Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження текстових генеративних систем з асоціативною пам'яттю
(тема)

Виконав:
студент 2 курсу, групи ІПЗм-22-3

Мамочка Є. І.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми Освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Керівник проф. Андрій ЄРОХІН
(посада, прізвище)

Допускається до захисту:

Зав. кафедри, проф. _____ Зоя ДУДАР
(підпис)

2024р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
зав. каф. ПІ, к.т.н., проф.
_____ Дудар З. В.
(підпис)
«___» _____ 2024р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

Студентові Мамочка Євгенію Ігоровичу

1. Тема роботи: «Дослідження текстових генеративних систем з асоціативною пам'яттю»

Затверджена наказом університету від «29» березня 2024 р. № 250 Ст

2. Термін здачі студентом закінченої роботи «07» червня 2024 р.

3. Вхідні дані до проекту:
інформація щодо нейронних мереж, галузі машинного навчання, інформація щодо мовних систем, датасет з відкритого ресурсу, модель GPT-2, середовище розробки PyCharm

4. Перелік питань, що потрібно опрацювати в роботі 1 постановка задачі

аналіз та порівняння двох систем з різними типами пам’яті відносно доцільності використання сам е асоціативної пам’яті у генеративних системах, проведення експерименту та аналіз результатів

Календарний план

Номер	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	29.03.2024	*виконано*
2	Виявлення проблематики галузі	31.03.2024	*виконано*
3	Постановка задачі	03.04.2024	*виконано*
4	Розробка моделі з асоціативною пам’яттю	03.04.2024 – 10.05.2024	*виконано*
5	Проведення порівняльного експерименту	11.05.2024	*виконано*
6	Підготовка пояснювальної записки	03.04.2024 – 15.05.2024	*виконано*
7	Підготовка презентації та доповіді	15.05.2024 – 20.05.2024	*виконано*
8	Нормоконтроль	22.05.2024	*виконано*
9	Рецензування	25.05.2024	*виконано*
10	Занесення диплома в електронний архів	27.05.2024	*виконано*
11	Попередній захист	27.05.2024	*виконано*
12	Допуск до захисту у зав. кафедри	06.06.2024	*виконано*
13	Захист магістерського дослідження	07.06.2024	*виконано*

Дата видачі «29» березня 2024р.

Керівник _____
(підпис)

Завдання отримав _____
(підпис)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка містить 69 стор., 20 рис., 2 табл., 15 джерел.

НЕЙРОННА МЕРЕЖА, МАШИННЕ НАВЧАННЯ, ПАМ’ЯТЬ, ГЕНЕРАЦІЯ, ГЕНЕРАТИВНІ МОДЕЛІ, ТЕКСТ, GPT-2.

Об’єкт дослідження – генеративні системи з різними типами пам’яті.

Мета роботи – Дослідження сфери машинного навчання, нейронних мереж та їх типів пам’яті, проведення практичного дослідження генеративних системи з різними типами пам’яті.

Результат роботи – виконане порівняння двох генеративних систем з метою виявлення доцільності використання саме асоціативної пам’яті у даних системах.

NEURAL NETWORK, MACHINE LEARNING, MEMORY, GENERATION, GENERATIVE MODELS, TEXT, GPT-2.

The object of research is generative systems with different types of memory.

The purpose of the work is to study the field of machine learning, neural networks and their types of memory, conducting a practical study of generative systems with different types of memory.

The result of the work is a comparison of two generative systems with the aim of identifying the expediency of using associative memory in these systems.

Заява щодо самостійного виконання кваліфікаційної роботи та можливості її публікації в електронному архіві відкритого доступу ElArKhNURE.

Я, *Мамочка Євгеній Ігорович*, студент гр.*ІПЗм-22-3*, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження текстових генеративних систем з асоціативною пам'яттю» що буде представлена для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Усі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови до допуску курсової роботи до захисту та застосування дисциплінарних заходів.

Робота виконана відповідно до вимог академічної доброчесності та пройшла перевірку на плагіат, який складає – 3.52%.

ЗМІСТ

Вступ 8

1 Аналіз предметної області та постановка задач дослідження 9

1.1 Проблематика що розглядається 9

1.2 Засоби проведення дослідження 9

1.3 Формування дослідницького завдання магістерського дослідження 10

2 Дослідження та аналіз предметної галузі 12

2.1 Аналіз сфери машинне навчання 12

2.2 Головні відомості про нейронні мережі 13

2.3 Асоціативна пам’ять 15

2.4 Архітектури пам’яті нейронних мереж 17

2.4.1 RNN 18

2.4.2 Трансформер 19

2.4.3 LSTM memory 22

2.5 Порівняння трансформерів та моделей LSTM 24

3 Підготовка до проведення дослідження 26

3.1 Підготовка даних до проведення експерименту 26

3.2 Вибір моделей нейронних мереж 26

3.3 Бачення реалізації задач дослідження 31

4 Процес розробки системи 32

4.1 Структура проекту 32

4.1 Розробка основної моделі трансформеру 33

4.2 Принцип роботи Fine Tuning (Перенавчання) 33

4.3 Робота асоціативної пам’яті 38

4.4 Опис взаємодії з системи 42

4.5 Розробка візуалізації результату 45

5 Проведення експерименту та аналіз результатів 46

5.1 Проведення аналізу отриманих результатів 46

Висновки 50

Перелік джерел посилання 51

Додаток А Перелік джерел посилання за науковими напрямами керівника та науковців кафедри програмної інженерії 53

Додаток Б Результати перевірки на унікальність тексту в базі ХНУРЕ 54

Додаток В Слайди презентації 55

Додаток Г Тексти наукових публікацій за темою кваліфікаційної роботи 62

Додаток Д Зміст файлу Result. Згенерований текст та показники 68

Додаток Е Експертний висновок результатів перевірки кваліфікаційної роботи на відповідність оформлення вимогам ДСТУ 3008: 2015 69

ВСТУП

Сучасний науковий прогрес не стоїть на місці та інтенсивно розвиває технології, випереджаючи їхній розвиток і введення новацій. Нещодавно для отримання необхідної інформації доводилося витрачати значний час на пошук відповідей на поставлені питання. А зараз майже кожна компанія націлена на вдосконалення процесу пошуку шляхом інтеграції систем, що розроблені за аналогією з людським мозком, у свої продукти. Ці системи, також відомі як нейронні мережі.

Нейронні мережі вміють працювати з різними типами даних, такими як зображення, звук, відео, текст тощо. Окрім того вони відрізняються одна від одної своєю архітектурою, типом пам’яті, тощо. У даному дослідженні ми вирішили дослідити асоціативний тип пам’яті саме в мовній моделі, через те що у цьому типі моделей легко перевірити асоціацію слів та контекст слів при генерації тексту.

На даний момент існує багато доступних систем генерації тексту, які відрізняються за якістю в залежності від моделі. Якість тексту прямо пропорційна якості моделі, яка, у свою чергу, залежить від розміру, типу та обсягу навчальних даних, а також типу пам'яті системи.

Теоретично асоціативна пам'ять у мовних моделях передбачає навчання пам'яті цих моделей без вчителя, що спрощує генерацію тексту в цілому, а також розробку таких систем. Це дослідження спрямоване на підвищення ефективності генерації тексту з використанням асоціативної пам’яті.

У даному магістерському дослідженні експериментально порівняно дві нейронні мережі з мето щодо дослідження доцільності використання асоціативної пам’яті у генеративних системах.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Проблематика що розглядається

На даний час з’являється велика кількість різних систем щодо генерації тексту. Якість згенерованого тексту залежить від самої моделі. Погана модель – поганий згенерований текст. Сама така модель, в свою чергу залежить не тільки від її розміру, її типу, кількості даних на яких навчалася, а й від типу пам’яті даної системи. Існує багато різних варіантів організування пам’яті у нейронних мережах, наприклад асоціативна пам’ять, що є більш природною для пам’яті людини. Цікаво дослідити такий тип організування пам’яті не у мозку люди, а у машині – нейронній мережі.

Теоретично, асоціативна пам’ять у мовних моделях, має на увазі – навчання пам’яті цих моделей без вчителя, що спрощує генерацію тексту у цілому, а також, розробку таких систем. У даному дослідженні, ми хочемо перевірити чи є доцільним використання асоціативної пам’яті у нейронних мережах з генерації тексту.

1.2 Засоби проведення дослідження

Проведення даного магістерського дослідження буде виконуватись завдяки мові програмування Python.

Сам код буде написано у середовищі розробки Pycharm.

У ході виконання дослідження буде використано декілька бібліотек, а саме:

TensorFlow - відкрита програмна бібліотека для машинного навчання, розроблена компанією Google для розв'язання завдань побудови і тренування нейронної мережі з метою автоматичного знаходження і класифікації образів, досягаючи якості людського сприйняття. У магістерському дослідженні буде використатися для створення, компіляції та навчання мовної моделі.

Keras – це високорівнева бібліотека для глибокого навчання. Вона надає простий та інтуїтивно зрозумілий інтерфейс для створення та навчання нейронних

мереж.

Matplotlib - це бібліотека, яка використовується для створення графіків, діаграм, графічних візуалізацій та інших видів ілюстрацій. У магістерському дослідженні буде використатися для візуального відображення результатів роботи розроблених систем.

NumPy - open-source модуль, який надає загальні математичні та числові операції у вигляді пре-компільованих, швидких функцій. У магістерському дослідженні буде використатися для спрощення процесу розробки.

Re - модуль який надає операції зіставлення шаблонів регулярних виразів, використовується для спрощення розробки програмних продуктів завдяки використанню певних шаблонів. У магістерському дослідженні буде використано для нормалізації тексту.

Саме навчання даних систем буде виконуватись на сервері Microsoft Azure.

Для перевірки даних систем буде використовуватись метрика accuracy, а більш детально - порівняння очікуваних даних і даних нейронної мережі.

Через велику кількість даних в матрицях, доцільно буде розпаралелити дії, для чого буде використано один із видів оптимізації, а саме - TPU.

Перейдемо до формування дослідницького завдання магістерського дослідження.

1.3 Формування дослідницького завдання магістерського дослідження

Самою суттю нашого дослідження в цілому – є перевірка двох текстових генеративним систем з різними типами пам’яті на однакових даних, з метою виявлення доцільності використання саме асоціативного типу пам’яті у таких системах.

Розкладемо задачу на пункти що необхідно буде виконати під час саме магістерського дослідження:

– розробити дослідницьку задачу щодо магістерського дослідження;
– зібрати теоретичний матеріал щодо теми та галузі дослідження –

машинного навчання;

- привести основні відомості щодо нейронних мереж;
- привести основні відомості щодо структури мовних систем;
- привести основні відомості щодо різних типів пам’яті у нейронних мережах;
- знайти мовну модель що відповідає певним вимогам до проведення магістерського дослідження;
- переробити обрану мовну модель щодо задач магістерського дослідження, а саме на роботу з асоціативною пам’яттю;
- занотувати процес розробки системи у текстовій записці магістерського дослідження;
- підготувати дані щодо навчання мовної системи;
- навчити розроблену систему на певних вхідних даних;
- порівняти дві отриманні нейронні мережі на однакових даних з метою виявлення доцільності використання саме асоціативного типу пам’яті у таких системах. Результати повинні бути представлені у графічному вигляді;
- обґрунтувати висновки зроблені з результату виконаного магістерського дослідження.

Після обирання засобів проведення, та розробки дослідницького завдання, перейдемо до теоретичної частини магістерського дослідження.

2 ДОСЛІДЖЕННЯ ТА АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

2.1 Аналіз сфери машинне навчання

Машинне навчання – це галузь науки, що вивчає розробку алгоритмів та статистичних моделей, які дозволяють комп'ютерним системам виконувати завдання без конкретних інструкцій. Замість цього вони спираються на аналіз шаблонів та логічних висновків. Алгоритми машинного навчання дозволяють комп'ютерним системам опрацьовувати великі обсяги статистичних даних і розпізнавати закономірності в них. Таким чином, системи можуть здійснювати більш точні прогнози на основі введених даних.

Окрім того машинне навчання - це підобласть штучного інтелекту, яка в широкому сенсі визначається як здатність машини імітувати розумну поведінку людини. Системи штучного інтелекту використовуються для виконання складних завдань аналогічно до того, як люди розв'язують проблеми.

Навчання машин починається з даних, які збирають і готують для використання як навчальні дані або інформацію, на якій модель машинного навчання буде навчатися. Розмір даних відіграє ключову роль, оскільки чим більше даних, тим ефективніше працює програма.

Потім розробники обирають модель машинного навчання, надають дані та дозволяють комп'ютерній моделі вивчати закономірності або робити прогнози. З плином часу людина-програміст також має можливість налаштовувати модель, включно зі зміною її параметрів для досягнення точніших результатів. Деякі дані з навчального набору зберігаються і використовуються як оціночні дані для перевірки точності моделі при введенні нових даних. У результаті формується модель, яку можна застосовувати з різними наборами даних.

Існує три основні категорії машинного навчання:

Моделі машинного навчання з вчителем використовують розмічені набори даних, що дає змогу моделям покращувати свою точність із плином часу. Наприклад, алгоритм може навчатися на зображеннях собак та інших об'єктах, позначених людьми, що зрештою дає змогу машині автоматично розпізнавати

зображення собак. Це є найпоширенішим типом машинного навчання, застосовуваним нині.

У випадку машинного навчання без вчителя програма шукає закономірності в нерозмічених даних, даючи змогу виявляти тенденції, які не є явними для людини. Наприклад, програма машинного навчання без вчителя може аналізувати дані онлайн-продажів і виявляти різні типи клієнтів, які здійснюють покупки.

Машинне навчання з підкріпленням - навчає машини шляхом проб і помилок з метою робити оптимальні дії на основі системи винагород. Наприклад, навчання з підкріпленням може використовуватися для навчання моделей грати в ігри або керувати автономними транспортними засобами. У цьому разі машина отримує зворотний зв'язок про правильність своїх дій, що допомагає їй з часом навчитися ухвалювати оптимальні рішення.

Машинне навчання тісно пов’язано з іншими сферами штучного інтелекту, такими як:

Обробка природної мови - це галузь машинного навчання, у якій машини вчяться розуміти природну мову, як говорять і пишуть люди, а не дані та числа, які зазвичай використовуються для програмування комп'ютерів. Це дає змогу машинам розпізнавати мову, розуміти її і реагувати на неї, а також створювати новий текст і перекладати між мовами.

Нейронні мережі - це широко використовуваний особливий клас алгоритмів машинного навчання. Штучні нейронні мережі змодельовані на людському мозку, в якому тисячі або мільйони вузлів обробки взаємопов'язані та організовані в шари.

2.2 Головні відомості про нейронні мережі

Нейронну мережу можна назвати програмою, що базується на принципі роботи головного мозку.

Нейронна мережа – різновид машинного навчання, у якому програма працює за принципом людського мозку. Ніхто на 100% не знає як саме працює мозок, але вважається що це самий приближений але спрощений варіант. Сама нейронна мережа складається з об’єднання нейронів – шарів. Шари бувають вхідні, сховані,

вихідні.

У свою чергу Нейронні мережі бувають:

Одношарова структура нейронної мережі - являє собою структуру взаємодії нейронів, у якій сигнали із вхідного шару одразу спрямовуються на вихідний шар, який, власне кажучи, не тільки перетворює сигнал, а й одразу ж видає відповідь.

Багатошарова нейронна мережа - крім вихідного і вхідного шарів, є ще кілька прихованих проміжних шарів. Число цих шарів залежить від ступеня складності нейронної мережі. Вона більшою мірою нагадує структуру біологічної нейронної мережі.

Вигляд простої нейронної мережі представлено на рисунку 2.1.

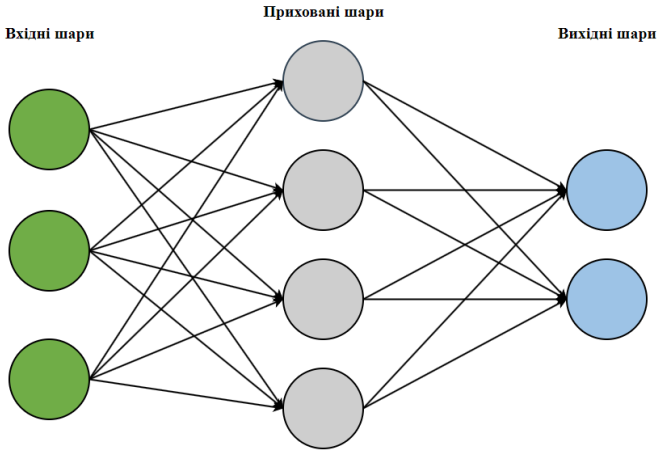

Рисунок 2.1 – Вигляд простої нейронної мережі (рисунок створено самостійно)

Крім кількості шарів, нейронні мережі можна класифікувати за напрямком розподілу інформації по синапсах між нейронами але спочатку треба розібрати що таке нейрони та синапси.

Нейрон – це одиниця, що виконує обчислення. Вона отримує дані з вхідного шару, виконуючи з нею прості обчислення, а потім передаючи наступному нейрону.

Синапс - зв'язок між нейронами, причому кожен синапс має свою вагу. Саме завдяки цьому вхідні дані видозмінюються під час передачі. Під час опрацювання передана синапсом інформація з великим показником ваги стане переважною.

Тобто на результат впливають не нейрони, а конкретно синапси, які дають

сукупність ваги вхідних даних, адже власне самі нейрони постійно виконують абсолютно однакові обчислення. Виставлення ваг здійснюється у випадковому порядку.

Існує велика кількість різних типів нейронних мереж, але вони у свою чергу також варіюються від типу задачі яку необхідно вирішити. Тобто нейронні мережі вміють працювати з різними типами даних, такими як зображення, звук, відео, текст тощо. Серед усього цього розмаїття типів нейронних мереж, ми вирішили дослідити асоціативний тип пам’яті саме в мовній моделі, через те що у цьому типі моделей легко перевірити асоціацію слів та контекст слів при генерації тексту.

2.3 Асоціативна пам’ять

Нейронна асоціативна пам'ять (НАП) - це нейромережеві моделі, що складаються з нейроноподібних та синапсоподібних елементів. У будь-який момент часу стан нейронної мережі задається вектором нейронної активності, який називається паттерном активності. Нейрони оновлюють значення своєї активності на основі вхідних даних, які вони отримують (через синапси). У найпростіших моделях нейронних мереж функція входу-виходу нейрона є функцією ідентичності або порогова операція. Треба зауважити, що в останньому випадку стан нейронної активності є бінарним: активний або неактивний. Інформація, яку обробляє нейронна мережа, представляється у вигляді шаблонів активності - паттернами активності (наприклад, представлення дерева може бути паттерном активності, де активні нейрони відображають зображення дерева). Таким чином, шаблони активності є представленнями елементів, що обробляються у мережі. Представлення називається розрідженим, якщо співвідношення між активними та неактивними нейронами невелике.

Синапси в нейронній мережі - це зв'язки між нейронами або між нейронами та волокнами, що несуть зовнішній вхід. Синапс передає пресинаптичну активність на постсинаптичну активність. У найпростіших моделях нейронних мереж кожен синапс має значення ваги і постсинаптичний сигнал є просто добутком пресинаптичної активності на вагу. Вхід нейрона - це сума постсинаптичних

сигналів усіх синапсів, що з'єднують нейрон. Таким чином, інформація обробляється в нейронній мережі шляхом поширення активності. Як поширюється, а отже, який алгоритм реалізується в мережі, залежить від того як формується синаптична структура, матриця синаптичних ваг в мережі шляхом навчання. У нейронній асоціативній пам'яті навчання забезпечує зберігання (великого) набору паттернів активності під час навчання, паттернів пам'яті.

Різні функції пам'яті визначаються тим, як можна отримати доступ до вивчених шаблонів вибірково звертатися до вхідного зразка. Функція розпізнавання зразків означає класифікування вхідних зразки на два класи: знайомі зразки та решту. Асоціація шаблонів описує функцію асоціювання певних вхідних зразків з певними зразками у пам'яті, тобто кожна пам'ять складається з пари вхідного і бажаного вихідного зразків.

Перевага нейронної асоціативної пам'яті над іншими алгоритмами зберігання шаблонів, наприклад, таблицями пошуку хеш-кодів, полягає в тому, що доступ до пам'яті може бути відмовостійким по відношенню до змін вхідного шаблону. Для асоціації шаблонів це означає, що вихідний шаблон може бути створений для набору вхідних шаблонів, які є (відносно такої метрики, як відстань Хеммінга) найближчі до вхідного шаблону, представленого під час навчання. Відмовостійкість - толерантність до помилок є ключовою у функції пошуку шаблонів за вмістом або завершення шаблону. Тут пари шаблонів, що використовуються під час навчання, повинні складатися з двох ідентичних шаблонів, що також називається автоасоціативним навчанням. У цьому випадку, якщо вхідні зразки є зашумлені версії вивчених шаблонів, асоціацію шаблонів можна розглядати як завершення шаблону, тобто зашумлена версія шаблону замінюється на безшумну версію шаблону.

Розрізняють два різні механізми пошуку, що відповідають архітектурам нейронної мережі з прямим або зворотним зв'язком:

При однокроковому пошуку кожен нейрон мережі оцінює вхідні дані лише один раз. При ітеративному пошуку активність протікає через зворотні зв'язки і нейрони оцінюють свої входи кілька разів протягом процесу пошуку.

Тренування або навчання в мережі означає, що паттерни нейронної активності можуть впливати на синаптичну структуру. Правила, що регулюють цей вплив, називаються правилами синаптичного навчання.

Найпростішою формою є правило локального навчання, коли зміна синаптичної ваги залежить тільки від пре- і постсинаптичної активності, тобто від сигналів, локально доступних у синапсі.

В асоціативній пам'яті може зберігатися багато асоціацій одночасно. Існують різні схеми накладання слідів пам'яті, сформованих різними асоціаціями. Суперпозиція може бути простим лінійним додаванням синаптичних змін необхідних для кожної асоціації (як у моделі Гопфілда) або нелінійним. Модель Віллшоу, наприклад, використовує відсікаючу суперпозицію (що призводить до бінарних ваг), яка є нелінійною. Існують різні моделі асоціативної пам'яті щодо процедури навчання. Вона може бути або одноразовою, коли кожен навчальний зразок подається один раз, або рекурентним. У рекурентному навчанні перевіряється пошук шаблону, і шаблони пред'являються кілька разів, поки не буде досягнутий бажаний рівень якості пошуку.

Продуктивність нейронної асоціативної пам'яті зазвичай вимірюється величиною що називається інформаційною ємністю, тобто інформаційним вмістом, який може бути вивчений і витягти, поділеним на кількість необхідних синапсів.

2.4 Архітектури пам’яті нейронних мереж

Існує велика кількість типів пам’яті нейронних мереж, але треба помітити, що пам’ять в нейронних мережах, це не завжди окремий шар, це може бути навіть інша архітектура або інший принцип роботи нейронної мережі. Тобто пам’ять в розумінні машинного навчання, це не те саме що і людська пам’ять. Нижче представлено декілька типів архітектур пам’яті нейронних мереж.

2.4.1 RNN

Базова рекурентна нейронна мережа (RNN) - це тип нейронної мережі, призначений для обробки послідовних даних. На відміну від традиційних нейронних мереж прямого поширення, RNN мають зв'язки, які утворюють спрямовані цикли, що дозволяє їм зберігати прихований стан, який фіксує інформацію про попередні входи в послідовності. Це робить RNN добре пристосованими до завдань, пов'язаних з послідовними або часовими рядами даних. RNN зображено на рисунку 2.2.

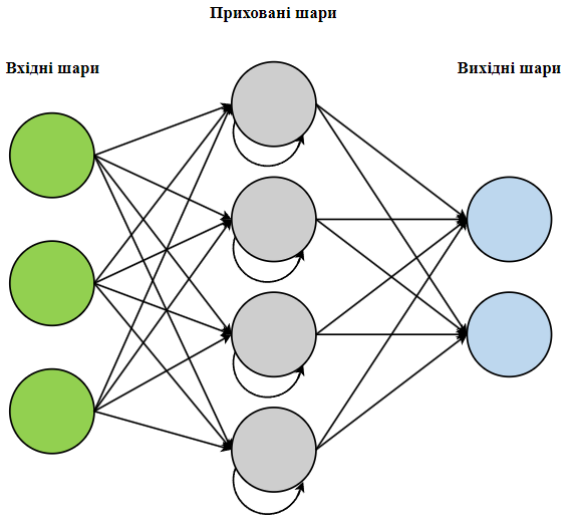

Рисунок 2.2 – Вигляд рекурентної нейронної мережі (рисунок створено самостійно)

Архітектура базового RNN складається з наступних компонентів:

Вхідний шар - представляє вхідні характеристики для кожного часового кроку в послідовності.

Рекурентний зв'язок. Ключовою особливістю RNN є рекурентний зв'язок, який дозволяє інформації зберігатися на різних часових кроках. На кожному часовому кроці прихований стан з попереднього часового кроку використовується в поєднанні з поточним входом для отримання виходу та оновлення прихованого стану.

Прихований стан - містить інформацію про попередні входи в послідовності. Він оновлюється на кожному часовому кроці на основі поточного входу та

попереднього прихованого стану.

Вихідний шар - виробляє вихід для поточного кроку часу на основі поточного входу та прихованого стану.

2.4.2 Трансформер

Трансформер – це система для обробки текстових послідовностей, таких як у завданнях машинного перекладу та автоматичного реферування. Основна його особливість – механізм уваги, який виявляє взаємозв'язки між частинами вхідних і вихідних даних.

Механізм уваги - ключова частина трансформеру, яка дозволяє моделі фокусуватися на різних частинах вхідних даних під час обробки послідовностей. Замість послідовної обробки вхідних даних, трансформер одночасно враховує всі елементи послідовності. Механізм уваги дозволяє моделі надавати вагу кожному елементу вхідних даних в залежності від їхньої важливості для завдання. Це досягається за допомогою вагових коефіцієнтів, які обчислюються на основі схожості між поточним елементом і всіма іншими елементами вхідної послідовності.

Модель трансформер базується на концепції самоуваги, що дозволяє моделі зважувати важливість різних слів у реченні під час генерування результату.

Далі представлено принцип роботи моделі.

Вбудовування вхідних даних.

Трансформер перетворює вхідний текст у вектори за допомогою шару вбудовування. На відміну від традиційних методів, які використовують вбудовування слів, трансформер використовує вивчені вбудовування. Щоб вхідні вставки не були надто малими, вони множаться на квадратний корінь з розміру вставки.

Вхідні дані являють собою послідовність токенів, кожен з яких представлений у вигляді однозначного вектора. Ці вектори згодом множаться на матрицю вбудовування для отримання вхідних вбудовувань. Ця матриця вбудовування є параметром, що вивчається в процесі навчання.

Позиційне кодування.

Трансформер не має рекурентності або згортки, тобто він не має жодного відчуття позиції або порядку слів. Щоб подолати це, до вхідних вкладень додаються позиційні кодування. Це вектори, які кодують положення слів у реченні і додаються до вхідних введень.

Позиційне кодування додається до вхідних вбудовувань, щоб дати моделі відчуття порядку слів.

Позиційні кодування додаються до вхідних вбудовувань, щоб отримати кінцевий вхід для трансформеру

Вони додаються до вхідних введень внизу стеків кодера та декодера.

Енкодер.

Енкодер показано на зображенні 2.3.

Ліва частина діаграми архітектури трансформера представляє енкодер. Він складається зі стеку однакових шарів. Кожен шар має два підшари: механізми самоуваги з кількома головками та повністю з'єднану за позицією мережу прямого зв'язку. Навколо кожного з двох підшарів є залишковий зв'язок, після чого відбувається нормалізація шарів.

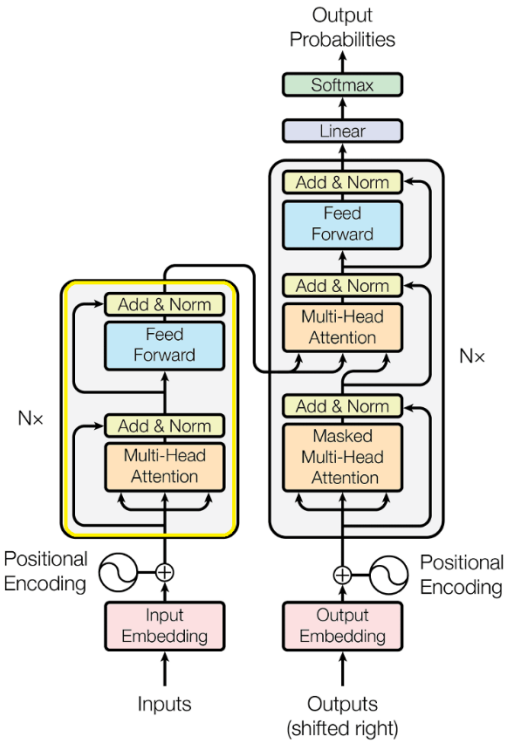

Рисунок 2.3. Архітектура трансформеру (за даними [1])

Багатоголова увага

Основна ідея моделі "Трансформер" полягає в механізмі уваги, який зважує важливість різних слів при створенні вихідного слова. У багатоголовій увазі вхідні дані розбиваються на кілька голів, і до кожної з них застосовується масштабована точкова увага. Потім виходи всіх голів об'єднуються і лінійно трансформуються, щоб отримати вихідний результат багатоголової уваги.

Це відбувається шляхом застосування механізму уваги кілька разів паралельно, звідси і термін "багатоголова увага".

Нейронні мережі прямого поширення

Після багатоголової уваги вихідні дані проходять через нейронну мережу прямого поширення, яка складається з двох лінійних перетворень з активацією ReLU між ними.

Нормалізація та залишкові зв'язки

Після кожного випадку багатоголової уваги та нейронної мережі прямого поширення, тобто після кожного підшару як в енкодері, так і в декодері, залишається залишковий зв'язок, за яким слідує нормалізація шару. Залишковий зв'язок допомагає уникнути проблеми зникаючого градієнта, а нормалізація шарів сприяє швидшому і стабільнішому навчанню.

Декодер.

Декодер представлено на рисунку 2.3.

Права частина діаграми архітектури трансформера представляє декодер. Він також складається зі стеку однакових шарів. На додаток до двох підрівнів у кожному шарі енкодеру, декодер вставляє третій підрівень, який виконує багатоголовочну обробку вихідних даних стека енкодера. Подібно до енкодера, навколо кожного з підшарів є залишкові зв'язки, після чого відбувається нормалізація шару.

Вихід.

Вихід декодера проектується на прогнозовані ймовірності на множині всіх можливих виходів за допомогою кінцевого лінійного шару з подальшим застосуванням функції softmax. Цей вихід можна використовувати різними

способами залежно від завдання, наприклад, подати його на лінійний шар з наступним softmax для задач класифікації.

2.4.3 LSTM memory

Long Short Term Memory - це різновид рекурентної нейронної мережі. У RNN вихід з останнього кроку подається на вхід поточного кроку. LSTM за замовчуванням може зберігати інформацію протягом тривалого часу. Вона використовується для обробки, прогнозування та класифікації на основі даних часових рядів. LSTM показано на рисунку 2.4.

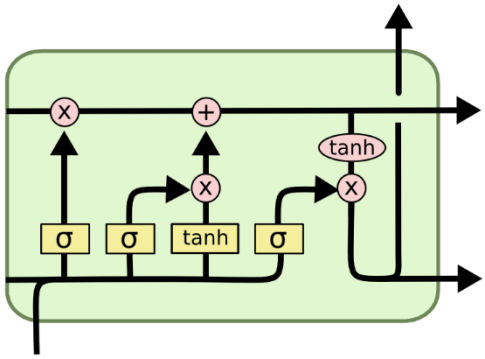

Рисунок 2.4. Архітектура LSTM (за даними [2])

Вона спеціально розроблена для роботи з послідовними даними, такими як тимчасові ряди, мова і текст. LSTM-мережі здатні вивчати довгострокові залежності в послідовних даних, що робить їх добре придатними для таких завдань, як переклад мови, розпізнавання мови і прогнозування часових рядів.

У LSTM присутній осередок пам'яті, який являє собою контейнер, здатний зберігати інформацію протягом тривалого часу. Осередок пам'яті керується трьома затворами: вхідним, таким, що забуває, і вихідним. Ці затвори вирішують, яку інформацію додавати в комірку пам'яті, видаляти з неї і виводити.

Вхідні ворота керують додаванням інформації в комірку пам'яті. Затвор забування керує тим, яка інформація видаляється з комірки пам'яті. А вихідний гейт керує тим, яка інформація виводиться з комірки пам'яті. Це дає змогу LSTM-мережам вибірково зберігати або відкидати інформацію в міру її проходження через мережу, що дає їм змогу навчатися довгострокових залежностей.

LSTM має ланцюгову структуру, яка містить чотири нейронні мережі та різні блоки пам'яті, які називаються комірками.

Інформація зберігається в комірках, а маніпуляції з пам'яттю здійснюються за допомогою воріт. Є три вентилі:

Ворота забуття: інформація, яка більше не є корисною у стані комірки, видаляється за допомогою воріт забуття. Два входи подаються на вентиль і перемножуються з ваговими матрицями з подальшим додаванням зсуву. Результат проходить через функцію активації, яка дає двійковий вихід. Якщо для певного стану комірки вихід дорівнює 0, то інформація забувається, а якщо вихід 1, то інформація зберігається для подальшого використання.

Вхідний вентиль: Додавання корисної інформації до стану комірки здійснюється вхідним вентилем. Спочатку інформація регулюється за допомогою сигмоїдної функції та фільтрує значення, які потрібно запам'ятати, подібно до воріт забування, використовуючи входи. Потім за допомогою функції tanh створюється вектор, що дає вихід від -1 до +1, який містить всі можливі значення з обох входів. Нарешті, значення вектора і регульованих значень перемножуються для отримання корисної інформації.

Вихідний вентиль: Завдання вилучення корисної інформації з поточного стану комірки для представлення на виході виконує вихідний вентиль. Спочатку генерується вектор шляхом застосування функції тангенса до комірки. Потім інформація регулюється за допомогою сигмоїдної функції та фільтрується за значеннями, які потрібно запам'ятати, за допомогою входів. Нарешті, значення вектора і регульованих значень перемножуються, щоб бути відправленими як вихід і вхід до наступної комірки.

Існує декілька модифікацій LSTM, наприклад такі, як:

Peephole LSTM (Long Short-Term Memory) - це модифікація стандартної LSTM-мережі, яка містить додаткові з'єднання між прихованим станом і вхідним/вихідним вентилями. Ці додаткові з'єднання дають змогу LSTM-мережі "заглядати" у вхідні та приховані стани під час ухвалення рішень про те, яку інформацію зберігати, забувати або передавати на наступний крок.

По суті, Peephole LSTM розширює можливості звичайної LSTM, враховуючи додаткові контекстні чинники під час обробки послідовності даних. Це може допомогти мережі краще вловлювати довгострокові залежності в даних і робити більш точні прогнози або класифікації в завданнях обробки послідовностей.

Bidirectional LSTM (Long Short-Term Memory) - це розширення звичайної LSTM мережі, яка здатна обробляти послідовності даних в обох напрямках: від початку до кінця (пряме) і від кінця до початку (зворотне).

Основна ідея полягає в тому, щоб захопити контекст як з минулого, так і з майбутнього для кожного елемента послідовності. Це дає змогу більш повно врахувати залежності в даних і краще розуміти контекст у кожній точці послідовності.

2.5 Порівняння трансформерів та моделей LSTM

Трансформери і моделі LSTM являють собою два різні підходи до побудови рекурентних нейронних мереж, які використовуються в обробці природної мови та інших завданнях. Їх можна порівняти за такими параметрами:

а) архітектура:
   1) LSTM - належить до типу рекурентних нейронних мереж (RNN), у яких використовують комірки з довгостроковою і короткостроковою пам'яттю для врахування залежностей у послідовних даних;
   2) трансформери - не використовують рекурентні зв'язки, натомість вони оперують із механізмом уваги (attention) і дають змогу моделі паралельно обробляти вхідні дані;

б) обробка послідовностей:
   1) LSTM - призначені для опрацювання послідовних даних, де кожен елемент послідовності обробляється по черзі;
   2) трансформери можуть обробляти вхідні дані паралельно завдяки механізму уваги, що робить їх більш ефективними під час обробки довгих послідовностей;

в) залежність від позиції:

1) LSTM не мають вбудованої структури для роботи з інформацією про позицію елементів у послідовності;
2) трансформери використовують додавання ембеддінгів позицій, що дозволяє їм враховувати порядок елементів у послідовності;

г) навчання:

1) LSTM можуть бути складнішими для навчання на довгих послідовностях через проблему затухаючих або вибухаючих градієнтів;
2) трансформери більше піддаються паралельному навчанню і можуть простіше навчатися на великих обсягах даних;

д) застосування:

1) LSTM - широко застосовуються в задачах обробки природної мови, передбаченні часових рядів та інших завданнях, де важливе оброблення послідовних даних;
2) трансформери стали основним інструментом у задачах машинного перекладу, генерації тексту та інших завданнях, де паралельне опрацювання даних може бути вигідним.

У нашому дослідженні ми плануємо об’єднати дві дані архітектури в одну, для перевірки доцільності таких дій. Теоретично дані дії можуть покращити генерацію та асоціацію слів у генеративних системах.

3 ПІДГОТОВКА ДО ПРОВЕДЕННЯ ДОСЛІДЖЕННЯ

3.1 Підготовка даних до проведення експерименту

Після етапу обирання даних на яких будуть навчатися нейронні мережі, ми перейшли саме до підготовки цих даних.

Далі для підготовки даних ми виконуємо наступні дії:

– після завантаження сирих даних, ми видаляємо шум, тобто прибраємо розділові знаки, перетворюємо великі літери у маленькі, виконуємо усі необхідні дії щодо редагування та приведення тексту до однорідного вигляду;
– отриманні речення перетворюємо на N-грами – послідовність слів з обмеженням N;
– далі N-грами перетворюємо у токіни – закодовані слова;
– нормалізуємо розмір отриманих масивів які складаються із токенів.

На даному етапі, данні готові для подальшого їх використання у навчанні моделей.

Після написання даних етапів, складається враження що даний процес є простим, але на справді, він є одним з найважливіших й складніших етапів, від якого заложить якість отриманих моделей.

3.2 Вибір моделей нейронних мереж

Для того щоб виконати саме дослідження текстових, генеративних систем, спочатку необхідна сама, така система, що буде досліджуватись.

Для виконання поставленої мети відразу зрозуміло що необхідні текстові (мовні) моделі нейронних мереж, тому що ми будемо працювати саме з асоціацією у тексті.

З усіх існуючих текстових моделей нейронних мереж, було обрано наступні п’ять. Короткий опис про кожну з п’яти моделей нейронних мереж:

GPT 1 - (Generative Pre-trained Transformer 1) - це одна з перших моделей створена на основі архітектури "Transformer". Вона була розроблена для обробки

природної мови та здатна генерувати текст. Сама модель складалася з 12 шарів, була навчена на 7000 книг, а максимальний розмір контексту – 512 токенів. Ця модель є прототипом, та використовувалася як приклад того, що можуть сучасні технології.

GPT-2 (Generative Pre-trained Transformer 2) – це друга модель у серії GPT. Основна її відмінність від першої моделі – це її розширення, вона стала набагато більше. Дану модель навчили на більш великому обсязі даних — до книг додали 8 мільйонів сайтів. Сумарно вийшло 40 гігабайт тексту. Тепер дана модель має 48 шарів та приблизно 1.5 мільярдів параметрів. Архітектурно модель змінилася не сильно - лише трохи перемістили верстви нормалізації. Максимальний розмір контексту у GPT-2 – 1024 токенів. Дана модель є OpenSource, тобто її можна легко змінити за потребами користувача.

GPT-3 (Generative Pre-trained Transformer 2) - це третя модель у серії GPT. Ця модель стала ще більш великою відносно попередніх двох. Її навчили на ще більшій кількості даних - 570GB тексту. Тепер вона має 175 мільярдів параметрів, та максимальний розмір контексту у GPT-3 став 2048 токенів. Архітектурно дана модель майже не змінилась, лише відбулась невелика оптимізація моделі. Дана модель є Partial Open Source.

LLaMA 13B – це невелика текстова модель у порівнянні з попередніми, перевага якої не в її розмірі, а в інших цікавих функціях. Наприклад вона може виконувати сумаризацію тексту, за допомогою чого можна зробити коротку витримку з великого об’єму тексту. Кількість параметрів даної моделі написано у її назві, тобто 13 мільярдів. У даній моделі 40 шарів та максимальний розмір контексту – 40 токенів. Дана модель є Open Source.

LLaMA 65B – дана модель така сама як й LLaMA 13B, але відрізняється більшої кількістю параметрів – 65 мільярдів, кількістю шарів – 80, та максимальний розмір контексту – 64 токенів. Дана модель також є Open Source.

Завчасно треба помітити що нейронні мережі, описані вище, у своєму початковому варіанті не підходять для реалізації теми магістерського дослідження, тому буде змінено їх архітектуру та вибірку, саме для перевірки асоціативної

пам’яті. Однак для того щоб виконати нашу мету, нам необхідно обрати ту архітектуру нейронної мережі, що буде легше переробити під наші потреби.

Виконавши пошук інформації щодо моделей нейронних мереж, відмітимо за якими критеріями ми їх обираємо:

- Розмір вибірки – кількість параметрів нейронної мережі;
- Розмір нейронної мережі – це загальна кількість слоїв нейронної мережі;
- Розмір вхідного токену – кількість токенів що можна подати на вхід нейронної мережі;
- Вид ліцензії – яку ліцензію має модель нейронної мережі, чи можна її змінювати;
- Можливість навчання – можливість переробити нейронну мережу за потребами.

Тепер обравши критерії вибору, додаймо їх до таблиці критеріїв відносно кожної моделі нейронних мереж.

Таблиця 3.1 – Значення критеріїв вибору моделей нейронних мереж

	Розмір вибірки	Розмір нейронної мережі	Розмір вхідного токену	Вид ліцензії	Можливість навчання
GPT 1	0,117 Billion	12	512	Прототип	Ні
GPT 2	1.5 Billion	48	1024	Open Source	Так
GPT 3	175 Billion	96	2048	Partial Open Source	Ні
LLaMA 13B	13 Billion	40	40	Open Source	Так
LLaMA 65B	65 Billion	80	64	Open Source	Так

Наступний крок – описання критеріїв вибору моделі нейронної мережі:

Перші три критерії вибору йдуть за абсолютною шкалою, тому змінювати їх немає необхідності.

Розмір вибірки – для реалізації мети магістерського дослідження нам необхідна велика кількість параметрів. Однак цей критерій не самий важливий, через те що буде змінено архітектуру нейронної мережі. Тому даний критерій має

коефіцієнт 0,1.

Розмір нейронної мережі – для реалізації теми нам необхідна велика кількість нейронів, які в свою чергу залежать від кількості шарів. Даний критерій також не самий важливий, через те що буде змінено архітектуру нейронної мережі. Тому даний критерій має коефіцієнт 0,1.

Розмір вхідного токену – щоб перевірити асоціативну пам'ять на більшій кількості даних, нам необхідна саме більша кількість самих даних, тобто вхідних токенів. Даний критерій більш важливий ніж попередні два, через те що ми будемо досліджувати нейронну мережу на великій кількості вхідної інформації. Тому даний критерій має коефіцієнт 0,2.

Наступні значення критерію представлені у вигляду тексту, тому оцінимо їх по шкалі від 0 до 5.

Вид ліцензії - щоб змінити архітектуру моделі нейронної мережі, необхідно мати доступ до самого коду, тому:

- прототип – 2;
- partial Open Source або – 3;
- open Source – 5.

Даний критерій важливий тому що ми будемо змінювати нейронну мережу, й не хотілось би мати проблеми через порушення ліцензії. Тому даний критерій має коефіцієнт 0,2.

Наступні значення критерію представлені у вигляду тексту, тільки два слова, Так та Ні, тому оцінимо їх по шкалі від 0 до 1.

Можливість навчання – необхідна така нейронна мережа, яку можна легко змінити під дослідження асоціативної пам'яті у генеративних системах. Тому:

- так – 1;
- ні – 0.

Даний критерій самий важливий тому що ми будемо змінювати нейронну мережу, а для цього необхідна сама можливість зміни архітектури нейронної мережі. Тому даний критерій має коефіцієнт 0,4.

За правилом Парето ми відразу бачимо що перша модель нейронної мережі

слабка по усім параметрам, тому її можна відразу викреслити з розрахунку.

Описавши шкали оцінок за критеріями та коефіцієнтами, тепер ми можемо знайти яка з цих моделей нам підходить більше усього. Для цього використовуємо лінійну адитивну згортку з ваговими коефіцієнтами, тому що деякі з окремих критеріїв є більш важливі за інші. Нижче представлено саму формулу (3.1):

$$Z^* = \max_{i=1,m} \sum_{j=1}^{n} \alpha_j \beta_j a_{ij} \quad (3.1)$$

де α_j–нормуруючі множники,

β_j – вагові коефіцієнти.

За правилом Парето ми відразу бачимо що GPT-1 слабка по усім параметрам, тому її можна відразу викреслити з розрахунку.

Розрахунок даної формули було виконано у середовищі Excel.

Отримані дані після розрахунку та перетворені критерії додаємо до таблиці 2.

Таблиця 3.2 – Критерії вибору та отримані дані

	Розмір вибірки	Розмір нейронної мережі	Розмір вхідного токену	Ліцензія	Можливість навчання	Z*
GPT 2	1,500	48	1024	5	1	0,272
GPT 3	175,000	96	2048	3	0	0,267
LLaMA 13B	13,000	40	40	5	1	0,212
LLaMA 65B	65,000	80	64	5	1	0,249

Тобто у підсумку, з усіх моделей нейронних мереж для виконання дослідження асоціативної пам’яті у генеративних системах, підходить GPT-2 тому що має значення 0.272, що більше за інші. Тому доцільно використовувати саме цю текстову модель. Однак треба зазначити, що у нашому випадку розмір даної моделі буде 117 мільйонів параметрів.

3.3 Бачення реалізації задач дослідження

У результаті ми отримаємо дві системи: одна система без архітектурних змін, тобто має свій звичайний вид GPT-2, а друга система – зі зміненим типом пам'яті, тобто окрім основної моделі GPT-2 має додаткову мультимодальну "голову" входу.

Вдосконалена система буде мати такі додаткові шари:

Якщо описувати алгоритм, то: Спочатку дані обробляються системою пам’яті, далі дані стискаються за подібним змістом. На наступному кроці двонаправлена LSTM виконує роботу з даними (прогнозування послідовності, класифікація даних, розпізнавання іменованих даних, позначення частин мови), а на останньому кроці – вибираються важливі данні.

У результаті до двох розроблених систем ми будемо ставити однакові запитання (а також N - грамми в процесі навчання). У трансформерів є ймовірність наступного слова: якщо текст буде написаний з більшим значенням ймовірності, то він вважається більш правильним також, якщо метрика точності буде високою. Тобто дві моделі дають схожу відповідь, але там, де точність відповіді більше, то там і краще текст. Окрім того рахується її відповідність до даних, на яких система навчалася.

Таким на момент написання тез ми бачили результат нашого дослідження, завдяки чому зможемо дослідити використання саме асоціативної пам’яті у генеративних системах.

4 ПРОЦЕС РОЗРОБКИ СИСТЕМИ

4.1 Структура проекту

Після обирання системи що буде модифікуватись, ми приступили до реалізації нашої ідеї. Як було розписано раніше, проект являє собою чотири моделі, дві з яких – це gpt-2, а інші дві - це асоціативна пам’ять та допоміжна модель.

Сам проект складається з кількох основних та допоміжних файлів, але розписувати код та принцип роботи усіх, не є доцільним. Серед основних файлів, можна виділити та більш розгорнуто написати про наступні: main, gpt-2, model, memory, visualizator.

Коротка інформація про дані файли:

Файл через який виконується запуск самого проекту – це main. У ньому виконується запуск, можливе навчання, дослідження та візуалізація моделей. Сам файл не є основною моделлю, це лише налаштування та запуск системи.

Gpt-2 – основний виконуючий файл. У ньому відбуваються запити до моделей, звертання до основної математики моделей, контроль файлів, тобто він є головним, виконуючим елементом даної системи.

Model – файл у якому відбувається основні розрахунки моделей, тобто він отримує дані з інших файлів та за патерном виконує різні розрахунки що необхідні для системи.

Memory – моделі асоціативної пам'яті та зв’язку з трансформером. Даний файл необхідний саме для тої моделі що використовує асоціативну пам’ять. Модель без такої пам’яті звертається тільки до трансформеру, не чіпляючи даний тип пам’яті.

Visualizator – у даному файлі виконується саме порівняння моделей. Він використовується для відображення графіків моделей та дослідження доцільності використання асоціативної пам’яті.

Після короткого розгляду, можна перейти до розгорнутого опису системи, опираючись на її код у хронологічному порядку.

4.1 Розробка основної моделі трансформеру

Розробка даної системи почалась з обирання основних параметрів моделі, що частково було розписано у розділі 3.2

За основу було узято з відкритих джерел, код трансформеру GPT-2 [3], [4] на 117 мільйонів параметрів. Даний трансформер використовується без архітектурних змін у двох варіантах моделей, у той що з асоціативною пам’яттю та той що без.

Код даного трансформеру знаходиться у файлі model, так як архітектурно він не має змін порівняно оригіналу, то розписувати його код построково не є доцільним. Лише можна описати його принцип роботи та важливі аспекти.

У нашому проекті він працює так само як і інші трансформери, принцип яких було розписано раніше, однак можна виділити:

- можливість редагувати ваги нейронів;
- змінений параметр top_k з 40 на 20. Даний параметр змінює кількість слів що буде обрано для генерації;
- можливість змінити контекстні токіни для асоціативної пам’яті;
- інші невеликі зміни, що спрощують розробку.

Дана частина є головною у даному проекті, тому що саме вона виконує усі основні математичні дії. Більш розгорнуто про принцип роботи трансформеру, розписано у пункті 2.4.2.

Треба помітити, що перед початком розробки наступних частин системи, на даному етапі виконувався тест працездатності трансформеру. Було декілька разів виконано генерацію тексту.

4.2 Принцип роботи Fine Tuning (Перенавчання)

Виходячи з попередніх пунктів даної кваліфікаційної роботи - модель у машинному навчанні - це величезне математичне рівняння з великою кількістю параметрів, яке може передбачити щось на основі однієї або декількох вхідних

змінних. Тренування моделі – це пошук правильних параметрів, щоб модель, давала вірну відповідь.

Навчити модель із двома параметрами просто. Але тренувати модель із мільярдами параметрів – це завдання, яке вимагає серйозної обчислювальної потужності.

Існує багато варіантів щодо навчання таких моделей, наприклад навчання з нуля, але це буде дуже довго. Тому можна використовувати процес Fine-tuning.

Fine-tuning - це процес налаштування або доопрацювання моделі машинного навчання, яка вже була попередньо навчена на великому наборі даних, щоб адаптувати її до конкретного завдання або набору даних. Це часто використовується в глибокому навчанні, особливо коли вже є модель, яка була навчена на великому обсязі даних, і необхідно налаштувати її для виконання конкретного завдання, яке може бути пов'язане з цими даними. Такий процес може включати зміну гіперпараметрів моделі, таких як швидкість навчання, архітектура моделі або розмір пакета, а також донавчання моделі на нових даних для кращої адаптації.

У нашому випадку Fine-tunning використовувався для того щоб помістити обидві розроблені системи в однакові умови, інакше, одна система, може працювати краще ніж інша. Код даної функції знаходиться у файлі GPT-2 та має наступні аргументи що показано на рисунку 4.1.

```
def finetune(sess,
             dataset,
             steps=-1,
             model_name='124M',
             model_dir='models',
             combine=50000,
             batch_size=1,
             learning_rate=0.0001,
             accumulate_gradients=5,
             restore_from='latest',
             run_name='run1',
             checkpoint_dir='checkpoint',
             sample_every=100,
             sample_length=1023,
             sample_num=1,
             multi_gpu=False,
             save_every=1000,
             print_every=1,
             max_checkpoints=1,
             use_memory_saving_gradients=False,
             only_train_transformer_layers=False,
             optimizer='adam',
             overwrite=False,
             reuse=False,
             asm = False,
             asm_train=False,
             asm_dim=256):
```

Рисунок 4.1 – Аргументи Fine Tuning (рисунок створено самостійно)

Опираючись на рисунок 2.2, опишемо лише основні й найбільш цікаві аргументи:

- sess – виконуюча функція що використовується як компілятор моделі;
- dataset – дані які подаються для перенавчання моделей;
- steps=-1 – зняття обмеження на кроки перенавчання;
- batch_size=1 – значення на скільки фрагментів було розділено дата сет. Один фрагмент дата сету у нашому випадку, тому що дозволяють ресурси – оперативна пам’ять;
- learning_rate=0.0001 – швидкість перенавчання. Чим вище цей параметр тим швидше але менш якісніше вийде модель. У нашому випадку ми обрали оптимальне значення у 0,1% (у процесі розрахунку значення множиться на 100);
- accumulate_gradients=5 – за суттю теж саме що й learning_rate, але відносно накопичування градієнту;
- checkpoint_dir='checkpoint' – зберігає моделі у дану папку. Це важливо виділити, тому що у інших файлах, буде виконуватись звертання до даного місця;
- sample_every=100 – виконування тесту моделей кожні 100 епох;
- sample_length=1023 – розмір тестового тексту для моделей;
- multi_gpu=False – дозвіл на використання декількох відеокарт для виконання навчання;
- save_every=1000 – виконання зберігання моделі кожні 1000 епох;
- only_train_transformer_layers=False – дозвіл на використання всіх нейронів моделі для перенавчання;
- optimizer='adam' – використання оптимізатору «adam» для найкращого навчання моделі;
- reuse=False – так як модель дуже багато важить, то даний параметр регулює чи можна використовувати цю ж саму модель кілька разів за сесію. У нашому випадку це заборонено;

– asm=False – параметр який визначає використання асоціативної пам’яті. Даний аргумент регулюється через файл main для кожної моделі окремо.
– asm_train=False – параметр який визначає навчання асоціативної пам’яті. Даний аргумент також регулюється через файл main для кожної моделі окремо.
– asm_dim=256 – розмірність входу та виходу асоціативної пам’яті.

Розписування коду перенавчання через його типову структуру не має сенсу, тому необхідно виділити лише основні функції, підфункції що притаманні нашій системі. Дані підфункції показано на рисунку 4.2.

```
246     def save():
247         maketree(checkpoint_path)
248         print(
249             'Saving',
250             os.path.join(checkpoint_path,
251                          'model-{}').format(counter-1))
252         saver.save(
253             sess,
254             os.path.join(checkpoint_path, 'model'),
255             global_step=counter-1)
256         with open(counter_path, 'w') as fp:
257             fp.write(str(counter-1) + '\n')
258 
259     def generate_samples():
260         context_tokens = data_sampler.sample(1)
261         all_text = []
262         index = 0
263         while index < sample_num:
264             out = sess.run(
265                 tf_sample,
266                 feed_dict={context: batch_size * [context_tokens]})
267             for i in range(min(sample_num - index, batch_size)):
268                 text = enc.decode(out[i])
269                 text = '======== SAMPLE {} ========\n{}\n'.format(
270                     *args: index + 1, text)
271                 all_text.append(text)
272                 index += 1
273         maketree(os.path.join(SAMPLE_DIR, run_name))
274         with codecs.open(
275                 os.path.join(SAMPLE_DIR, run_name,
276                              'samples-{}').format(counter), mode: 'w', encoding: 'utf8') as fp:
277             fp.write('\n'.join(all_text))
278 
```

Рисунок 4.2 – Основні підфункції обробки даних (рисунок створено самостійно)

На рисунку 4.2 виділено дві підфункції – функція Save та функція generate_sampless.

Підфункція Save виконує збереження моделі GPT.

Підфункція generate_sampless – створення тесту моделі.

```
avg_loss = (0.0, 0.0)
start_time = time.time()

if asm:
    if asm_train:
        ASGM.compile()
        ds = []
        for i in range(len(chunks[0])//asm_dim):
            if i % 32 == 0:
                ds.append(chunks[0][i*asm_dim:(i+1)*asm_dim])
        #delete duplicate lists
        ds = [list(item) for item in set(tuple(row) for row in ds)]
        ds = np.array(ds)
        h1, h2 = ASGM.train(ds, epochs=steps, batch_size=batch_size)
        h1_norm = model.softmax(h1.history['loss'])
        h2_norm = model.softmax(h2.history['loss'])

        ASGM.save()
    else:
        ASGM.load()
if steps:
    steps = int(steps)

try:
    while True:
        if steps > 0 and counter == (counter_base + steps):
            save()
            return
        if (counter - 1) % save_every == 0 and counter > 1:
            save()
        if (counter - 1) % sample_every == 0 and counter > 1:
            generate_samples()

        cs = sample_batch()
        if accumulate_gradients > 1:
            sess.run(opt_reset)
            for _ in range(accumulate_gradients):
                if asm:
                    for c in ASGM.associate(cs)[0]:
                        np.insert(cs[0], obj: 0, c)
                    sess.run(
                    opt_compute, feed_dict={context: cs})
                else:
                    sess.run(opt_compute, feed_dict={context: cs})
                (v_loss, v_summary) = sess.run((opt_apply, summary_loss))
        else:
            if asm:
                for c in ASGM.associate(cs)[0]:
                    np.insert(cs[0], obj: 0, c)
                (_, v_loss, v_summary) = sess.run(
                (opt_apply, loss, summary_loss),
            feed_dict={context: cs})
            else:
                (_, v_loss, v_summary) = sess.run(
                (opt_apply, loss, summary_loss),
                feed_dict={context: cs})

        summary_log.add_summary(v_summary, counter)

        if counter % print_every == 0:
            avg_loss = (avg_loss[0] * 0.99 + v_loss,
                        avg_loss[1] * 0.99 + 1.0)

            print(
                '[epoch={counter} | {time:2.2f}m] loss={loss:2.2f} avg={avg:2.2f}'
                .format(
                    counter=counter,
                    time=(time.time() - start_time)/60,
                    loss=v_loss,
                    avg=avg_loss[0] / avg_loss[1]))

        counter += 1
except KeyboardInterrupt:
    print('interrupted')
    save()
```

Рисунок 4.3 – Основна частина коду Fine Tuning (рисунок створено самостійно)

На рисунку 4.3 зображено основну функцію Fine Tuning, завдяки якої і виконується перенавчання системи.

Спочатку отримується і обробляється датасет [5], далі визначається тип моделі (з асоціативною пам’яттю чи без), і потім у циклі виконується навчання з оптимізатором. Це увесь високорівневий процес який виконується для перенавчання наших моделей.

Навчання обох систем виконувалось на віддаленому сервері з даними властивостями:

– Процесор: xeon e5 2667 - 2 одиниці;
– Графічні карти: quadro p6000 – 4 одиниці;
– Оперативна пам’ять: 128 gb ddr4 – 4 планки по 32 гігабайта.

Після даного етапу, можна перейти до розгляду асоціативної пам’яті у нашій системі.

4.3 Робота асоціативної пам’яті

На даному етапі, одна з систем генерації тексту вже готова, наступний крок – створення модулю асоціативної пам’яті для другої моделі.

Про саму асоціативну пам’ять та її принцип роботи вже було розписано у розділі 2.3, тому відразу перейдемо до розпису асоціативної пам’яті саме у нашому проекті.

Асоціативна пам’ять складається з двох моделей: основна пам'ять - центральний склад інформації, який використовується моделлю під час роботи. Додаткова – для поєднання трансформеру та пам’яті - використовується для ефективного поєднання архітектури з основною пам'яттю. Вона допомагає зберігати важливі зв'язки між різними частинами інформації в нейронній мережі.

Опираючись на інформацію, що було розписано у розділі 3.3, ми почали розробку даної пам’яті у нашій системі.

На рисунку 4.4 показано шари асоціативної пам’яті нашої моделі.

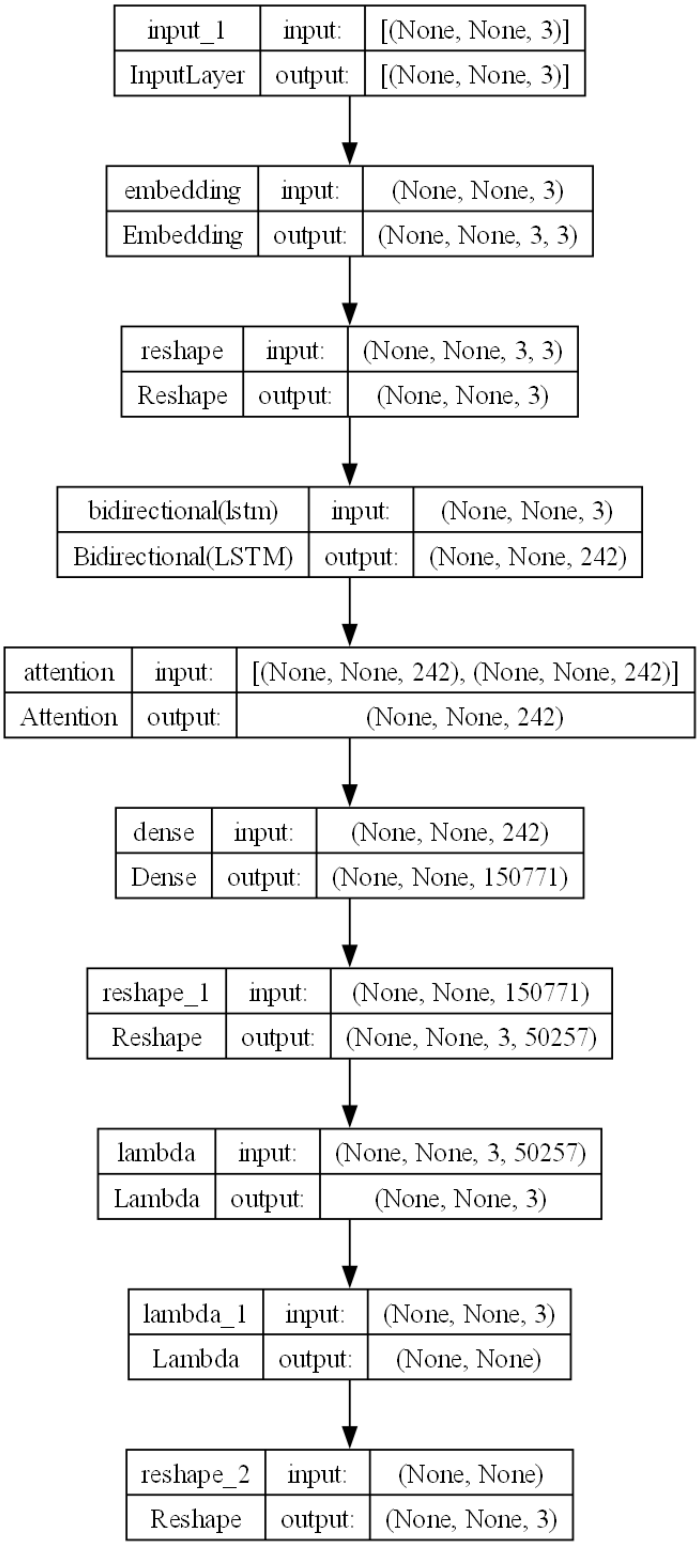

Рисунок 4.4 – Структура асоціативної пам'яті (рисунок створено самостійно)

Вдосконалена система з асоціативною пам’яттю має такі додаткові шари: Memory output –> Embedding –> Bidirectional LSTM –> Attention Mechanism.

Усе це працює за тим алгоритмом який було розписано у розділі 3.3, тоді

розглянемо цю структуру більш детально.

Через те що дана надбудова асоціативної пам’яті, не є рідною для трансформеру, її необхідно якимось-чином під’єднати до нього. Для цього використовуються декілька додаткових шарів що розтягують вхідні токіни на розмір словника (словник - усі токіни що може використовувати трансформер), який є рідним для даної системи.

Цей процес можна побачити на рисунку 4.4, де у шостому та сьомому шарах (dense – звичайний шар нейронів), змінюється розмірність токенів (3) у розмірність словника (3, 50257 – двовимірний вихід). Треба помітити, що сьомий шар є продовженням шостого і не впливає на вихід.

У свою чергу, восьмий та дев’ятий шари перетворюють розмірність словника у зрозумілий для трансформеру вид, матрично додаючи дані.

Наступним кроком виконується поєднання мультимодальних голів у єдине ціле для входу. Цей процес показано на рисунку 4.5.

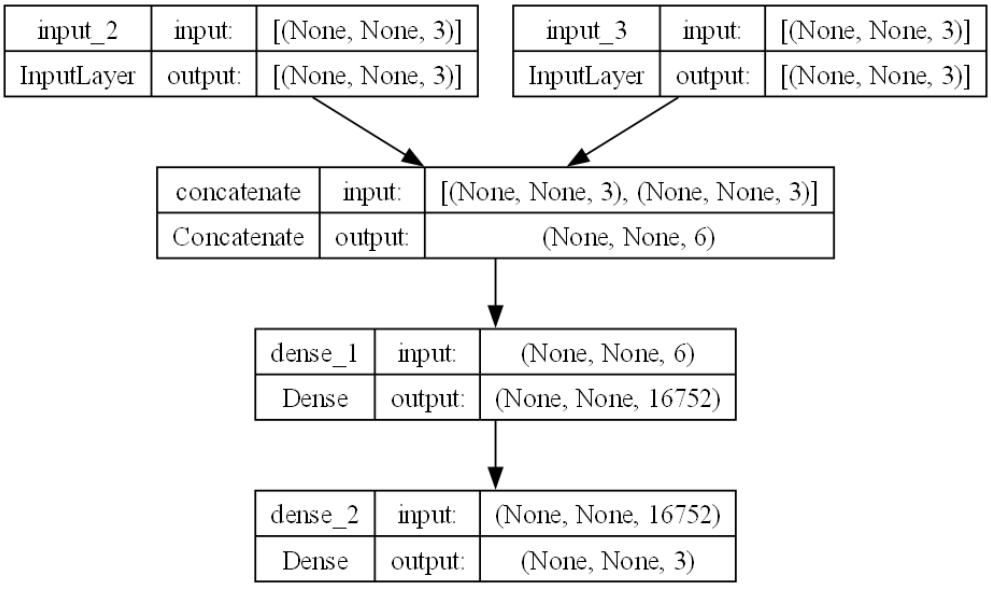

Рисунок 4.5 – Структура мультимодальних голів (рисунок створено самостійно)

Розглянувши принцип поєднання пам’яті та трансформеру на графічних зображеннях, перейдемо до розглядання реалізації даного типу пам’яті саме у коді нашого проекту. Код показано на рисунку 4.6.

```python
def build_memory(self):
    print(f"Building memory with latent dim {self.latent_dim}")
    input_shape = (None, self.latent_dim)

    k = (self.latent_dim+8)**2

    query_input = keras.Input(shape=input_shape)

    embedding = keras.layers.Embedding(self.hparams.n_vocab + 1, self.latent_dim, trainable=True, mask_zero=True)(query_input)

    embedding_query = keras.layers.Reshape((-1, self.latent_dim))(embedding)

    value = keras.layers.Bidirectional(keras.layers.LSTM(k, return_sequences=True, trainable=True, activation='gelu'))(embedding_query)

    attention_out = keras.layers.Attention()([value, value])

    output_de = layers.Dense(self.latent_dim*self.hparams.n_vocab, activation='sigmoid')(attention_out)
    output_de = layers.Reshape((-1, self.latent_dim, self.hparams.n_vocab))(output_de)
    output_de = layers.Lambda(lambda x: tf.reduce_sum(x, axis=-1))(output_de)
    output_de = layers.Lambda(lambda x: tf.reduce_sum(x, axis=-1))(output_de)

    output_de = layers.Reshape((-1, self.latent_dim))(output_de)

    en2de = keras.Model(inputs=[query_input], outputs=output_de)

    return en2de

1 usage
def build_concate(self):
    memory_input = keras.Input(shape=(None, self.latent_dim))
    memory_output = keras.Input(shape=(None, self.latent_dim))

    memory_concate = keras.layers.Concatenate()([memory_input, memory_output])
    de = layers.Dense(self.hparams.n_vocab//3, activation='gelu')(memory_concate)
    # de = layers.Dense(1, activation='gelu')(de)
    output_de = layers.Dense(self.latent_dim, activation='gelu')(de)

    en2de = keras.Model(inputs=[memory_input, memory_output], outputs=output_de)

    return en2de
```

Рисунок 4.6 – Програмний код асоціативної пам’яті (рисунок створено самостійно)

Відносно реалізації – на рисунку ми бачимо дві функції:

– функція build_memory – створює модель асоціативної пам’яті, однак можна зауважити що дана модель є гнучкою, тому може змінюватись в залежності від потреби;
– функція build_concate – виконує поєднання асоціативної пам’яті та трансформеру у єдине ціле.

Треба помітити що даний код реалізує у собі усе те що було розписано вище на рисунках 4.4 та 4.5.

Коли моделі GPT-2 необхідно звернутися до пам’яті, вона викликається функцією associate. Код даної функції показано на рисунку 4.7

```
def associate(self, X):
    if (type(X[0]) != list):
        if (type(X[0]) != np.ndarray):
            X = [X]
        else:
            if len(X[0]) == 1:
                X = X[0]
    if len(X[0]) != self.latent_dim and len(X[0]) < self.latent_dim:
        for i in range(self.latent_dim - len(X[0])):
            X[0].append(0)
    elif len(X[0]) > self.latent_dim:
        X = [X[0][:self.latent_dim]]
    else:
        pass
    X = np.array(X[0]).reshape(-1, 1, self.latent_dim)

    r = self.memory.predict(X)[0][0][:self.latent_dim].reshape(1, self.latent_dim).astype(int)
    Y = self.concate.predict(
        [X, np.array(r).reshape(-1, 1, self.latent_dim)]
    )[0][0][:self.latent_dim].reshape(1, self.latent_dim).astype(int)
    for i in range(len(Y[0])):
        if Y[0][i] < 0:
            Y[0][i] = 0
        elif Y[0][i] > self.hparams.n_vocab:
            Y[0][i] = self.hparams.n_vocab-1
    return Y
```

Рисунок 4.7 – Функція виклику асоціативної пам’яті (рисунок створено самостійно)

Таким чином ми розробили асоціативну пам’ять та її виклик у нашій системі. Тепер початкова модель GPT-2 вміє працювати з асоціативною пам’яттю, та викликати її, коли вона буде необхідна. Перейдемо до реалізації наступних частин нашої системи.

4.4 Опис взаємодії з системи

Ще одним важливим моментом для роботи даної системи, є саме генерація тексту, яку необхідно було реалізувати.

Реалізація даного процесу, розписана у файлу GPT-2. Система виконує запит на генерацію тексту, звертається до даної частини коду, і отримує необхідний набір дій.

Код реалізації процесу генерації занадто великий, том акцентуємо увагу тільки на важливих моментах.

Перша частина коду майже повторює код з Fine Tuning, тому що ми зводимо наші системи до однакових вимог.

На рисунку 4.8 показано частину коду генерації.

```
if prefix and not asm:
    context = tf.compat.v1.placeholder(tf.int32, shape: [batch_size, None])
    context_tokens = enc.encode(prefix)
else:
    HPARAMS = model.HParams(n_vocab=hparams.n_vocab, n_ctx=hparams.n_ctx, n_embd=hparams.n_embd,
                            n_head=hparams.n_head, n_layer=hparams.n_layer, model_dir=checkpoint_path)
    global ASGM
    ASGM = memory.ASGM(HPARAMS, dim=asm_dim)
    try:
        ASGM.load()
    except:
        raise FileNotFoundError("No memory file found. Please run finetune() first.")
    context = tf.compat.v1.placeholder(tf.int32, shape: [batch_size, None])
    context_tokens = enc.encode(prefix)
    context_tokens_n = ASGM.associate(context_tokens)[0].tolist()
    for cn in context_tokens:
        context_tokens_n.append(cn)

    context_tokens = context_tokens_n
    context_tokens = [x for x in context_tokens if x != 0]
```

Рисунок 4.9 – Код запиту на токенізацію (рисунок створено самостійно)

Система коли необхідно виконати генерацію, визначає чи необхідно використовувати асоціативну пам’ять та редагує вхідні данні за потребою. Якщо все ж є необхідність використовувати асоціативну пам’ять, текст перетворюється на токіни, а далі вже редагується асоціативною пам’яттю, процес. Процес перетворення було розписано раніше.

Далі, на рисунку 4.10, показано код генерації тексту.

```
p, p0, raw_output = sample.sample_sequence(
    hparams=hparams,
    length=min(length, length - (len(context_tokens) if prefix else 0)),
    start_token=enc.encoder['<|endoftext|>'] if not prefix else None,
    context=context if prefix else None,
    batch_size=batch_size,
    temperature=temperature, top_k=top_k, top_p=top_p, proba=True
)
proba = sess.run(tf.nn.softmax(p), feed_dict={context: batch_size * [context_tokens]})
try:
    print(f"Proba: {round((proba[0].max())*100, 2)}%")
except:
    print(f"Proba: 100%")
output = raw_output[:, 1:]
```

Рисунок 4.10 – Код запиту на генерацію (рисунок створено самостійно)

p – це проба яка показує якість згенерованого тексту.

Завдяки даному коду, система звертається до самого коду генерації, що по своєї суті, представляє собою математичні дії з даними. Пізніше, результат видається до консолі, та зберігається в графік.

Окрім генерації тексту, розглянемо конфіг файл main у якому знаходяться основні налаштування системі. На рисунку 4.11 зображено даний код.

```
config = {
    'associative': True,  #use associative memory
    'train': False,       #train model

    'title_length': 10,    #write title
    'logic_length': 20,    #write logic answer
    'essay_length': 100,   #write essay

    'visualize': False,   #visualize model
    'reserch': True,     #reserch model
    'server': False,      #run on server

    'asm_dim': 3,         #associative memory dimension
    'top_k': 20,          #top k logits
}

questions = {
    'logic': ['The best phone is',
              'Apple is better than Android because',
              'Android is better than Apple because',],
    'title': ['Apple released new',
              'Samsung released new'],
    'essay': ['Apple',
              'Samsung'],
}
```

Рисунок 4.10 – Основні налаштування та питання у системі (рисунок створено самостійно)

Розглянемо налаштування що можна зробити через даний файл:

- associative – даний пункт відповідає за дозвіл використання асоціативної пам’яті;
- train – відповідає за виконання навчання системи. Навчання використовується лише перший раз і пізніше вимикається. Постійно перенавчати систему не має сенсу;
- title/logi/essay _length - розмір згенерованого тексту у окремих випадках;

- visualize - відповідає за створення графіків моделей і ваг;
- reserch - відповідає за вмикання та вимикання порівняння моделей між собою. Даний пункт використовувався коли система була на етапі розробки, щоб при перенавчанні, не вмикати дослідження обох моделей;
- server - пункт для запуску моделі з серверу чи з тільки комп'ютеру. Змінює підхід до використання ресурсів;
- asm_dim – відповідає за зміну розмірність асоціативної пам'яті;
- top_k – виконує ті ж самі дії що було описано раніше, а саме змінює кількість слів що буде обрано для генерації.

Окрім того у даному файлі знаходяться питання до системи що поділені на три різні завдання.

Logic – логічне запитання, якому не вчили модель. Модель сама повинна відповісти на нього в залежності від свого типу пам'яті.

Title – генерація питання якому вчили модель. Даний тип питання використовується для перевірки працездатності системи.

Essay – генерація великого тексту на тему, якій система навчалась.

Після розглядання цих частин системи, перейдемо до розробки системи дослідження обох моделей, та проведення експеременту.

4.5 Розробка візуалізації результату

Даний модуль використається у нашій системі для дослідження та порівняння двох моделей з різними типами пам'яті. По суті, він необхідний лише для візуалізації даних, що нам потрібні. Саме порівняння моделей, буде виконуватись власноруч опираючись на створений графічний матеріал.

Розписувати програмний код даного модулю не є доцільним через те що це лише генератор графіків, тобто зображень, що побудовано на tensorflow, matplotlib та інших бібліотеках для візуалізації інформації.

Після розглядання усієї розробленої системи, перейдемо до основного дослідження, порівняння двох моделей.

5 ПРОВЕДЕННЯ ЕКСПЕРИМЕНТУ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

5.1 Проведення аналізу отриманих результатів

Після розробки самої системи, ми приступили до проведення дослідження.

Як було описано раніше, у розділі 4.4, перед двома моделями (З асоціативною пам’яттю та без) буде поставлено три різні завдання які вони повинні виконати.

Система аналізує дані речення, та вираховує значення двох основних параметрів:

Accuracy (Акуратність) - параметр який визначає наскільки токен є приближеним до потрібного значення.

Probability (Точність) - даний параметр показує з якою ймовірністю токен є наступним словом.

Наше порівняння обох моделей буде спиратися на створені за допомогою візуалізатору, графіки що в свою чергу залежать від даних запитань та якості моделі. Окрім того, порівняємо середнє значення показників, та згенерований текст.

Дані графіки містять необхідні нам параметри зазначені раніше, від яких буде будуватись висновок даного дослідження. Розглянемо графіки що показано на рисунках 5.1 та 5.2.

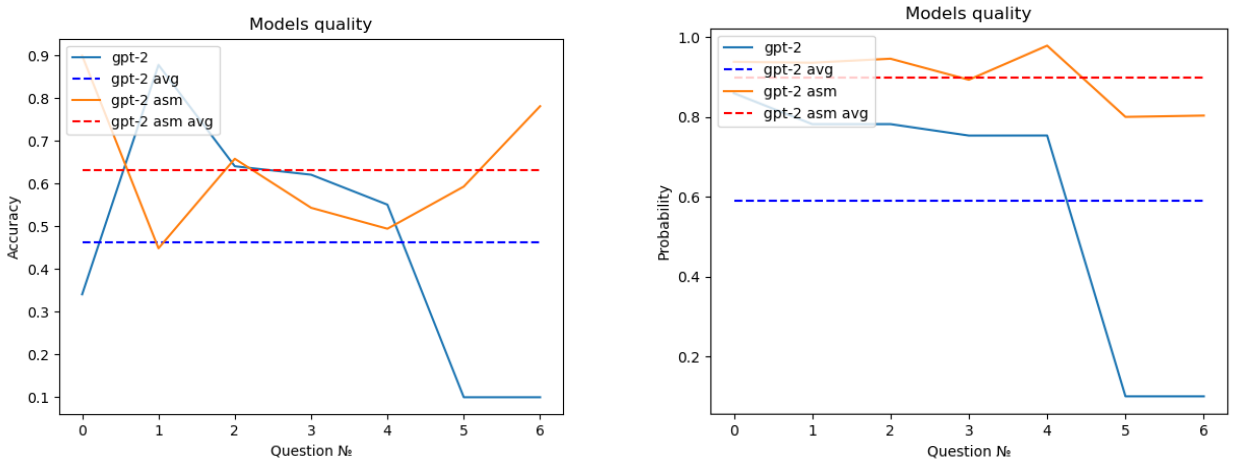

Рисунок 5.1 – Графіки акуратності та точності моделей (рисунок створено самостійно)

По-перше треба помітити, що від спроби до спроби, точність та акуратність моделей змінюватись не буде, тому що трансформер передає ймовірність токенів, а текст є лише проекцією випадкових токенів на їх ймовірність, а якщо не змінюється ймовірність, то не буде мінятись і акуратність. Значення можуть мінятися у тому випадку, якщо присутній шум, але для більш точного дослідження, його було вимкнено.

На графіку можна помітити різкий спад акуратності звичайної gpt-2, це виникає через те, що дана модель вміє працювати з великою кількістю токенів, але її якість у такому випадку, зменшується.

Порівнюючи за графіками обидві моделі, бачимо що у деяких моментах GPT-2 є більш точною, але досить ясно видно, що GPT-2 ASM є більш точною та акуратною.

На графіку також можна побачити середнє арифметичне значення обох моделей по двом показникам. Вони свідчать про значну різницю між точністю та акуратністю двох моделей. Дане значення також видається у консолі середовища програмування, що показано на рисунку 5.2.

```
Average Probability gpt-2: 0.5903813430241175
Average Probability asm: 0.8999042425836835
[0.3409748673439026, 0.8781747221946716, 0.640
[0.8999070525169373, 0.4486837387084961, 0.658
Average Accuracy gpt-2: 0.46172812325613843
Average Accuracy asm: 0.6314385661057064
```

Рисунок 5.2 – Середні значення показників точності та акуратності (рисунок створено самостійно)

Середнє значення показників моделі з асоціативною пам’яттю:

– точність – 0,900;
– акуратність – 0,631.

Середнє значення показників моделі без асоціативною пам’яті:

– точність – 0,590;
– акуратність – 0,461.

Розрахуємо відсоток різниці показників обох моделей та отримуємо такі

результати.

GPT-2 ASM на 31% за точністю, та на 17% за акуратністю, краще ніж GPT-2.

Наступним кроком порівняємо моделі між собою за згенерованим текстом. Фрагмент згенерованого тексту, представлено на рисунку 5.3.

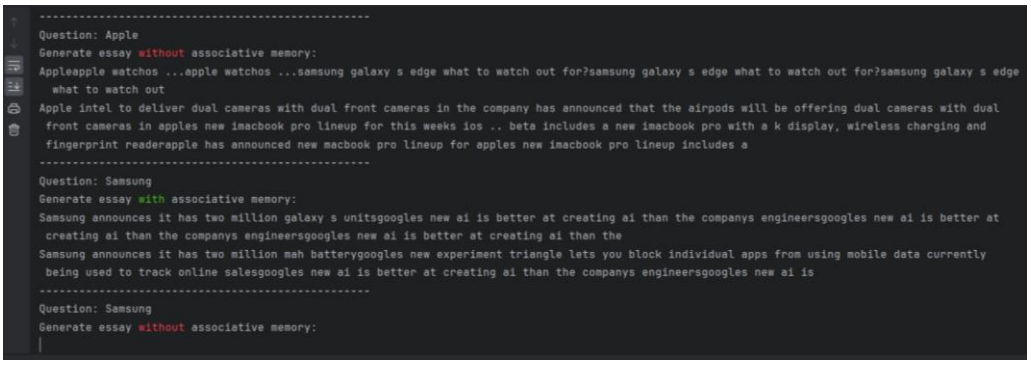

Рисунок 5.3 – Згенеровані речення (рисунок створено самостійно)

Можна помітити що у тексті присутні помилки, від чого жодна система не застрахована. Через те що ми використовуємо найменьшу модель, текст може не зовсім змістовним навіть при найвищій якості.

Результати зі значеннями та самі речення, зберігаються у окремому файлі Result. Даний файл знаходиться у додатку Г. Частину змісту даного файлу показано на рисунку 5.4.

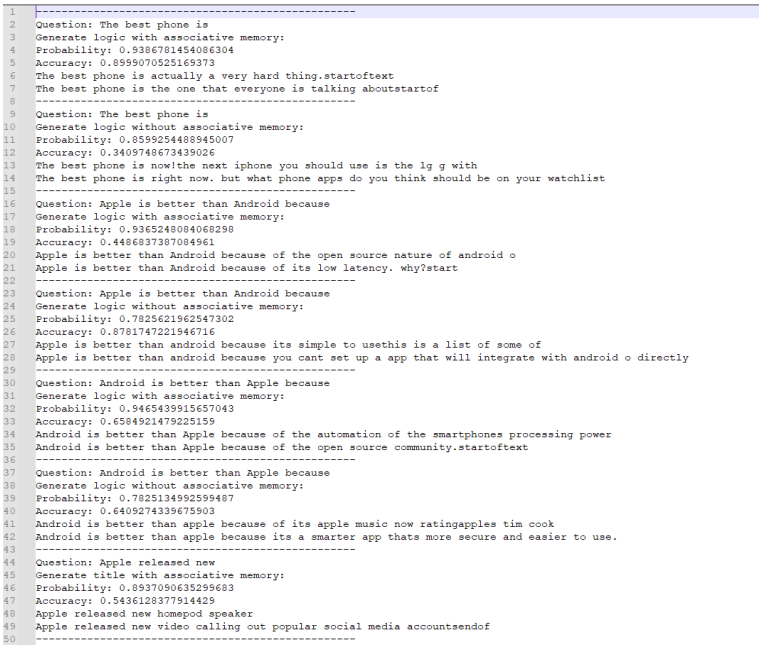

Рисунок 5.4 – Згенеровані речення (рисунок створено самостійно)

З даного файлу, можна узяти самі значення та порівняти їх між собою. Виконаємо порівняння двох систем на питаннях типу Logic та Essay. Розглянемо спочатку рисунок 5.5.

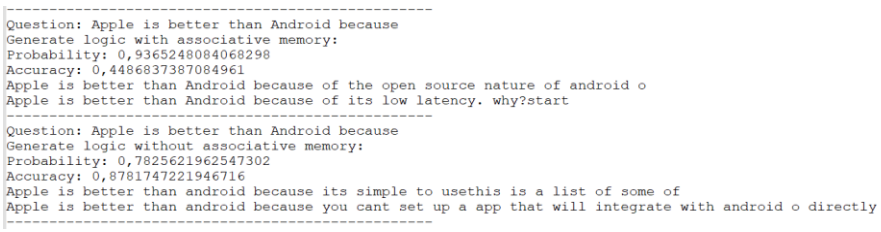

Рисунок 5.5 – Згенеровані речення від двох моделей - тест на питанні типу Logic (рисунок створено самостійно)

На ньому зображено згенерований текст двома моделями при тесті на питанні типу Logic. Бачимо що тексти краще там, де вище точність. Запам’ятаємо значення точності та акуратності, порівняємо із рисунком 5.6.

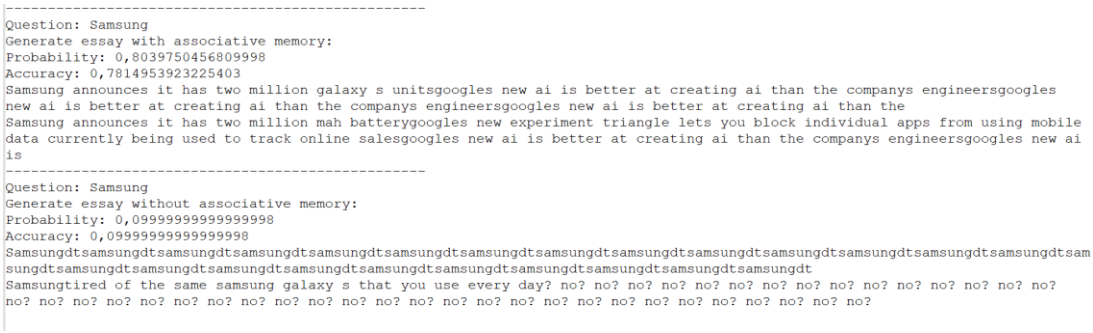

Рисунок 5.6 – Згенеровані речення від двох моделей тест на питанні типу Essay (рисунок створено самостійно)

Бачимо що чим вище показники, тим якісніше буде згенеровано текст. Окрім того, асоціативна модель краще генерує великий текст, а звичайна модель, у свою чергу, починає генерувати поганий текст який складається із повторів..

Тобто можемо зробити такий висновок, що асоціативна пам’ять потребує багато ресурсів (звертаючи увагу на те, які компоненти було використано для запуску даної моделі у розділі 4.2) для того щоб генерувати великий, відносно якісний текст (в нашому випадку використовується сама найменша модель). Коли у свою чергу, GPT-2 таких ресурсів не потрібно, але і текст буде гірше поєднано за змістом. Після усього виконаного, можна перейти до висновків.

ВИСНОВКИ

У результаті виконання даного магістерського дослідження було виконано усі поставлені мети з дослідницького завдання даного проекту.

Порівнявши обидві моделі між собою, можемо сказати, що використання асоціативної пам'яті є доцільним у випадках, коли необхідна генерація великих обсягів тексту, де надзвичайно важлива смислова схожість даного тексту. Така модель здатна забезпечити більш якісне та змістовне відтворення контексту, що є критично важливим для задач, де контекстуальна цілісність та логічна узгодженість тексту відіграють ключову роль, наприклад, при написанні великих статей, книг або детальних звітів.

З іншого боку, модель GPT-2 більше підходить для генерації невеликих відповідей на запитання або створення коротких текстів. Її можна ефективно використовувати у чат-ботах, автоматизованих службах підтримки клієнтів, або для створення коротких анотацій та описів. Основною перевагою GPT-2 є її відносна простота у використанні та менші вимоги до ресурсів у порівнянні з моделлю, що використовує асоціативну пам'ять.

Також варто зазначити, що навчання моделі з асоціативною пам’яттю є складним та ресурсозатратним процесом. Для успішної реалізації таких моделей необхідні значні обчислювальні потужності та великий обсяг даних для тренування. Тому їх доцільно використовувати лише за наявності відповідних ресурсів та потреби у генерації великих обсягів тексту.

На основі розглянутих вище графіків, можна зробити висновок, що якість моделі з асоціативною пам'яттю перевершує звичайну GPT-2 за точністю на 31%, а за акуратністю – на 17%. Це свідчить про значну перевагу моделей з асоціативною пам'яттю у завданнях, де точність та акуратність є критично важливими, підтверджуючи доцільність їх використання у відповідних випадках.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. A Clear Explanation of Transformer Neural Networks - http://surl.li/ttdzh

2. Zhao H. Global asymptotic stability of Hopfield neural network involving distributed delays / H. Zhao // Neural Networks. - 2004. - Vol. 17, N 1. - P. 48 - 53.

3. Github. Jaymody – PicoGPT. URL: http://surl.li/ttdza

4. Github. Jayveersinh-Raj - code_generation_gpt2 URL: http://surl.li/ttdyx

5. Github. Minimaxir - datasets/reddit_apple_android_2000.txt. URL: http://surl.li/ttear

6. Baolin Peng, Chenguang Zhu, Chunyuan Li, Xiujun Li, Jinchao Li, Michael Zeng, Jianfeng Gao. Few-shot Natural Language Generation for Task-Oriented Dialog. November – 2020.

7. Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Language Models are FewShot Learners. July – 2020.

8. Machine learning normalization. URL: https://www.analyticsvidhya.com/ blog/2020/04/feature-scaling-machine-learning-normalization-standardization/ (дата звернення: 09.04.2023).

9. G. Hinton, Improving neural networks by preventing co-adaptation of feature detectors, 2012, p. 18.

10. Євгеній Мамочка, Андрій Єрохін. «Дослідження Текстових Генеративних Систем з Асоціативною Пам’яттю». Збірник 12ої Міжнародної науково-технічної конференції «Інформаційні системи та технології» (ICT-2023) Частина 2. Молодіжна секція.

11. Mamochka E. I., student, Gerasymchuk T. V. « NEURAL NETWORKS. GPT TECHNOLOGY. MIDJOURNEY». Збірник наукових статей. Питання сучасної модернізації науки та освіти. Харків – 2023р.

12. Filatov V. O., Yerokhin A. L., Zolotukhin O. V., Kudryavtseva M. S. Hybrid simulation models for complex decision-making problems with partial uncertainty.

Information Extraction and Processing. 2022, 50(126), 78-86. DOI:https://doi.org/10.15407/vidbir2022.50.078

13. Dmytro Panchenko, Daniil Maksymenko, Olena Turuta, Andriy Yerokhin, Yana Daniiel, Oleksii Turuta . Evaluation and Analysis of the NLP Model Zoo for Ukrainian Text Classification // Communications in Computer and Information Science, 2022, 1698 CCIS, pp. 109–123. DOI: 10.1007/978-3-031-20834-8_6

14. Daniil Maksymenko, Nataliia Saichyshyna, Oleksii Turuta, Olena Turuta, Andriy Yerokhin, Andrii Babii. Improving the Machine Translation Model in Specific Domains for the Ukrainian Language // International Scientific and Technical Conference on Computer Sciences and Information Technologies, 2022, 2022-November, pp. 123–129. DOI: 10.1109/CSIT56902.2022.10000529

15. Extension Multi30K: Multimodal Dataset for Integrated Vision and Language Research in Ukrainian. Nataliia Saichyshyna, Daniil Maksymenko, Oleksii Turuta, Andriy Yerokhin, Andrii Babii, Olena Turuta / EACL 2023 - 2nd Ukrainian Natural Language Processing Workshop, UNLP 2023 - Proceedings of the Workshop, 2023, P.P. 54–61 / DOI: 10.18653/v1/2023.unlp-1.7