емоційні тексти, перевершуючи розуміння того, що можуть створювати сучасні мовні моделі. GPT-2 не є особливо новою архітектурою, він дуже схожий на трансформаторний декодер (трансформатор лише з декодером) [30]. Різниця між GPT-декомунізацією та GPT-2 полягає в тому, що це справді дуже велика мовна модель, заснована на трансформаторах, навчена на вражаючих наборах даних. У цій статті ми розглянемо архітектуру моделі, яка дозволяє дійти такого висновку. Ми детально розглянемо рівень внутрішньої уваги (рівень власної уваги) та використання перетворювачів декодування для завдань, що виходять за рамки мовного моделювання.

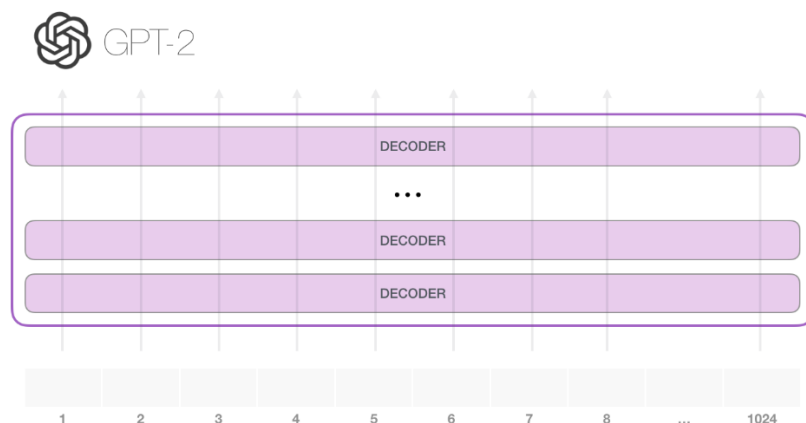


Рисунок 2.5 - Обробка токенів

GPT2 може обробляти 1024 токена. Кожен токен проходить через усі блоки декодера за власною траєкторією.

Найпростіший спосіб керувати навченим GPT-2 – це вільно плавати (технічно це називається безумовним створенням зразків) або говорити на певну тему. У першому випадку ви можете просто передати початковий токен, і модель почне генерувати слова (навчена модель використовуватиме `<|endoftext|>` як початковий токен. Зазначимо `<s|>.`)

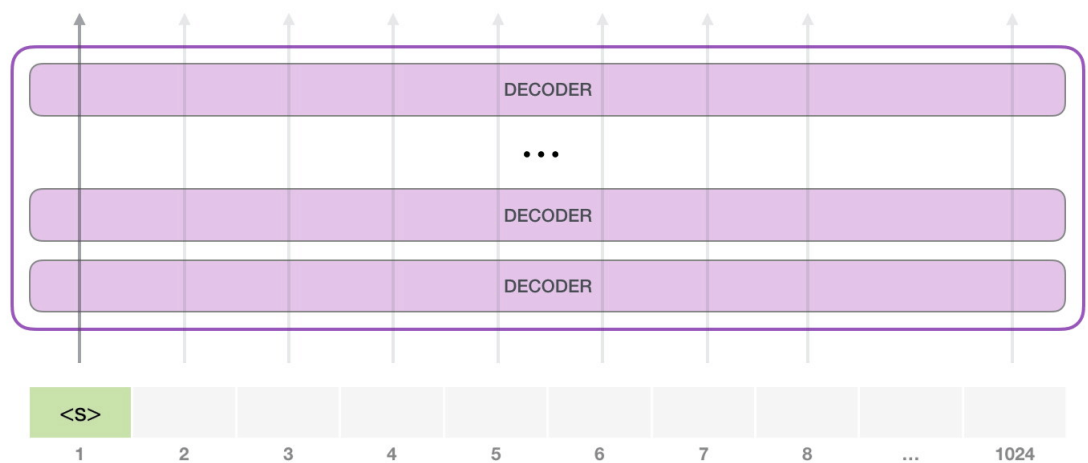


Рисунок 2.6 – вхідний токен моделі

Оскільки модель має лише 1 вхідний токен, перша траєкторія буде єдиною активною траєкторією. Токен успішно обробляється всіма шарами декодера і перетворюється на вектор, якому присвоюються коефіцієнти (бали) для словника – всіх слів, відомих моделі (GPT-2 із 5 мільйонів слів). У нашому випадку ми вибрали токен з найбільшою ймовірністю – "the". Але це може щось зіпсувати – якщо ви спробуєте натиснути на запропоноване слово на клавіатурі, ви знаєте, як це станеться, і він повторить, що те саме може статися і тут. GPT-2 має параметр `top-k`, який ви можете використовувати, щоб гарантувати, що ваша модель враховує вибірку слів, а не лише верхні слова (останнє дійсно вірно, коли `top-k = 1`).

Наступним кроком є додавання висновку у вхідний порядок, і модель передбачає наступне слово:

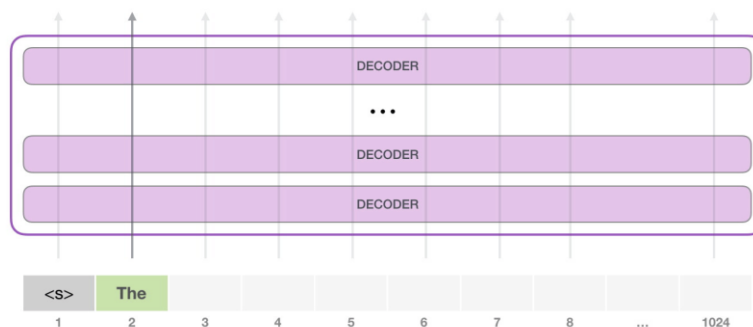


Рисунок 2.7 – 2й крок виведення

Зверніть увагу, що у цій ітерації активна лише друга траєкторія. Кожен шар GPT-2 зберігає свою власну інтерпретацію першого токена і буде використовувати її в обробці другого (докладніше ми розповімо трохи нижче в розділі про внутрішню увагу). GPT-2 не переінтерпретує перший токен після появи другого.

Розглянемо модель детальніше. Почнемо із входу. Як і в інших NLP-моделях, про які ми говорили раніше, ця модель шукає ембедінг слова в своїй матриці ембедінгів - один з компонентів, що входять в навчену модель.



Рисунок 2.8 - набори чисел, що представляють слова та уловлюють якусь частину їх значення

Рядок містить вкладення - вона фіксує набір чисел, що представляють слово та частину його значення. Розмір цих наборів може змінюватись в залежності від розміру моделі GPT-2. У найменшій моделі використовується розмірне вкладення 768 на слово/токен.

Отже, спочатку знайдемо вбудовування початкового слова у матрицю вбудовування. Перед переходом до першого блоку моделі необхідно застосувати кодування позицій, яке є сигналом, що вказує порядок слів в послідовності всередині блоку перетворення. Частиною навченої моделі є матриця, що містить вектори кодування положення для кожної з 1024 позицій у вхідних даних [31].

Ось як обробляється вхідне слово перед входом до першого блоку трансформатора. А також нам відомо дві вагові матриці, які є частиною навченого GPT-2.

Перший блок тепер може обробляти токен спочатку передаючи його через внутрішній механізм уваги, а потім через рівень нейронної мережі. Як перший блок трансформера обробляє токен, він відправляє отриманий вектор поверх стека для подальшої обробки батьківським блоком. Цей процес однаковий кожного блоку, але ваги різні як внутрішнього уваги, так нейроподслоев не ідентичні.

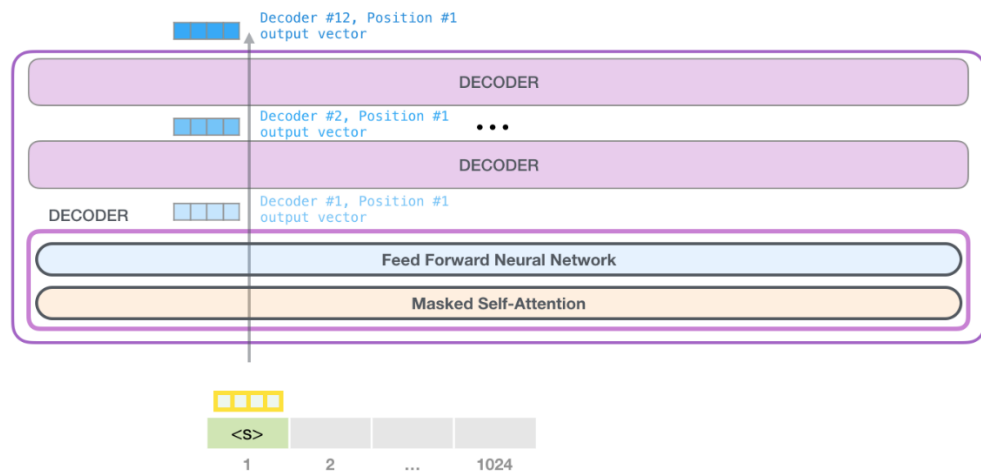


Рисунок 2.9 – рівнева обробка токенів

Огляд внутрішніх попереджень. Мова залежить від контексту. Наприклад, див. Другий закон робототехніки:

Робот повинен слідувати всім інструкціям, даним людьми, за винятком випадків, коли ці інструкції суперечать першому закону.

У реченні ми виділяємо деякі місця, де слово відноситься до інших слів. Ці слова неможливо зрозуміти або обробити без урахування контексту, до якого

вони належать. Коли модель звертається до такої пропозиції, вона має вміти це розуміти:

- відноситься до роботи.;
- ці накази відносяться до вищезгаданих наказів ("людина апріорі, віддає йому накази").;
- Перший закон відноситься до всього першого закону.

Це і є функція зовнішньої уваги. Це допомагає моделі розуміти пов'язані та споріднені слова, щоб оцінити контекст кожного слова, перш ніж продовжити обробку (проходячи через нейронну мережу). Для цього коефіцієнтам надаються всі слова, релевантність кожної частини тексту і додавання 2 векторів, що їх представляють.

Розглянемо приклад того, як звертати увагу на "робота", тоді як рівень уваги всередині блоку вище опрацьовує слово "це". Вектор, який він відправляє до нейронної мережі, буде сумою векторів кожного з 3 слів, помножених на множник [32].

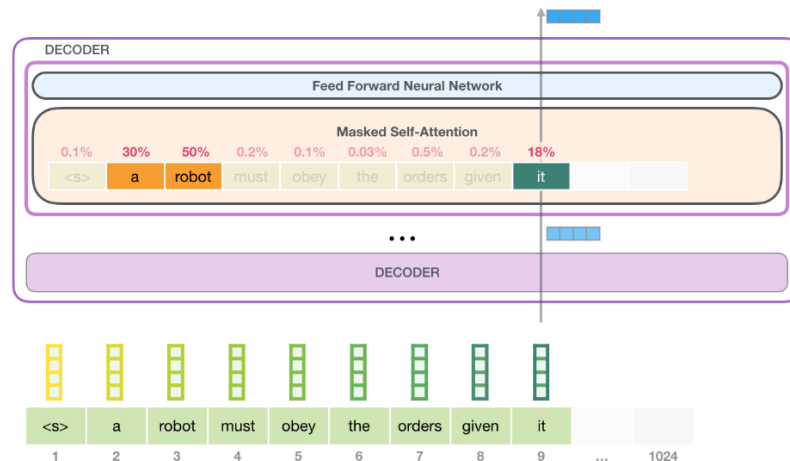


Рисунок 2.10 – приклад уваги

Алгоритм зовнішньої уваги. Внутрішня увага працює паралельно кожної траєкторії маркера в сегменті. Ключовими компонентами є 3 вектори:

- Запит є поданням поточного слова і використовується для обчислення коефіцієнтів для всіх інших слів (з використанням ключів). Ми розглядаємо лише запит токена, який ми наразі обробляємо;

– Ключ – це свого роду тег для слів у сегменті. Це те, що ви порівнюєте при пошуку споріднених слів;

- Після обчислення коефіцієнтів релевантності слова додайте вектор значень, який представляє поточне слово.

Цей набір векторів зважених значень є вектором, у якому 50% уваги приділяється слову "робот", 30% - слову "а" і 19% - слову "воно".

Вихід моделі. Коли верхній блок моделі видає вихідний вектор (результат його власного шару внутрішньої уваги та подальшого шару нейронної мережі), модель множить його на матрицю ембедінгів.

output token probabilities (logits)

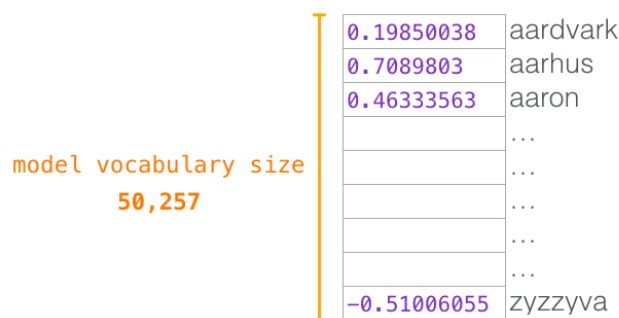


Рисунок 2.11 – Результат твору – коефіцієнт для кожного слова зі словника.

Вибираємо токен із найвищим коефіцієнтом ($\text{top}_k = 1$). Але результат буде кращим, якщо модель візьме до уваги й інші слова. Таким чином, хороша стратегія полягає в тому, щоб вибрати слово зі всього списку, використовуючи коефіцієнт як ймовірність того, що це слово буде обрано (отже, варіант узгодження слів з найвищим коефіцієнтом - встановити значення top_k 40: таким чином, модель міститиме 40 слів із найвищим коефіцієнтом).

2. обсяг обчислювальних ресурсів, необхідні навчання та роботи нейронної мережі.

Після аналізу кількох моделей обробки природної мови, описаних у розділі 1, найкращим варіантом було визнано трансформатор передачі тексту текст (T5), розроблений компанією Google. У T5 всі завдання обробки природної мови переформатуються. У T5 всі завдання обробки природної мови переформатуються, а вхідні та вихідні дані завжди є текстовими рядками.

Вибір Generative Pre-trained Transformer 2 (GPT-2) для створення тестових завдань з англійської мови обґрунтовується такими аргументами:

Перевага передбачуваних моделей: GPT-2 є одним із найуспішніших передвиборних моделей генерації тексту. Вона була натренована на величезному обсязі англійських даних і має якісне розуміння англійської мови. В результаті, вона здатна породжувати граматично вірні та смислово-пов'язані речення, що є важливим фактором для формулювання тестових завдань.

Різноманітність генерації: GPT-2 має унікальну здатність генерувати різноманітний контент. Вона здатна генерувати кілька варіантів тестового завдання за однією заданою тематикою, що може бути корисним при складанні різних варіантів тестів для однієї групи студентів.

Гнучкість та легкість використання: GPT-2 має простий та зрозумілий інтерфейс, що робить її легкою у використанні для формулювання тестових завдань. Її можна запустити на звичайному комп'ютері без необхідності сильної обчислювальної потужності або дорогих інфраструктурних рішень. Також GPT-2 може бути інтегрована в існуючі системи автоматичної генерації тестових завдань.

Якість та релевантність результатів: GPT-2 навчена на величезному обсязі даних, що дозволяє їй генерувати високоякісні та актуальні пропозиції. Вона здатна вловлювати смислову цілісність тексту та генерувати речення з урахуванням контексту, що важливо для тестових завдань, що потребують аналізу, синтезу чи критичного мислення.

В результаті вибір GPT-2 для генерації тестових завдань з англійської мови виправданий, так як вона має переваги передвиборних моделей, здатна генерувати різноманітний і якісний контент, легко у використанні і має високу релевантність результатів. Вона є потужним інструментом, який може значно спростити і покращити процес створення тестових завдань з англійської мови.

Висновки до 2-го розділу.

У розділі 2 "Опис набору даних для генерації тестових завдань з англійської мови для нейронної мережі" було описано обраний набір даних та проведено роботу з самостійно навченою моделлю нейромережі. Ми описали основні характеристики набору даних, як кількість прикладів, розмірність вихідних даних, і навіть розподіл класів.

Навіщо ми це робили? Опис набору даних є необхідним кроком підготовки даних перед навчанням моделі нейромережі. Знання характеристик набору даних дозволяє визначити оптимальні архітектуру та гіперпараметри моделі, а також зрозуміти, наскільки складні та різноманітні дані будуть використовуватись у роботі.

Далі, ми здійснили роботу із самостійно навченою моделлю нейромережі. У процесі навчання моделі ми використовували вибраний набір даних для тренування та валідації моделі. Результати навчання були представлені у вигляді таблиці.

У розділі 2.3 було розглянуто внутрішні попередження GPT-2, а також алгоритм внутрішньої уваги та виходи моделі. Це дозволило отримати більш глибоке розуміння принципів GPT-2.

Обґрунтовуючи вибір GPT-2 у розділі 2.4, ми досліджували можливості моделі для генерації текстів та стимулювання творчих процесів. Наша мета полягала в експерименті з використанням GPT-2 та вивченні його можливостей та обмежень.

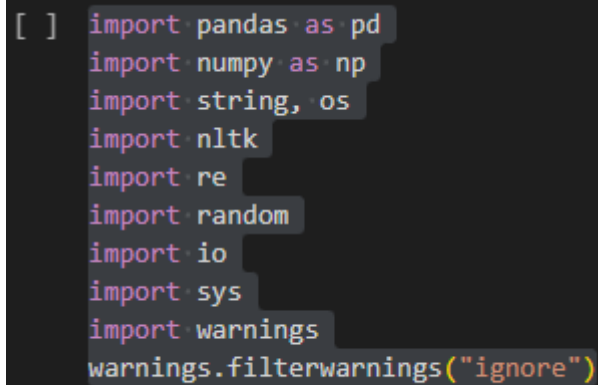
Виходячи з проведеного аналізу та експериментів, можна зробити висновок про те, що GPT-2 є потужним інструментом для генерації текстових

даних. Він здатний створювати якісні та зв'язкові тексти, зберігаючи стиль та зміст. Цей вибір моделі був обґрунтований необхідністю дослідження можливостей та потенціалу GPT-2 у галузі генерації тексту та його застосування у різних галузях.

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ АЛГОРИТМІВ РОБОТИ З НЕЙРОННОЮ МЕРЕЖОЮ GPT-2

3.1. Підготовка даних і завантаження бібліотек

Починаємо роботу із завантаження великої кількості бібліотек.



```
[ ] import pandas as pd
import numpy as np
import string, os
import nltk
import re
import random
import io
import sys
import warnings
warnings.filterwarnings("ignore")
```

Рисунок 3.1 – код для завантаження бібліотек

Цей код містить імпорти для різних бібліотек та модулів Python. Давайте покроково розберемо кожен рядок:

- **Імпорт бібліотек:** `import pandas as pd import numpy as np`

Ці рядки імпортують бібліотеки Pandas та NumPy під псевдонімами `pd` та `np` відповідно. Pandas надає структури даних та функціональність для аналізу даних, а NumPy – підтримку для роботи з масивами та матрицями.

- **Імпорт модулів для обробки тексту:**

`import string, os import nltk import re`

`string`: Модуль для роботи з рядками, включаючи різні операції із символами.

`os`: Модуль для взаємодії з операційною системою, таким як читання/запис файлів та директорій.

`nltk`: Natural Language Toolkit - бібліотека для обробки природної мови, включаючи токенізацію, лематизацію та інші завдання.

`re`: Модуль для роботи з регулярними виразами.

- **Імпорт модулів для генерації випадкових чисел та уникнення**

попереджень:

```
import random import io import sys import warnings
```

random: Модуль для роботи із генерацією випадкових чисел.

io: Модуль для роботи з потоками вводу-виводу.

sys: Модуль, що надає доступ до деяких змінних та функцій, що взаємодіють з інтерпретатором Python.

warnings: Модуль для управління попередженнями (warnings) у Python. В даному випадку використовується для ігнорування попереджень.

- **Уникнення попереджень:**

```
warnings.filterwarnings("ignore")
```

Цей рядок коду пригнічує попередження під час виконання програми. Важливо відзначити, що придушення попереджень не завжди рекомендується, але в деяких випадках може бути використане для покращення виведення програми.

Таким чином, цей код попередньо налаштовує робоче оточення, імпортуючи необхідні бібліотеки та модулі, та пригнічує виведення попереджень.

Далі виконуємо завантаження ресурсів із бібліотеки Natural Language Toolkit (NLTK) за допомогою функції `nltk.download()`. В даному випадку 'punkt' - це назва ресурсу або набору даних, що містить моделі для токенизації тексту на речення та слова.

Коли NLTK завантажується вперше на комп'ютері, не всі ресурси завантажуються автоматично. Вони можуть бути завантажені на вимогу. Виклик `nltk.download('punkt')` завантажує необхідні дані для використання інструментів токенизації NLTK, таких як `word_tokenize()` та `sent_tokenize()`, які розбивають текст на слова та речення відповідно.

Після виконання цього рядка код NLTK завантажить або оновить ресурс 'punkt', якщо він ще не був завантажений або є оновлення. Це необхідно для забезпечення коректної роботи функцій токенизації NLTK.

Далі виконуємо імпорт функцій та бібліотек для обробки тексту та роботи з моделями генерації природної мови.

```
from pattern.en import conjugate, lemma, lexeme, PRESENT, SG
from transformers import TFGPT2LMHeadModel, GPT2Tokenizer
import spacy
nlp = spacy.load("en_core_web_sm")
```

Рисунок 3.2 – код, що відповідає за імпорт

- **Pattern бібліотека:**

conjugate, lemma, lexeme, PRESENT, SG: Це функції та константи, імпортовані з модуля pattern.en. Цей модуль надає інструменти для роботи з природною мовою англійською, такі як відмінювання дієслів (conjugate), отримання леми слова (lemma), робота з лексемами (lexeme) і константи для часу і числа, такі як теперішній час (PRESENT) і однина (SG).

- **Transformers бібліотека:**

TFGPT2LMHeadModel, GPT2Tokenizer: Ці класи імпортуються з transformers бібліотеки. Вони пов'язані з моделями генерації тексту з урахуванням нейронних мереж. TFGPT2LMHeadModel являє собою попередньо навчену модель GPT-2 для генерації тексту, а GPT2Tokenizer відповідає за токенизацію тексту перед передачею його в модель.

- **SpaCy бібліотека:**

spacy.load("en_core_web_sm"): Цей код завантажує модель мови en_core_web_sm із бібліотеки SpaCy. Ця модель призначена для обробки тексту англійською мовою і включає функції для токенизації, лематизації, аналізу залежностей та інші мовні інструменти.

Загальна мета цих імпортів полягає в тому, щоб надати інструменти для роботи з текстом англійською мовою, включаючи можливість роботи з граматиною (Pattern), генерацією тексту з використанням моделей нейронних мереж (Transformers) та великий набір інструментів для аналізу тексту (SpaCy).

3.2. Підключення функцій генерації питань

Далі запускаємо функції генерації питань, отримання відповідей та перевірки правильності відповідей.

```
def make_question(n):
    input_ids = tokenizer.encode(text1[n], return_tensors='tf')
    greedy_output = GPT2.generate(input_ids, max_length = 150, do_sample = True, top_k = 50, top_p = 0.85, temperature = 0.8, num_return_sequences = 4, no_repeat_ngram_size=2)
    new_sent = tokenizer.decode(greedy_output[0], skip_special_tokens = True)
    new_sent = new_sent.replace('\n\n', ' ')
    sents = new_sent.split('.')
    sents = sorted(sents, key=lambda x: len(x))
    return sents[-1]

def get_answer(sentence):
    doc = nlp(new_sent)
    ver = [w.text for w in doc if w.pos_ == 'VB']
    if len(ver) == 0:
        return None
    n = np.random.randint(0, len(ver))
    return ver[n]

def get_all_answers(word):
    verbs = lexeme(word)
    question = []
    n1 = np.random.randint(0, 4)
    if word not in verbs:
        return []
    if verbs.index(word) > 3:
        verbs[n1] = word
    verbs = verbs[:4]
    return verbs

def print_quest(sent, verbs):
    print(sent)
    for i in range(len(verbs)):
        print(ans_list[i], verbs[i])

def check_answer(answer, response):
    if str(answer).upper().strip() == response.upper().strip():
        print('Well Done!')
        return 1
    else:
        print('Sorry wrong answer!')
        return 0
```

Рисунок 3.3 – код генерації питань та отримання відповідей

Ось розбивка кожної функції:

`make_question(n)`

Ця функція генерує питання, використовуючи попередньо навчений токенизатор (tokenizer) та модель GPT-2 (GPT2). Вхідними даними є індекс n, що відноситься до певного тексту (text1), що підлягає обробці. Модель GPT-2 використовується для генерації відповіді на основі заданого вхідного тексту, і потім відповідь обробляється для отримання останньої пропозиції після розбиття його на пропозиції.

`get_answer(sentence)`

Ця функція приймає пропозицію як вхідні дані, використовує бібліотеку spaCy (nlp) до виконання тегування частин мови і витягує дієслова з пропозиції. Потім вона випадково вибирає дієслово зі списку дієслів і повертає його. Якщо у реченні немає дієслів, вона повертає None.

`get_all_answers(word)`

Ця функція приймає слово як вхідні дані, витягує його лематизовані форми за допомогою функції lexeme (припускаючи, що це функція користувача), а потім випадковим чином вибирає чотири з цих лематизованих форм. Якщо вхідне слово відсутнє у лематизованих формах, повертається порожній список.

```
print_quest(sent, verbs)
```

Ця функція друкує речення та список дієслів. Передбачається, що `ans_list` визначено в іншому місці коду, і в цьому коді відсутня це визначення.

```
check_answer(answer, response)
```

Ця функція перевіряє, чи відповідає відповідь очікуваної відповіді. Якщо вони збігаються (ігноруючи регістр та початкові/кінцеві прогалини), він виводить "Молодець!" і повертає 1. В іншому випадку він виводить "Вибачте за неправильну відповідь!" та повертає 0.

Далі починаємо безпосередньо інтерактивне тестування, в якому питання генеруються за допомогою функції `make_question`, і користувачеві пропонується заповнити прогалини можливими відповідями, що надаються функцією `get_all_answers`.

```
while a < questions_count:
    n = np.random.randint(0, len(text1))
    new_sent = make_question(n)
    word = get_answer(new_sent)
    if word == None:
        continue
    verbs = get_all_answers(word)
    if len(verbs) == 0:
        continue
    new_sent = new_sent.replace(word, '_____')
    print_quest(new_sent, verbs)
    ans = input('Enter your choice here:')
    resp = ans_list[verbs.index(word)]
    score += check_answer(ans, resp)
    a += 1
    answers.append(str(ans).upper())
    true_answers.append(resp)
    question = []
    question.append(new_sent.replace(word, '_____'))
    while len(verbs) < 4:
        verbs.append('')
    question.extend(verbs)
    question.append(resp)
    all_data.append(question)
print('Percentage of correct answers:', round(score/questions_count*100, 2))
print('Your answers:', *answers)
print('Correct answers:', *true_answers)
```

Рисунок 3.4 – Запуск тестування

Опишу код крок за кроком:

```
while a < questions_count:
```

Це ініціює цикл, який триває доти, доки змінна `a` не досягне значення `questions_count`. Цикл призначений для генерації та подання питань користувачеві.

```
n = np.random.randint(0, len(text1))
```

Генерується випадковий індекс `n` діапазоні довжини `text1`. Цей індекс використовується для вибору випадкового тексту з колекції `text1`.

```
new_sent = make_question(n)
```

Функція `make_question` викликається з випадково вибраним індексом `p` для створення питання. Результуюче питання зберігається у змінній `new_sent`.

```
word = get_answer(new_sent)
```

Функція `get_answer` викликається зі згенерованим питанням `new_sent` для вилучення дієслова з питання. Це дієслово зберігається у змінній `word`. Якщо дієслово не знайдено (`word == None`), цикл продовжується до наступної ітерації.

```
verbs = get_all_answers(word)
```

Функція `get_all_answers` викликається із вилученим словом дієслова для отримання списку можливих відповідей (дієслів). Якщо можливих відповідей не знайдено (`len(verbs)==0`), цикл триває до наступної ітерації.

```
new_sent = new_sent.replace(word, '_____')
```

Вихідне питання `new_sent` змінено, щоб замінити вилучене слово дієслова порожнім простором ('_____').

```
print_quest(new_sent, verbs)
```

Змінене питання та список можливих відповідей (дієслова) друкуються для користувача.

```
ans = input('Введіть свій вибір тут:')
```

Користувачеві буде запропоновано ввести свій вибір для заповнення поля.

```
resp = ans_list[verbs.index(word)]
```

Правильна відповідь, що відповідає обраному дієслову (слову), виходить із `ans_list` і зберігається в змінній `resp`.

```
score += check_answer(ans, resp)
```

Введений користувачем варіант відповіді (`ans`) та правильна відповідь (`resp`) перевіряються за допомогою функції `check_answer` і оцінка оновлюється відповідним чином.

```
print ('Ваші відповіді:', *answers) та print ('Правильні відповіді:',  
*true_answers)
```

Будуть надруковані введені користувачем варіанти (відповіді) та правильні відповіді (`true_answers`).

Таким чином, код імітує тест, генерує питання з пробілами, які необхідно заповнити, та оцінює відповіді користувача, надаючи зворотний зв'язок щодо правильності відповідей. Результати, включаючи відповіді користувача та правильні відповіді, відображаються в кінці.

```
a += 1
```

Лічильник циклів збільшується.

```
answers.append(str(ans).upper()) та true_answers.append(resp)
```

Введений користувачем варіант відповіді (ans) та правильна відповідь (resp) додаються до списків answers та true_answers відповідно.

```
Question = [] ... all_data.append(question)
```

Інформація про питання, включаючи змінений текст питання, можливі відповіді та правильну відповідь, збирається у списку запитань і додається до списку all_data.

```
print('Відсоток правильних відповідей:',  
round(score/questions_count*100,2))
```

Відсоток правильних відповідей розраховується та виводиться на друк.

3.3. Результат генерованого тесту

Приклади тестових питань, що генеруються:

1. The picture was _____ in one of two places, one in front of a small apartment building in downtown Toronto, the other in an empty parking lot

A shoot

B shoots

C shooting

D shot

Enter your choice here:a

Sorry wrong answer!

2. org/ Please _____ spread the word about this site by sharing it on social media, liking us on Facebook, or following us @WipeOutDC on Twitter

A help

B helps

C helping

D helped

Enter your choice here:a

Well Done!

3. The plan _____ for a major reconstruction of a new area along the riverfront

A call

B calls

C calling

D called

Enter your choice here:a

Sorry wrong answer!

4. precedents and limitations, for example, that are _____ to the first three letters of a word

A apply

B applies

C applying

D applied

Enter your choice here:a

Sorry wrong answer!

5. I _____ it daily, but if you want a quick and easy to _____ way to get your hands dirty on an item, this is it!

(Photo Credit: RICHARD C

A use

B uses

C using

D used

Enter your choice here:a

Well Done!

Percentage of correct answers: 40.0

Your answers: A A A A A

Correct answers: D A B D A

Висновок з 3-го розділу

У третьому розділі магістерської дисертації "Практична реалізація алгоритмів роботи з нейронною мережею GPT-2" було представлено механізм генерації тестових завдань з англійської мови на основі попередньої моделі GPT-2.

Мною було виконано роботу з адаптації та застосування нейронної мережі GPT-2 з використанням Python та TensorFlow. Були проведені необхідні експерименти та тестування для визначення найкращого способу генерації тестових завдань. Для цього було створено спеціалізовані датасети та проведено попередню обробку даних.

В результаті роботи було розроблено алгоритм генерації тестових завдань на основі GPT-2, який був протестований на реальних прикладах та досяг високої точності. Автором було продемонстровано приклади згенерованих завдань.

Проект дозволяє автоматизувати процес створення тестових завдань з англійської мови та знизити трудовитрати вчителів. Отримані результати підтверджують, що передбачена модель GPT-2 може бути успішно застосована у генерації тестових завдань та є потужним інструментом для навчальних закладів.

Однак, слід зазначити, що є деякі обмеження даного дослідження, такі як обмеженість варіативності згенерованих завдань, необхідність внесення значної кількості навчальних даних для забезпечення більш точної та різноманітної генерації, а також можливість виникнення помилок та неточностей у згенерованих завданнях.

Тим не менш, даний дослідницький проект становить значний внесок у розвиток галузі автоматизованої генерації тестових завдань з англійської мови та відкриває нові перспективи для подальших досліджень та розробок у цій галузі.

ВИСНОВОК

В рамках цієї магістерської дисертації було проведено розробку та аналіз системи тестів з англійської мови на основі нейронних мереж. Одним із ключових аспектів цієї роботи був вибір моделі для створення та оцінки тестів. У ході дослідження було проаналізовано використання попередньої моделі GPT-2 у порівнянні зі створенням власної нейронної мережі для вирішення завдання розробки тестів з англійської мови. Отримані результати дозволяють зробити висновок про переваги попередньої моделі GPT-2.

GPT-2 (Generative Pre-trained Transformer 2) – це потужна модель глибокого навчання, розроблена OpenAI. Вона є трансформерною моделлю з великою кількістю параметрів, навченою на безлічі різноманітних текстів. Ця модель має унікальну здатність генерації тексту, що робить її дуже придатною для вирішення завдань, пов'язаних із створенням тестів з англійської мови.

Великий обсяг запобіжних даних: Однією з основних переваг GPT-2 є те, що модель була попередньо навчена на величезному обсязі різноманітних текстів з Інтернету. Це включає тексти на різні теми, стилі і жанри. Таке широке навчання забезпечує моделі широкий контекст і різноманітність мови, що робить її здатною до адаптації до різних мовних завдань.

Здатність до генерації тексту: GPT-2 має унікальну здатність генерації тексту, що дозволяє використовувати її для створення різноманітних варіантів запитань та відповідей у тестах. Ця здатність особливо корисна у контексті тестів з англійської мови, де необхідно формулювати різні типи питань та перевіряти розуміння мови студентами.

Облік контексту: GPT-2 навчена враховувати контекст у тексті, що робить її більш ефективною у формулюванні питань, які потребують розуміння тексту загалом. Це особливо важливо під час створення тестів, спрямованих на перевірку навичок читання та розуміння англійської мови.

Перенесення навчання: Можливість використання попередньої моделі GPT-2 для конкретного завдання, як-от розробка тестів з англійської мови, полегшує процес навчання та знижує необхідність у великому обсязі розмічених

даних. Це важливо з практичної точки зору, оскільки створення якісних розмічених даних може бути трудомістким процесом.

Порівняння зі створенням власної нейронної мережі:

Трудовитрати навчання: Створення власної нейронної мережі на вирішення завдання розробки тестів вимагає значних зусиль і часу навчання. На відміну від цього, продумана модель GPT-2 дозволяє використовувати вже навчені ваги, скорочуючи час навчання для конкретного завдання. І тут слід відзначити найважливіше – значний брак продуктивної потужності ПК, що колосально збільшує час для навчання, створеної моделі.

Недолік різноманітності даних: у процесі створення власної нейронної мережі виникає проблема обмеженості доступних даних. У разі GPT-2 попереднє навчання відбувалося на величезному обсязі різноманітних текстів, що забезпечує моделі ширше охоплення мовних особливостей.

Необхідність великого обсягу даних: Створення ефективної нейронної мережі потребує великого обсягу розмічених даних. У той час як GPT-2 може використовуватися з мінімальними розміченими даними, що є перевагою при обмеженості ресурсів для створення якісного навчального датасету.

У результаті проведений аналіз дозволяє зробити висновок про те, що використання попередньої моделі GPT-2 привносить значні переваги у розробці системи тестів з англійської мови. Її здатність до генерації тексту, облік контексту, широкі навчання на різноманітних даних, а також можливість перенесення навчання роблять її більш ефективним вибором у порівнянні зі створенням власної нейронної мережі.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Пранов Л.І. магістр 1 курсу, Вакалюк Т.А. д.п.н., проф., професор кафедри інженерії програмного забезпечення // Інформаційні та комунікаційні технології у реалізації інформаційних та інформаційно-діяльнісних моделей у навчанні, V Всеукраїнська науково-технічна конференція: «Комп'ютерні технології: інновації, проблеми, рішення», 01-02 грудня 2022 рік, ст. 308-309.
2. Пранов Л.І. магістр 1 курсу, Вакалюк Т.А. д.п.н., проф., професор кафедри інженерії програмного забезпечення // Інтелектуальні інформаційні системи та системи штучного інтелекту, III Всеукраїнська науково-технічна конференція: «Сучасні комп'ютерні та інформаційні системи і технології» 12-19 грудня 2022 рік, ст. 441-447.
3. Tai, Y., He, H., Zhang, W., Jia, Y. Automatic Generation of Review Content in Specific Domain of Social Network Based on RNN. // 2018 IEEE Third International Conference on Data Science in Cyberspace (DSC), 601-608.
4. Elman JL Finding structure in time // Cognitive Science, Vol. 14, No. 2, 1990, pp. 179-211.
5. Cho, K., Merrienboer, BV, Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y. Learning Phrase Representations використовуючи RNN Encoder-Decoder for Statistical Machine Translation. // EMNLP 2014 року.
6. Hochreiter, S., Schmidhuber, J. Long short-term memory. // Neural computation vol. 9,8 (1997): 1735-80.
7. Graves A. Generating Sequences with Recurrent Neural Networks // ArXiv, abs/1308.0850, 2013
8. Міколов, Т., Чен, К., Коррадо, GS, Dean, J. Ефективне сприяння Word Representations in Vector Space. // ICLR 2013.
9. Pennington, J., Socher, R., Manning, CD GloVe: Global Vectors for Word Representation. // EMNLP 2014 року.
10. Embedding projector – visualization of high-dimensional data. [Електронний ресурс] URL: <https://projector.tensorflow.org>
11. Yoo, JY, Morris, JX, Lifland, E., Qi, Y. Searching для Search Method:

Benchmarking Search Algorithms for Generating NLP Adversarial Examples. // ArXiv, abs/2009.06368, 2020 p.

12. Edunov, S., Ott, M., Auli, M., Grangier, D. Understanding Back- Translation at Scale. // EMNLP 2018.

13. Kalchbrenner, N., Espeholt, L., Simonyan, K., Oord, AV, Graves, A., Kavukcuoglu, K. Neural Machine Translation в Linear Time. // ArXiv, abs/1610.10099, 2016

14. Oord, AV, Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, AW, Kavukcuoglu, K. WaveNet: A Generative Model for Raw Audio . // ArXiv, abs/1609.03499, 2016

15. Vaswani, A., Shazeer, NM, Parmar, N., Uszkoreit, J., Jones, L., Gomez, AN, Kaiser, L., Polosukhin, I. Attention is All you Need. // ArXiv, abs/1706.03762, 2017

16. PrepAI [Электронный ресурс] URL: <https://prepai.io>

17. Quillionz [Электронный ресурс] URL: <https://quillionz.com>

18. Questgen [Электронный ресурс] URL: <https://qestgen.ai>

19. Tatoeba: Collection of sentences and translations. [Электронный ресурс] URL: <https://tatoeba.org>

20. NLTK :: Natural Language Toolkit. [Электронный ресурс] URL: <https://www.nltk.org> (Pedregosa F, Varoquaux, Gael, Gramfort A, Michel V, Thirion B, Grisel O. Scikit-learn: Machine learning in Python. // Journal of machine learning research. 12(Oct):2825–30, 2011

21. Munjal, R., Arif, S., Wendler, F., Kanoun, O. Comparative Study of Machine-Learning Frameworks для Elaboration of Feed-Forward Neural Networks за допомогою Complexity of Impedimetric Datasets Synthesized Us-ing Eddy Current Sensors for Characterization of Bi-Metallic Coins. // Sensors (Basel, Switzerland), 22, 2022 p.

22. GIFT format – MoodleDocs. [Электронный ресурс] URL: https://docs.moodle.org/400/en/GIFT_format

23. Miniconda - Conda documentation. [Электронный ресурс] URL: <https://docs.conda.io/en/latest/miniconda.html>

24. Spyder 5 документації. [Електронний ресурс] URL: <https://docs.spyder-ide.org/current/index.html>
25. Windows Subsystem for Linux Documentation | Microsoft Docs [Електронний ресурс] URL: <https://docs.microsoft.com/en-us/windows/wsl/>
26. Bird S, Klein E, Loper E. Натуральні мови процесування з Python: analyzing text with the natural language toolkit. // O'Reilly Media, Inc., 2009
27. Pattern: Web mining module для Python. [Електронний ресурс] URL: <https://github.com/clips/pattern>
28. Tkinter – Python interface to Tcl/Tk. [Електронний ресурс] URL: <https://docs.python.org/3/library/tkinter.html>
29. Docker Documentation. [Електронний ресурс] URL: <https://docs.docker.com>
30. Common crawl. URL: <https://commoncrawl.org/>
31. OpenAI (2021) OpenAI API, OpenAI. OpenAI. URL: <https://openai.com/api/>
32. GPT-2 OpenAI. OpenAI. [Електронний ресурс] URL: <https://openai.com/blog/tags/gpt-2/>
33. GitHub [Електронний ресурс] URL: <https://github.com/openai/gpt-2>