Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій

Кафедра програмних систем і технологій

УДК 004.912

На правах рукопису

ВИПУСКНА КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

Тема: "ГЕНЕРАЦІЯ УКРАЇНОМОВНОГО ТЕКСТУ ЗА ДОПОМОГОЮ НЕЙРОННИХ МЕРЕЖ"

Спеціальність – 121 "Інженерія програмного забезпечення"

ПОЯСНЮВАЛЬНА ЗАПИСКА

ВКБР.ІПЗ.....ІПЗ

Студент

ІПЗ-44_____ /Андрій СТАДНІК/

Науковий керівник

д.т.н., с.н.с._____/Віктор ШЕВЧЕНКО/

Консультант з питань нормоконтролю

_____/Тамара ЧАПОВСЬКА/

Допускається до захисту

Завідувач кафедри

д.т.н., проф_____/Олексій БИЧКОВ/

Київ - 2021

Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра програмних систем і технологій
Освітньо-кваліфікаційний рівень - бакалавр
Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖЕНО
Зав. кафедри програмних систем і технологій
_____ (Олексій БИЧКОВ)
“_____”_____2021 р.

ЗАВДАННЯ
НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Стадніку Андрію Валерійовичу

(прізвище, ім’я, по батькові)

1. **Тема роботи: «**Генерація україномовного тексту за допомогою нейронних мереж**»**

Керівник роботи: д.т.н., с.н.с. В. Л. Шевченко

Затверджені наказом №6 вищого навчального закладу від “11” листопада 2020р.

2. **Строк здачі студентом закінченої роботи** «___»_____2021р.

3. **Вихідні дані до дипломної роботи:** монографії, підручники, навчальні посібники, статті та тези конференцій вітчизняних і зарубіжних авторів, Інтернет-ресурси з питань обробки мови.

4. Зміст пояснювальної записки:

1. Аналіз існуючих моделей, їх порівняння та визначення доцільності використання.
2. Підготовка даних.
3. Створення моделей

4. Порівняння моделей

5. **Перелік графічного матеріалу (із зазначенням обов'язкових креслень)**:

_____

_____

6. Консультанти розділів проекту:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1 і 2 розділи аналітична частина	Віктор ШЕВЧЕНКО		
3 і 4 розділи практична частина	Віктор ШЕВЧЕНКО		

7. **Дата видачі завдання** «05» листопада 2020 р.

Керівник_____(Віктор ШЕВЧЕНКО)

Завдання прийняв до виконання_____(Андрій СТАДНІК)

Календарний план

№ з/п	Назва етапів виконання етапів бакалаврської роботи	Термін виконання етапів роботи	Відмітка про виконання
1	Уточнення постановки задачі	29.11.2020-05.12.2020	Виконано
2	Аналіз літератури	06.12.2020-04.01.2021	Виконано
3	Аналіз існуючих методів, концепцій та алгоритмів вирішення завдання	09.01.2021-16.01.2021	Виконано
4	Пошук та обробка даних	28.01.2021-14.02.2021	Виконано
5	Робота над моделлю, що використовує ланцюги Маркова	15.02.2021-20.03.2021	Виконано
6	Робота над моделлю, що використовує рекурентні нейронні мережі	21.03.2021-30.04.2021	Виконано
7	Робота над моделлю, що використовує модель трансформерів	02.05.2021-15.05.2021	Виконано
8	Оформлення і друк пояснювальної записки	16.05.2021-26.05.2021	Виконано
9	Оформлення презентації	27.05.2021-03.06.2021	Виконано

Студент_____(Андрій Стаднік)

Керівник роботи_____(Віктор Шевченко)

АНОТАЦІЯ

Випускна кваліфікаційна бакалаврська робота: 74 с., 11 рис., 4 додатки, 11 джерел.

Тема: Генерація україномовного тексту за допомогою нейронних мереж.

Об'єкт дослідження: моделі для генерації тексту, україномовні тексти.

Мета роботи: створення програмного забезпечення для генерації україномовного тексту.

Предмет дослідження: генерація тексту, нейронні мережі, ланцюги Маркова.

Результати дослідження:

Досліджено наявні моделі для генерації тексту, реалізовано 3 з них, створено програму для їх використання, порівняно їх результати.

Висновок:

В результаті роботи було створено 3 моделі для генерації тексту, та реалізовано програму для зручного їх використання.

ГЕНЕРАЦІЯ ТЕКСТУ, НЕЙРОННІ МЕРЕЖІ, ТРАНСФОРМЕРИ, ЛАНЦЮГИ МАРКОВА, ГЕНЕРАЦІЯ МОВИ.

АННОТАЦИЯ

Выпускная квалификационная бакалаврская работа: 74 с., 11 рис., 4 дополнения, 11 источника.

Тема: Генерация украиноязычного текста с помощью нейронных сетей.

Объект исследования: модели для генерации текста, украиноязычные модели.

Цель работы: создание программного обеспечения для генерации украиноязычного текста.

Предмет исследования: генерация текста, нейронные сети, цепи Маркова.

Результаты исследования:

Исследованы имеющиеся модели для генерации текста, реализовано 3 из них, создана программа для их использования, проведено сравнение их результатов.

Вывод:

В результате работы было создано 3 модели для генерации текста, и реализована программа для удобного их использования.

ГЕНЕРАЦИЯ ТЕКСТА, НЕЙРОННЫЕ СЕТИ, ТРАНСФОРМЕРЫ, ЦЕПИ МАРКОВА, ГЕНЕРАЦИЯ ЯЗЫКА.

SUMMARY

Final qualifying bachelor's thesis: 74 pages, 11 pictures, 4 appendices, 11 sources.

Topic: Generation of Ukrainian text using neural networks.

Object of research: models for text generation, texts in ukrainian.

Purpose: to create software for generating Ukrainian text.

Subject of research: text generation, neural networks, Markov chains.

Results of the research:

The existing models for text generation are studied, 3 of them are implemented, a program for their use is created, their results are compared.

Conclusion:

As a result, 3 models were created to generate text, and a program was implemented for their convenient use.

TEXT GENERATION, NEURAL NETWORKS, TRANSFORMERS, MARK'S CHAINS, LANGUAGE GENERATION.

ЗМІСТ

ПЕРЕЛІК ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

ГНН - Глибокі нейронні мережі.

ШІ - Штучний інтелект.

ШН - Штучне навчання.

ГН - Глибоке навчання.

ВСТУП

Актуальність роботи

З розквітом машинного навчання та, кажучи більш точно, глибоких нейронних мереж вчені отримали змогу розв’язувати завдання зовсім нового рівня. Обробка природньої мови є однією з них. За допомогою новітніх технологій ми змогли створити програми, що здатні розпізнавати настрій текстів, генерувати зображення по їх опису, відповідати на питання, спілкуватись як справжні люди. Одним із таких напрямів, який стало можливо імплементувати на належному рівні відносно недавно, є генерація тексту, який має сенс. Це має безліч використань: від відповідей на запитання до написання книг та пісень. Втім, саме в напрямі обробки природньої мови є особливість: здебільшого моделі повинні створюватись для кожної мови окремо. Очевидно, що для англійської мови було проведено набагато більше роботи, ніж для будь-якої іншої. Актуальною є задача використання напрацювань вчених, та перенесення їх для інших мов.

Порівняння роботи з відомими розв’язаннями проблеми

Наразі практично не існує якісних прикладних програм для генерації україномовного тексту. Під час пошуків було знайдено лише один інструмент:

Джеджалик [1] — Використовує рядки з текстів Івана Франка. Цей інструмент загалом призначається для типографічних задач. Текст не генерується за допомогою певної моделі, та речення є повністю взяті з наявних творів.

Мета і задачі дослідження

Метою бакалаврської роботи є створення моделей для генерації україномовного тексту, та реалізація інтерфейсу для їх використання. Одним із

прикладів застосування такого програмного інтерфейсу може бути створення так званого тексту “риби” — тексту, який використовують веброзробники для наповнення макетів сайтів перед демонстрацією. Такий текст повинен бути написаний тією ж мовою, що й майбутній контент, та, бажано, відповідати стилю та темі сайту.

Об’єктом дослідження є процес генерації україномовних текстів.

Предметом дослідження є моделі генерації тексту: ланцюги Маркова, технології згорткових нейронних мереж та архітектура трансформерів.

Наукова новизна отриманих результатів: вперше створено та адаптовано модель ланцюгів Маркова для задачі генерації україномовного тексту, набули подальшого розвитку моделі рекурентних нейронних мереж та трансформера для задачі генерації україномовного тексту. Порівняно їх результати.

Практичне значення одержаних результатів створено програму, що може бути використана при потребі створити штучний україномовний текст визначеної довжини.

Апробація результатів випускної кваліфікаційної бакалаврської роботи

Роботу було представлено на Всеукраїнському конкурсі студентських наукових робіт за спеціальністю “Інженерія програмного забезпечення”, де вона посіла 22-ге місце із 51 роботи.

Публікації

За результатами наукових досліджень, проведених у роботі, було підготовлено 2 публікації в збірнику матеріалів VII Східно-Європейська конференція “Математичні та програмні технології Internet of Everything”, 22-23.12.2020, Київ.

РОЗДІЛ 1

АНАЛІЗ МОДЕЛЕЙ, ЩО ІСНУЮТЬ, ЇХ ПОРІВНЯННЯ ТА ВИЗНАЧЕННЯ ДОЦІЛЬНОСТІ ВИКОРИСТАННЯ.

1.1 Огляд літератури за темою

Попри те, що обробка природньої мови (і, в тому числі й генерація природньої мови) за останні десятиліття розвивається шаленими темпами, існує потреба адаптації алгоритмів, створення датасетів для кожної мови окремо. Через це немає можливості взяти модель, що була натренована на англійськомовних даних, та використати її для української мови. Зважаючи на це, кількість літератури, що може бути релевантною є сильно обмеженою.

Statistical and neural language models for the Ukrainian Language — Anastasiia Khaburska, 2020 [2]

Стаття концентрується на моделюванні мови: отриманні вірогідності токена на основі попередніх. В статті описуються кілька схожих підходів, втім, кінцева мета робіт відрізняється, і жодного коду знайдено не було. Відповідно, в практичному плані задача не є вирішеною. Також, поточна робота використовує більш досконалі моделі.

На жаль, наразі це є єдиною роботою, пов’язаною із генерацією україномовного тексту. Напрям генерації тексту за допомогою машинного навчання успішно розвивається десятиліттями, втім, зовсім мало роботи було проведено саме для української мови.

1.2 Виклад загальної методики та основних методів досліджень

В межах цієї роботи було протестовано кілька моделей, а саме:

- Ланцюги Маркова [3]

  Ця модель була обрана як найпростіша, яка показує результат більш-менш наївного підходу. З нею буде порівнюватись результат інших моделей.

- Рекурентні нейронні мережі [4]

  Рекурентні мережі є класичним підходом до проблем, пов’язаних із задачами, в яких велику вагу має контекст, зокрема, з задачами, що пов’язані з обробкою природньої мови.

- Трансформери широкого призначення [5]

  Це досить новітня технологія, яка використовує цілком новий підхід до розв'язання задач, пов’язаних з обробкою природньої мови. Модель є дуже складною, і потребує багато технічних ресурсів, та дуже великої кількості даних. В цій роботі використано модель, яку було натреновано Tereveni-AI [6].

1.3 Ланцюги Маркова

Ланцюги Маркова є досить поширеним і відносно простим способом статистичного моделювання випадкових процесів. Вони використовувались у багатьох різних сферах - від генерації тексту до фінансового моделювання. Загалом, ланцюги Маркова концептуально досить інтуїтивні та дуже доступні тим, що їх можна реалізувати без використання будь-яких передових статистичних або

математичних концепцій. Вони є чудовим способом розпочати вивчення імовірнісного моделювання та методів науки про дані.

Для початку опишемо їх на дуже поширеному прикладі:

Уявіть, що існує два можливі стани погоди: сонячна чи хмарна. Ви завжди можете безпосередньо спостерігати поточний стан погоди, і це гарантовано завжди буде одним із двох вищезазначених станів.

Тепер ви вирішили, що хочете мати можливість передбачити, якою буде погода завтра. Інтуїтивно ви припускаєте, що в цьому процесі є важливим перехід, оскільки поточна погода має певне відношення до того, якою буде погода наступного дня. Отже, бувши відповідальною особою, ви збираєте дані про погоду протягом декількох років і підраховуєте, що шанс сонячного дня після похмурого дня становить 0,25. Ви також зауважили, що, по суті, шанс на похмурий день, що настане після похмурого дня, повинен бути 0,75, оскільки можливих лише два стани.

Тепер ви можете використовувати цей розподіл для прогнозування погоди на наступні дні, виходячи із поточного стану погоди на той час.

Цей приклад ілюструє багато ключових концепцій ланцюга Маркова. Ланцюг Маркова по суті складається з набору переходів, які визначаються певним розподілом ймовірностей, які задовольняють Марківська власність.

Зверніть увагу, як у прикладі розподіл ймовірностей отримується виключно шляхом спостереження за переходами від поточного дня до наступного. Це ілюструє властивість Маркова, унікальну характеристику марківських процесів: відсутність пам’яті. Це, як правило, не дає їм змоги успішно виробляти послідовності, в яких, як очікується, буде якась основна тенденція. Наприклад, хоча ланцюжок Маркова може імітувати стиль письма автора на основі частоти слів, він не зможе створити текст, що містить глибоке значення або тематичне значення, оскільки вони розробляються на значно довші послідовності тексту. Тому їм не вистачає можливості створювати контекстно-залежний вміст, оскільки вони не можуть врахувати повний ланцюжок попередніх станів.

1.3.1 Математичне представлення ланцюгів Маркова

Ланцюг Маркова є марковським процесом з дискретним часом та дискретним простором станів. Отже, ланцюг Маркова - це дискретна послідовність станів, кожен з яких виведений з дискретного простору станів (скінченна чи ні), і яка слідує за властивістю Маркова.

Математично ми можемо позначити ланцюг Маркова через $X = (X_n)_{n\in\aleph} = (X_0, X_1,...)$, де в кожен момент часу процес приймає свої значення в дискретному наборі E таких, що $X_n \in E \; \forall n \in \aleph$.

Тоді властивість Маркова можна описати наступним чином: $P(X_{n+1} = s_{n+1}|X_n = s_n, X_{n-1} = s_{n-1},...) = P(X_{n+1} = s_{n+1}|X_n = s_n)$. Ця формула показує, що вірогідність стану в момент часу n+1 залежить лише від стану процесу в момент часу n.

1.4 Рекурентні нейронні мережі

1.4.1 Нейрони

Штучний нейрон імітує властивості біологічного нейрона. На вхід штучного нейрона поступає множина сигналів, які є виходами інших нейронів. Кожен вхід множиться на відповідну вагу, і всі виходи підсумовуються, визначаючи рівень активації нейрона. Хоча мережеві парадигми досить різноманітні, в основі майже всіх їх лежить ця конфігурація. Тут множина вхідних сигналів, позначених x1, x2,…, xn, поступає на штучний нейрон. Ці вхідні сигнали, що в сукупності позначаються вектором X, відповідають сигналам, що приходять в синапси біологічного нейрона. Кожен сигнал множиться на відповідну вагу w1, w2,…, wn, і поступає на сумуючий блок (рис. 2.1).

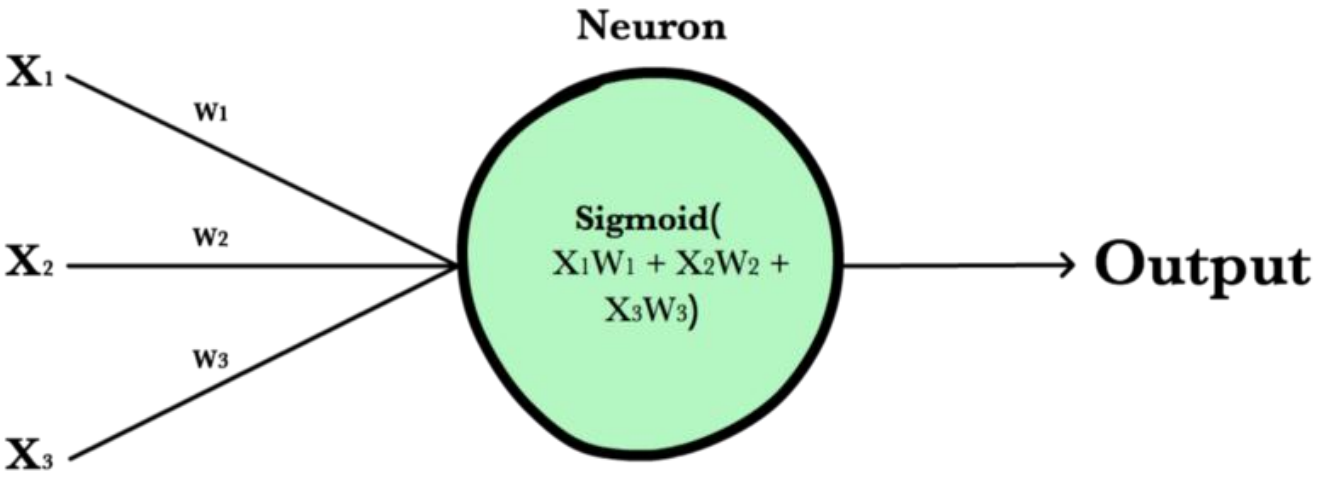

Рис. 2.1 Нейрон

1.4.2 Нейронні мережі

Ці нейрони поєднуються в нейронні мережі (рис. 2.2), які дозволяють апроксимувати значно складніші функції. Першопочатково існували одношарові нейронні мережі - персептрони. М.Мінський строго проаналізував цю проблему [7] і показав, що є жорсткі обмеження на те, що можуть виконувати одношарові перцептрони, і, отже, на те, чому вони можуть навчатися.

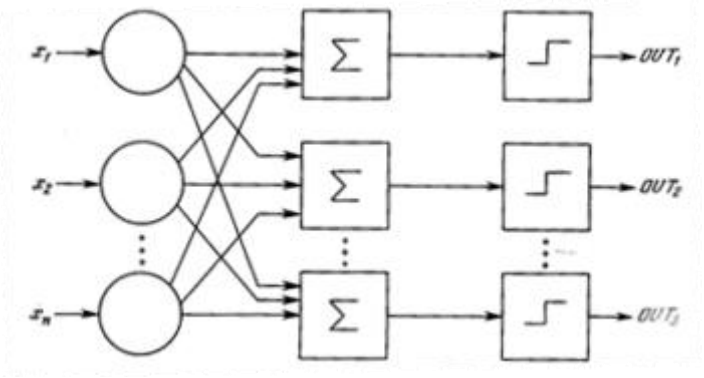

Рис. 2.2 Персептрон

1.4.3 Багатошарові нейронні мережі

Все змінилося, коли у 1986 р. були відкриті багатошарові нейронні мережі [8] (рис. 2.3), що володіють значно більшими можливостями, ніж одношарові.

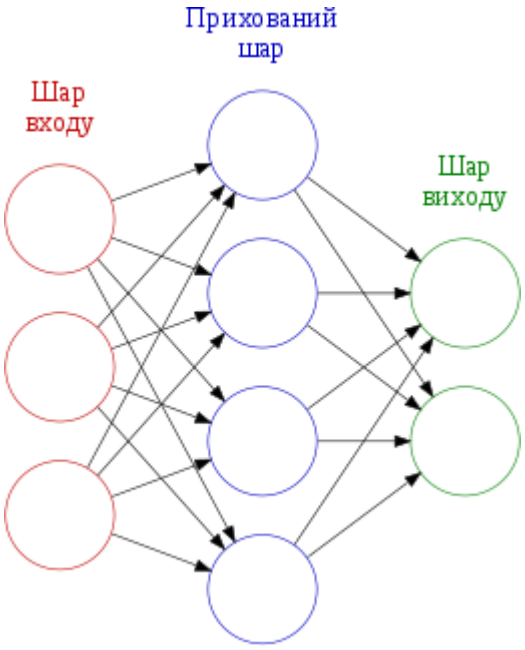

Рис. 2.3 Багатошарова нейронна мережа

Особливістю багатошарової нейронної мережі є нелінійна функція активації, що, комбінуючись багато раз дозволяє апроксимувати дуже складні нелінійні функції. Прикладом такої функції є сігмоїда (рис. 2.4).

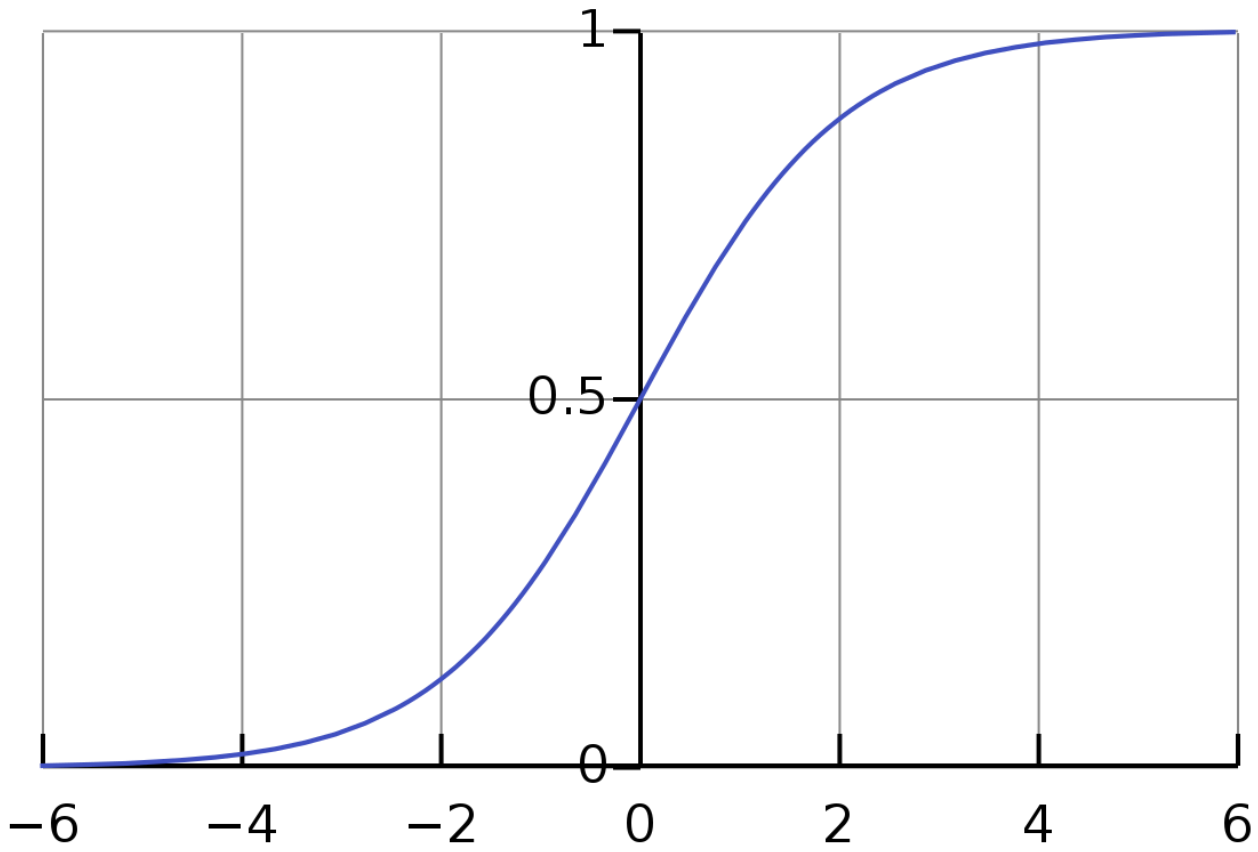

Рис. 2.4 Сігмоїда

1.4.4 Рекурентні нейронні мережі

Рекурентна нейронна мережа (RNN) - це тип штучної нейронної мережі, яка використовує послідовні дані або дані часових рядів. Ці алгоритми глибокого навчання зазвичай використовуються для порядкових або часових проблем, таких як переклад мови, обробка природної мови (nlp), розпізнавання мови та субтитри до зображень; вони включені в такі популярні програми, як Siri, голосовий пошук та Google Translate. Подібно до прямих та згорткових нейронних мереж (CNN), періодичні нейронні мережі використовують навчальні дані для навчання. Вони

відрізняються своєю “пам’яттю”, оскільки беруть інформацію з попередніх входів, щоб впливати на поточний вхід і вихід (рис. 2.5). У той час як традиційні глибинні нейронні мережі припускають, що входи та виходи не залежать один від одного, вихід повторюваних нейронних мереж залежить від попередніх елементів у послідовності. Хоча майбутні події також могли б допомогти у визначенні результату даної послідовності, однонапрямні рекурентні нейронні мережі не можуть враховувати ці події у своїх прогнозах.

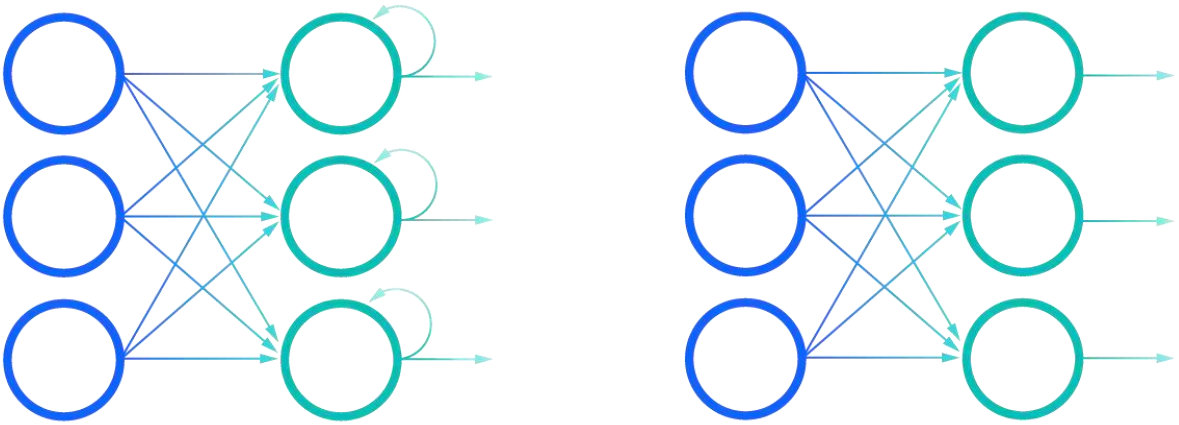

Рис 2.5 Рекурентні(а) та звичайні(б) шари

1.5 Модель трансформера

1.5.1 Увага

Увага (attention) - це використання (як правило, лінійної) комбінації попередніх прихованих векторів у вхідній послідовності (речення, у випадку моделі послідовності до послідовності). Ця комбінація прихованих векторів, як правило, поєднується з вектором поточного слова і подається на наступний шар моделі.

Цей підхід дозволяє моделі використовувати слова, які важливі через контекст.

На рис. 2.6 ми можемо побачити, що слово it_ асоціюється зі словами The animal, і це знання дозволяє моделі більш широко використовувати контекст речення.

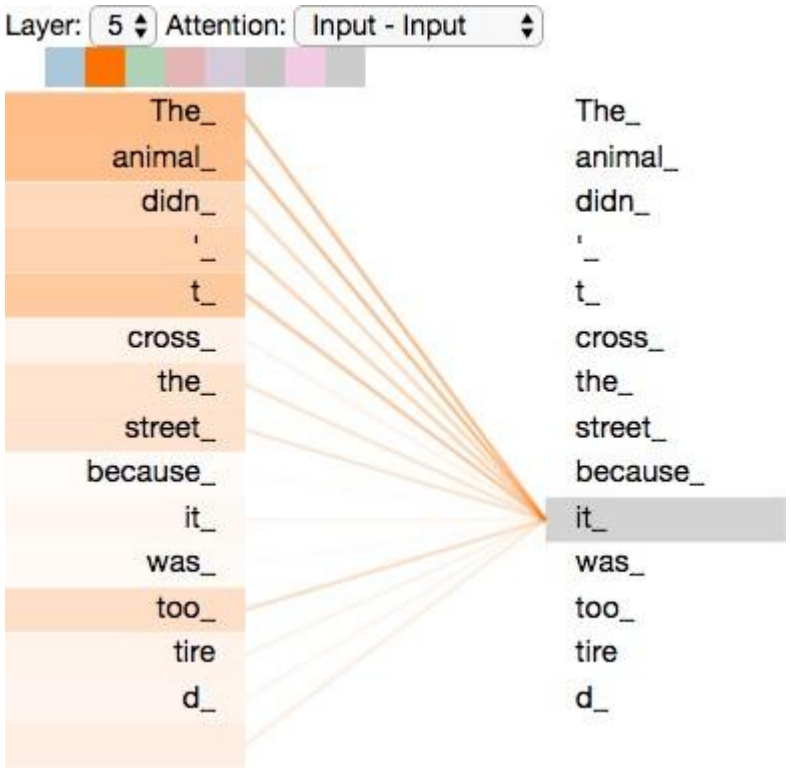

Рис 2.6 Рекурентні(а) та звичайні(б) шари

Самоувага — це нова версія техніки уваги. Замість того, щоб розглядати лише попередні приховані вектори при розгляді ембедінгу слів, самоувага — це зважена комбінація всіх інших слів (включаючи ті, що з’являються далі у реченні):

Перший шар кодерів отримує результат ембедінгу слова, навченого на вихідних словах. Це може бути токенізація в простому випадку, або, зазвичай, результат алгоритму ембедінгу. Кожному енкодеру в мережі після цього надаються вихідні дані попереднього енкодера.

Інша чудова річ щодо обчислення самоуваги полягає в тому, що оскільки вона передбачає лише множення точок та деякі скалярні операції над векторами, її можна легко записати як (швидку, ту, що паралелізується) однопрохідну матричну операцію. Це дозволяє швидко обчислити як прямий, так і зворотних проходів на цьому шарі.

1.5.2 Ембедінг позиції

Самоувага не усвідомлює позиції, наприклад. Вона не може розрізнити речення "Мишка з'їла кота". і "Кіт з’їв мишку".

Додавання цієї інформації до моделі вимагає певної форми позиційного кодування. У трансформері зі статті це досягається додаванням позиційного вектора до вбудовування слова перед введенням до рівня самоуваги. Позиційний вектор - це структура (наприклад, з використанням sin, cos), яка дозволяє дізнатись відносну позицію слова в реченні. Додавання цього до загального ембедінгу створює залежність від позиції в даних, яку можна дізнатись з ембедінгу позиції; якщо ця позиційна залежність важлива для завдання, модель може використовувати цю інформацію для її вивчення.

1.5.3 Трансформери

Трансформери були однією з найважливіших нових ідей у 2019 році. Трансформери були представлені в статті "Attention is all you need"[9]. Це архітектура моделі для завдання машинного перекладу, яка використовує увагу, щоб перевершити попередні моделі SOTA, послідовності до послідовності, за багатьма завданнями.

Трансформери - це архітектура моделі, що перетворює послідовність на послідовність: вони все ще мають окремі компоненти енкодера та декодера. Різниця полягає у використанні вдосконаленої форми уваги, відомої як самоувага, яка, крім своєї виражальної сили, має обчислювальну перевагу, тому що вона виражається як матрична операція.

Основна архітектура складається з набору енкодерів, повністю підключених до набору декодерів. Кожен енкодер складається з двох блоків: блоку самоуваги та повнозв’язного шару. Кожен декодер складається з трьох блоків: блоку самоуваги, блоку уваги кодера-декодера та повнозв’язного шару.

Також, в оригінальному трансформері використовуються остаточні зв’язки, що дозволяють сигналу переходити до наступних шарів напряму, таким чином, дозволяючи пропускати в даному разі непотрібні шари. Це дозволяє звертати увагу у різних контекстах (рис. 2.7).

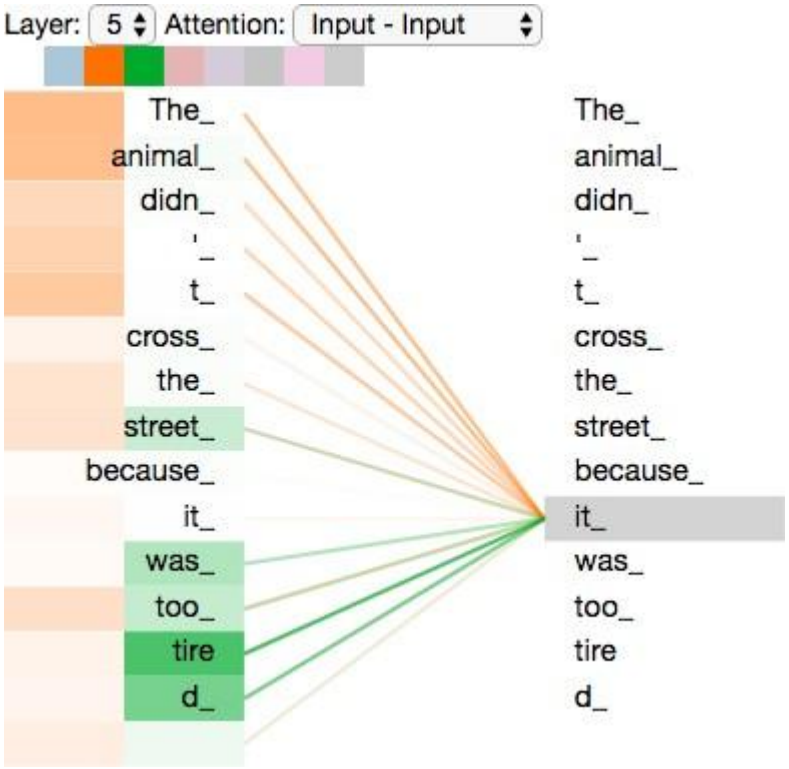

Рис 2.7 Рекурентні(а) та звичайні(б) шари

1.5.4 GPT

Оригінальний документ про генеративну попередню підготовку (GPT) мовної моделі був написаний Алеком Редфордом та його колегами та опублікований в препринті на вебсайті OpenAI 11 червня 2018 р. Це показало, як генеративна модель мови здатна здобувати світові знання та обробляти дальні залежності шляхом попередньої підготовки на різноманітному корпусі з довгими відрізками суміжного тексту.

1.5.5 GPT-2

Generative Pre-trained Transformer 2, широко відомий під своєю скороченою формою GPT-2, є некерованою мовою трансформатора та наступником GPT. GPT-2 був вперше анонсований у лютому 2019 року, лише демонстративні версії спочатку були оприлюднені для широкого загалу. Повна версія GPT-2 не була негайно випущена через занепокоєння з приводу можливого зловживання, включаючи програми для написання фейкових новин.

РОЗДІЛ 2

ПІДГОТОВКА ДАНИХ

2.1 Створення датасету

Для того, щоб навчити певну модель, необхідно мати базу даних. Для цієї роботи було вирішено обмежитись простими моделями, які не потребують надвеликих ресурсів та гігабайти даних. Як дані було обрано повну збірку віршів Шевченка. Після завантаження даних, опрацювання, фільтрації їх, було отримано 22628 рядків україномовного тексту.

Приклад:

Реве та стогне Дніпр широкий,

Сердитий вітер завива,

Додолу верби гне високі,

Горами хвилю підійма.

І блідий місяць на ту пору

Із хмари де-де виглядав,

Неначе човен в синім морі,

То виринав, то потопав.

Ще треті півні не співали,

Ніхто нігде не гомонів,

Сичі в гаю перекликались,

Та ясен раз у раз скрипів.

В таку добу під горою,

Біля того гаю,

Що чорніє над водою,

Щось біле блукає.

Може, вийшла русалонька

Матері шукати,

А може, жде козаченька,

Щоб залоскотати.

РОЗДІЛ 3

СТВОРЕННЯ МОДЕЛЕЙ

3.1 Ланцюги Маркова

Тепер ми можемо використати наявні знання для створення моделі для генерації україномовного тексту.

```
from collections import defaultdict

def markov_chain(text):
    words = text.split()
    collocation_dict = defaultdict(list)
    for current_word, next_word in zip(words[0:-1], words[1:]):
        collocation_dict[current_word].append(next_word)
    collocation_dict = dict(collocation_dict)
    return collocation_dict
```

Таким чином, для кожного слова ми отримаємо список слів, які будь-коли за ним слідували. Замість вірогідностей використовуються повтори слів. Далі лишається лише обрати одне з них, та створити таким чином речення необхідної довжини:

```
def generate_sentence(chain, word_count, cur_word=None):
    cur_word = cur_word or random.choice(list(chain.keys()))
    sentence = cur_word.capitalize()
    for _ in range(word_count-1):
        next_word = random.choice(chain[cur_word])
```

```
        sentence += ' ' + next_word
        cur_word = next_word

    sentence += '.'
    return sentence
```

Тепер лишається лише використати певний текст для створення “таблиці переходів”. У цьому випадку використано датасет із віршами Шевченка:

```
def main():
    with open("../data/Shevchenko/Shevchenko_no_latin.txt", "r")
as f:
        text = f.read()
    text = text.translate(dict.fromkeys(map(ord,
'\n!"\'(),-.---:;?'), ' '))
    print(generate_sentence(markov_chain(text), 40))

if__name__== "__main__":
    main()
```

Також необхідно видалити розділові знаки, бо вони сильно збільшують кількість “окремих слів”: “привіт” та “привіт,” вважатимуться різними словами.

3.1.1 Результат

Тепер можемо запустити, та подивитись на отриманий результат:

Пучині Нашій правді жить Знать добре Московщина Кругом хлопці та хрещатий Притоптаний коло Будищ І в наймах марніє моя Укрылася от що й весна чорну землю спалить Орла ж була Де ж ті добрі люди Кохалася а н е лю лі Питала

б ється Ось на Кубань хто небудь Мені так на Ярему Веде перед образом твоїм людям Та й укинув у журби питається Де поділась Занапастилась одуріла Мороз розум наш діла його речам А він і віки Тяжко брате Луже Хортице сестро Аполлона Навчи голубко поможи Полазить трохи і не таку достать я плачу Ба ні на сонці дуже страх аж ніби сам бачив А дітям На небі ні оселі Ні не сталося тойді було і слава Стане їм На мене розпинають Морозять шкварять на музики

Поки що результат не найкращий. Звісно, як вже було сказано, ланцюги Маркова не використовують історію та контекст, тож марно очікувати на зв’язний текст. Втім, часто частини речень все ж мають сенс.

3.2 Рекурентні нейронні мережі

3.2.1 Підготовка

Для створення моделі було використано фреймворк tensorflow.

Спочатку необхідно під’єднати необхідні бібліотеки:

```
import tensorflow as tf

from tqdm.auto import tqdm, trange
import numpy as np
import os
import time
```

Далі завантажимо датасет із файлу:

```
path_to_file = '../data/Shevchenko/Shevchenko_no_latin.txt'
text = open(path_to_file, 'rb').read().decode(encoding='utf-8')
```

3.2.2 Підготовка даних

Після цього нам необхідно створити множину всіх символів, які будуть генеруватись нейронною мережею. На відміну від попереднього методу, нейронна модель працює на рівні символів, тож їй необхідно буде, перш ніж працювати на рівні речень, навчитись створювати справжні слова.

```
vocab = sorted(set(text))
```

```
print('{} unique characters'.format(len(vocab)))
char2idx = {u:i for i, u in enumerate(vocab)}
idx2char = np.array(vocab)

text_as_int = np.array([char2idx[c] for c in text])
```

І одразу переведемо текст у масив чисел, бо нейронні мережі саме так сприймають інформацію.

Після цього необхідно визначити довжину одного речення, з яким працюватиме мережа. Нехай це буде 100. Тоді кількість таких речень можна однозначно порахувати. Після цього створимо датасет та послідовність батчів.

```
seq_length = 100
examples_per_epoch = len(text)//(seq_length+1)
char_dataset = tf.data.Dataset.from_tensor_slices(text_as_int)

sequences = char_dataset.batch(seq_length+1,
drop_remainder=True)
```

Далі нам необхідно створити пари вхідні/вихідні дані. Для цього кожному символу, що знаходиться на позиції n ми поставимо у відповідність символ на позиції n+1.

```
dataset = sequences.map(lambda s: (s[:-1], s[1:]))
```

Тепер визначимо розмір батчу, змішаємо датасет, щоб отримати більш рівномірний розподіл, та створимо із нього набір батчів.

```
dataset = dataset.shuffle(10000).batch(64, drop_remainder=True)
```

3.2.3 Визначення моделі

Тепер, коли ми маємо необхідний датасет, створимо модель для генерації тексту. Для цієї моделі використаємо такі види нейронних шарів:

1. Ембедінг

Ембедінги дають нам спосіб використовувати ефективне, щільне подання, в якому подібні слова мають подібне кодування. Важливо, що вам не потрібно вказувати це кодування вручну. Ембедінги - це щільний вектор значень із плаваючою комою (довжина вектора є параметром, який ви вказуєте). Замість того, щоб вказувати значення для вбудовування вручну, це параметри, що піддаються навчанню (ваги, вивчені моделлю під час навчання, так само, як модель вивчає ваги для щільного шару).

2. GRU

GRU (Gated Recurrent Unit) — шар, що реалізує рекурентний нейрон. Його особливістю є використання так званого бар'єру — кожен нейрон перед використанням "пам'яті" стирає та перезаписує певним чином деякі її значення. Те, яким чином це відбувається, змінюється під час навчання.

3. Повнозв'язний шар

Звичайний шар, де кожне вихідне значення є лінійною комбінацією вхідних даних, пропущене через функцію активації для забезпечення нелінійності.

```
def build_model(
    vocab_size, embedding_dim, rnn_layers, rnn_units,
batch_size, stateful=True
):
    return tf.keras.Sequential(
        [
            tf.keras.layers.Embedding(
                vocab_size, embedding_dim,
```

```
batch_input_shape=[batch_size, None]
            )
        ]
        + [
            tf.keras.layers.GRU(
                rnn_units,
                return_sequences=True,
                stateful=stateful,
                recurrent_initializer="glorot_uniform",
            )
            for _ in range(rnn_layers)
        ]
        + [tf.keras.layers.Dense(256)]
    )
```

Кількість нейронів у шарі, та кількість шарів — змінні, які можна змінювати в залежності від кількості даних.

Побудуємо модель:

```
model = build_model(
    vocab_size=len(vocab),
    embedding_dim=256,
    rnn_layers=3,
    rnn_units=256,
    batch_size=BATCH_SIZE)
```

В результаті модель має такий вигляд:

```
model.summary()
```

Model: "sequential"

_____

Layer (type) Output Shape Param #

==========

embedding (Embedding) (64, None, 256) 24832

_____

gru (GRU) (64, None, 256) 394752

gru_1 (GRU) (64, None, 256) 394752

gru_2 (GRU) (64, None, 256) 394752

dense (Dense) (64, None, 97) 24929

==========

Total params: 1,234,017

Trainable params: 1,234,017

Non-trainable params: 0

В якості функції втрат використаємо sparse_categorical_crossentropy. Це функція втрат, яку використовують, якщо результат є категорією (не числом), і, при цьому, він подається не у вигляді вектору з n елементів, де кожен із елементів може приймати значення від 0 до 1, і означає ймовірність відповідного результату, а у вигляді числа, де кожне число означає свій клас:

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

model.compile(optimizer='adam', loss=loss)
```

Після цього лишається натренувати модель:

```
EPOCHS = 30

history = model.fit(dataset, epochs=EPOCHS,
callbacks=[checkpoint_callback])
```

Epoch 1/30

83/83 [==============================] - 5s 30ms/step - loss: 3.7702

Epoch 2/30

83/83 [==============================] - 3s 29ms/step - loss: 2.5826

...

Epoch 28/30

83/83 [==============================] - 3s 30ms/step - loss: 1.3424

Epoch 29/30

83/83 [==============================] - 3s 30ms/step - loss: 1.3348

Epoch 30/30

83/83 [==============================] - 3s 30ms/step - loss: 1.3097

Після цього натреновану модель для посимвольної генерації можна буде отримати наступним чином:

```
model = build_model(vocab_size, embedding_dim, rnn_units,
batch_size=1)

model.load_weights(tf.train.latest_checkpoint(checkpoint_dir))

model.build(tf.TensorShape([1, None]))
```

Нарешті, для того, щоб згенерувати текст, необхідно викликати функцію, яка повертає наступний символ на основі попередніх бажану кількість разів, та правильно сконвертувати числа в символи за допомогою таблиці, яку ми отримали раніше.

```
def generate_text(model, start_string, vocab, num_generate=400):
    char2idx = {u: i for i, u in enumerate(vocab)}
    idx2char = np.array(vocab)

    input_eval = [char2idx[s] for s in start_string]
    input_eval = tf.expand_dims(input_eval, 0)

    text_generated = []

    # Increase to choose less probable characters more often
    temperature = 1.0

    model.reset_states()
```

```
    for _ in trange(num_generate, leave=False):
        # Get predictions for the model
        predictions = model(input_eval)
        # Remove unccecessary dimension (because batch_size ==
1)
        predictions = tf.squeeze(predictions, 0)

        # Change the probabilities
        predictions = predictions / temperature
        # Choose the result
        predicted_id = tf.random.categorical(
            predictions, num_samples=1)[-1, 0].numpy()

        # Update input
        input_eval = tf.expand_dims([predicted_id], 0)

        text_generated.append(idx2char[predicted_id])

    return start_string + ''.join(text_generated)
```

3.2.4 Аналіз гіперпараметрів

Нарешті, щоб отримати текст, необхідно просто викликати функцію:

```
print(generate_text(model, start_string=u"Діточки"))
```

Приклад результату:

Діточки!

Отож, може, де твого – Максимом

Хто на світі робити?

Не побачиш, як горою

У неділю на гердою

Пропалося на далекогих.

Минула моя-таки и градудом,

Суділи ноги. та спочила,

Голосно Алквень злая си, чути!"

– "Яка буде?

Дивуються люди!

Умирає, євас!..

А тепер мені

Як суєтосуть була,

Лукаві руки жить.

Посылів поховались,

З глібом свище, що стан вирву помелал.

Мамами свої любить. сни ги

Можна побачити, що є багато слів, які не існують в українській мові, хоча й схожі на справжі. Схоже, що модель є недостатньо складної для даної задачі, або ж вона недостатньо навчена. Оскільки наразі метою є лише отримання правдоподібного тексту, допустимо прийняти перенавчену модель.

Результат при кількості нейронів у рекурентних шарах рівній 512

Діточки і хресті

Розіб'є доглядавтую прії.

За що мене байдуже!

За ними жнива!

Сама хочеш – лихо мріє.

Мій батько ж моє! серце плаче,

Мов парубками

Козу з брандірою чорнують.

Нехай мій синочку, сумують,

Великий задумчеко...

Минули літа молодії,

Таки будуть водою,

І не бачили й собі з ним,

З дорогом

До гімности щебетата,

Темному полюбила.

Мені їх наші козак?

Добре намисто, обнімати,

Проки

Результат при кількості нейронів у рекурентних шарах, рівній 1024:

Діточкиму

Мірку страда. А заплакав

Або слав'яться

В Далакий великамить,

Дрьго любила порадила.

А Хозмиетленсь словом ізами,

З могил стреїть).

Немним, бальше сова!

Нехай и, серце, зелетає,

Штарий помолишесь,

Та що сонняки село, за поликали,

І ганува підла,

Утехеліворні тепереві

Остало серце, моя сладино,

Згалвіми одиноким

Тепер святиє

Слігами долюбом козаки

Поставнему той що тюмую,

Ніж з

Можемо побачити, що з ростом кількості нейронів результат погіршився. Це сталось через те, що тренування відбувалось лише протягом 30 епох. Чим більша модель, тим більше часу необхідно для того, щоб її натренувати.

Спробуємо перебрати різні кількості шарів при середній кількості нейронів в шарі (403):

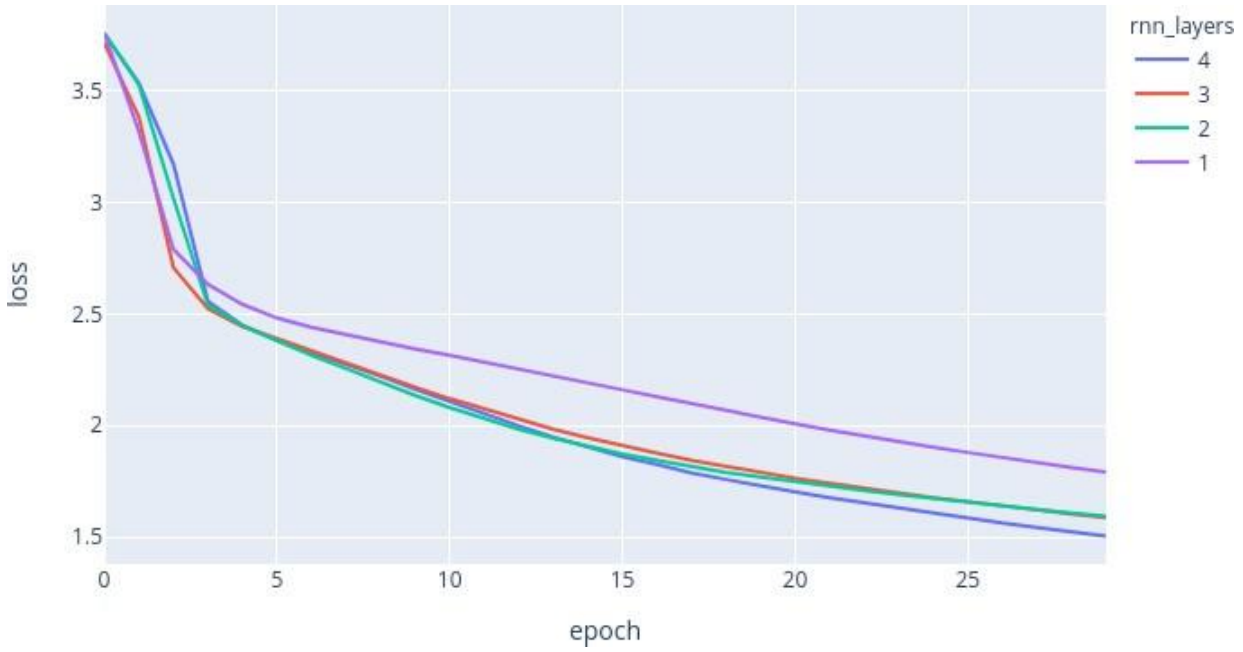

Рис. 3.1 Залежність помилки від кількості шарів в мережі

Як бачимо, при поточній кількості нейронів та епох кількість шарів не має великого значення. Втім, пам’ятаємо, що при великій кількості епох більші нейронні мережі можуть краще навчитись. Тепер зафіксуємо кількість шарів (3), та порівняємо результати з різною кількістю нейронів в шарі:

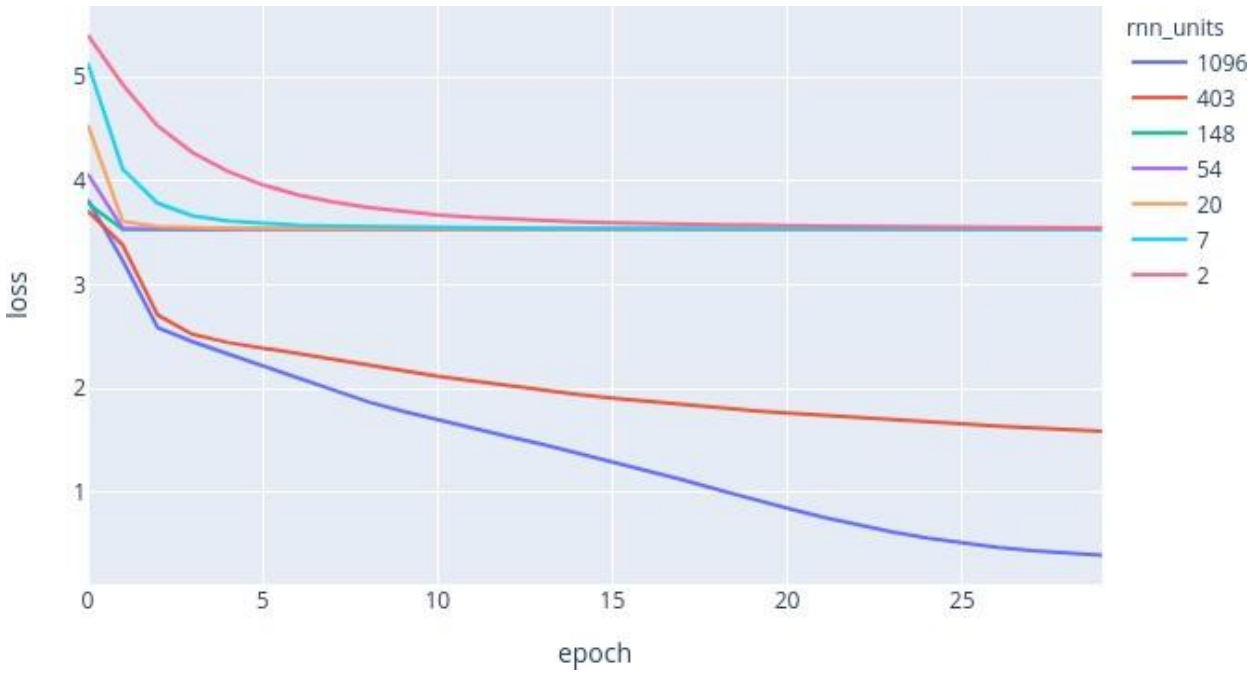

Рис. 3.2 Залежність помилки від кількості нейронів в шарі

З даного графіку можемо зробити висновок, що використовувати менше ніж 403 нейронів в одному шарі неефективно.

Врешті-решт, зробимо пошук по всій множині можливих пар кількостей шарів та нейронів в кожному шарі, та порівняємо результати. Зауважте, що кількість епох дорівнює 30, тож цілком можливо, що більші нейронні мережі могли б досягти кращих результатів, втім, в такому разі тривалість навчання сягла б надто високих значень:

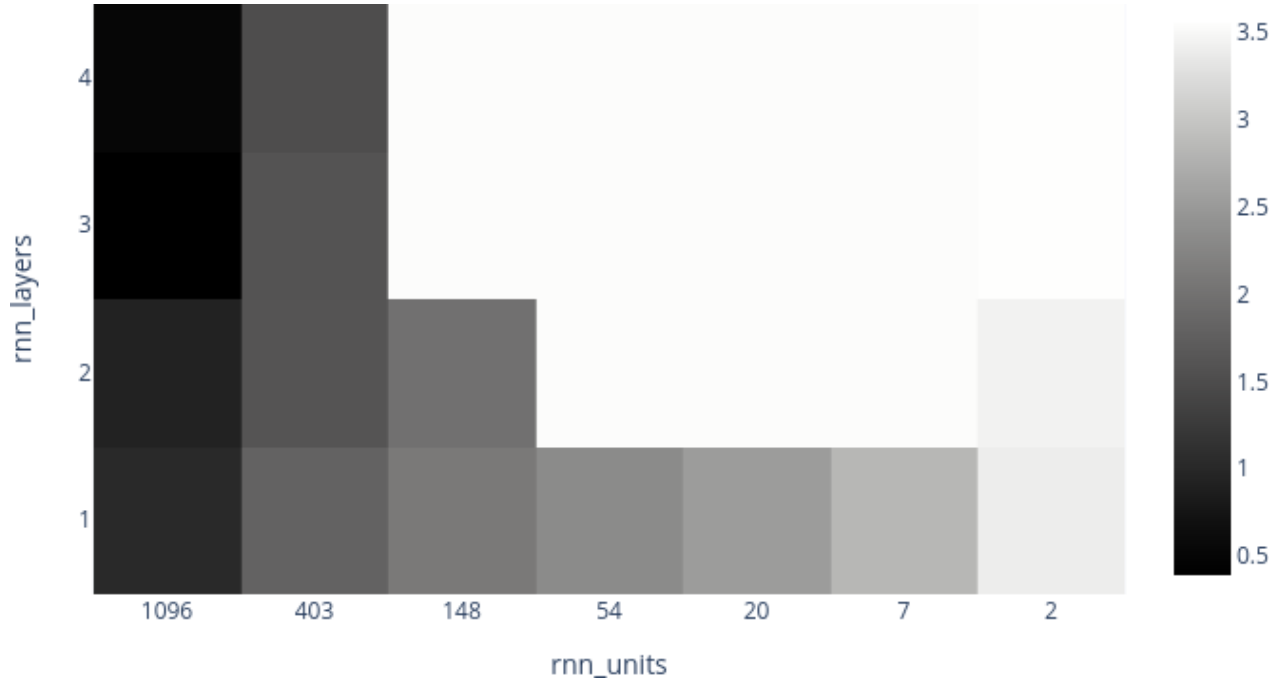

Рис. 3.3 Залежність помилки від кількості шарів в мережі та від кількості шарів

Як бачимо, можна чітко виділити оптимальну кількість шарів та нейронів в шарі. Згідно з цими результатами, було обрано 3 шари, та 1096 нейронів в кожному шарі.

Також, варто звернути увагу на розмір ембедінгу. Наразі воно дорівнює 200, втім, згідно з дослідженнями [10], він повинен бути ближчим до кореню 4-го ступеню від кількості елементів. В нашій моделі 97 символів, тож насправді має сенс використати 3 або 4. Але спершу спробуємо взагалі його прибрати. В такому разі необхідно змінити розмірність вхідних даних, адже GRU очікує на 1 вимір більше, ніж подається першопочатково:

```
def build_model(
    vocab_size, embedding_dim, rnn_layers, rnn_units,
batch_size, stateful=True
):
    model = tf.keras.Sequential(
```

```
        [
                    tf.keras.layers.Reshape((-1, 1),
                            batch_input_shape=(batch_size,
None)),
        ]
        + [
            tf.keras.layers.GRU(
                rnn_units,
                return_sequences=True,
                stateful=stateful,
                recurrent_initializer="glorot_uniform",
            )
            for _ in range(rnn_layers)
        ]
        +
[tf.keras.layers.TimeDistributed(tf.keras.layers.Dense(256))]
    )
    model.build()
    return model
```

В результаті отримаємо наступну модель:

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
reshape (Reshape)	(512, None, 1)	0
gru (GRU)	(512, None, 1024)	3154944
gru_1 (GRU)	(512, None, 1024)	6297600
gru_2 (GRU)	(512, None, 1024)	6297600

dense (Dense) (512, None, 97) 99425

Total params: 15,849,569

Trainable params: 15,849,569

І приклад результату:

За ва пі песьк сріти. — дрйре с ноцве ів оаі? я тубл, Ро дашв

З пинрнра са вищзли с

Чр нейй хощ пек доеитью,

Щи к пи чосожй д пндочв, па щл біс, Нрві в мвлнле —

Секндн- іебу з см пошис слсдн с н ссму зрзеиіє,

Ярл ше щ зме о плщерьне,

Се хьмл цуне порюв птч ноче м

Ма виннп,

Па у се селихел.

А ч са зрлеран. щ ое зк Риб с єхрдйль

Зеди Уоиа со ци поо

Як бачимо, ембедінг є дуже важливим для успішної роботи моделі.

Тепер порівняємо із моделлю, де ембедінг присутній, але розмірність простору результатів зменшено до 4-х:

Model: "sequential"

Layer (type) Output Shape Param #

embedding (Embedding) (64, None, 256) 24832

gru (GRU) (64, None, 512) 1182720

gru_1 (GRU) (64, None, 512) 1575936

gru_2 (GRU) (64, None, 512) 1575936

dense (Dense) (64, None, 97) 49761

Total params: 4,409,185

Trainable params: 4,409,185

Non-trainable params: 0

Діточки і хресті

Розіб'є доглядавтую прії.

За що мене байдуже!

За ними жнива!

Сама хочеш – лихо мріє.

Мій батько ж моє! серце плаче,

Мов парубками

Козу з брандірою чорнують.

Нехай мій синочку, сумують,

Великий задумчеко...

Минули літа молодії,

Таки будуть водою,

І не бачили й собі з ним,

З дорогом

До гімности щебетата,

Темному полюбила.

Мені їх наші козак?

Добре намисто, обнімати,

Проки

Як бачимо, результат не гірший, а, можливо, навіть кращий, ніж коли ембедінг в 50 раз більший!

Перевіримо кілька можливих розмірностей ембедінгів при кількості шарів, та кількості нейронів в шарі, що показали найкращі результати:

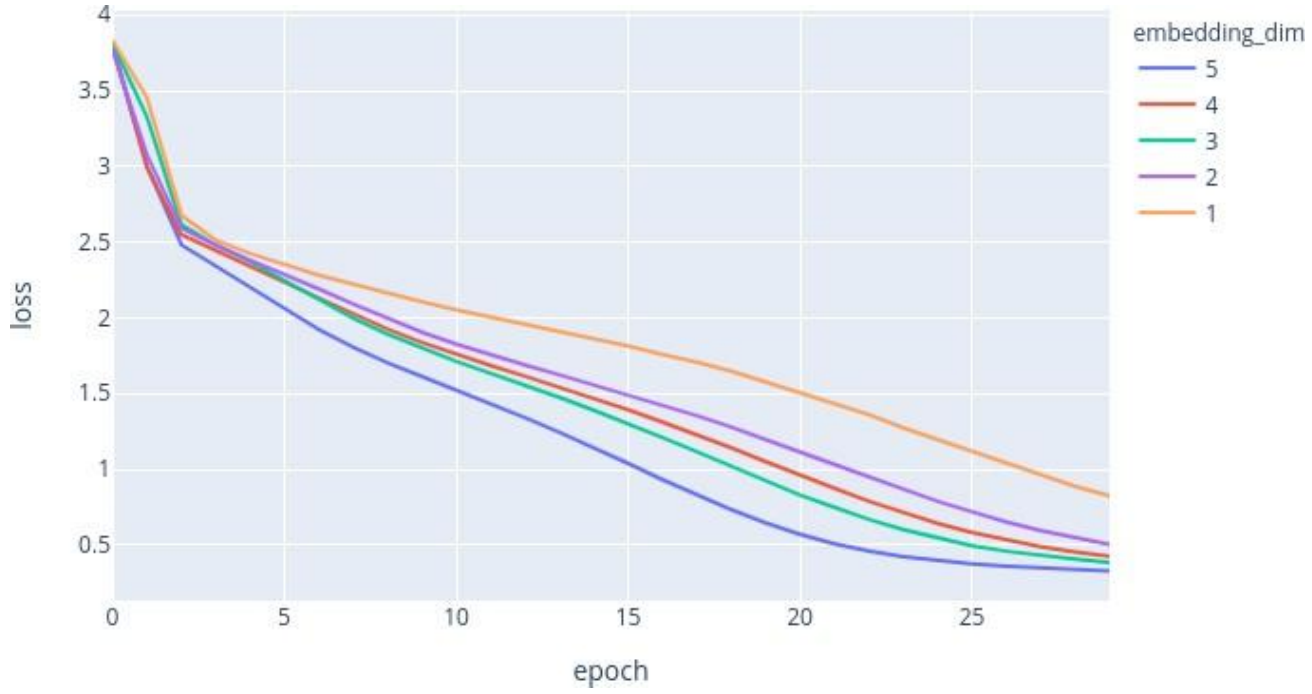

Рис. 3.4 Залежність помилки від розмірності ембедінгу

Можна побачити, що в такому випадку цілком вистачає ембедінгу з розміром 2, і що вибір розміру ембедінгу не сильно впливає на результат. Це пов’язано з тим, що множина символів містить всього близько 100 елементів.

3.2.5 Результат

Використовуючи наступні гіперпараметри:

Кількість рекурентних шарів: 3

Кількість нейронів в кожному шарі: 1096

Розмірність ембедінгу: 2

Кількість епох: 100

Збільшимо кількість епох, протягом яких модель тренуватиметься, та додамо ранню зупинку:

```
checkpoint_dir = "../checkpoints"
checkpoint_prefix = os.path.join(checkpoint_dir,
"rnn_3_ckpt_{epoch}")

checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_prefix, save_weights_only=True
)
callbacks = [
    checkpoint_callback,
    tf.keras.callbacks.EarlyStopping(monitor="loss",
patience=3),
]

model = build_model(
    vocab_size=len(vocab),
    embedding_dim=2,
    rnn_layers=3,
    rnn_units=1096,
    batch_size=BATCH_SIZE,
)
```

```
model.compile(optimizer="adam", loss=loss)
history = model.fit(dataset, epochs=100, callbacks=callbacks)
```

Epoch 1/100

83/83 [==============================] - 16s 167ms/step - loss: 4.3182

Epoch 2/100

83/83 [==============================] - 14s 167ms/step - loss: 3.3688

...

Epoch 78/100

83/83 [==============================] - 15s 172ms/step - loss: 0.2761

Epoch 79/100

83/83 [==============================] - 15s 172ms/step - loss: 0.2780

Epoch 80/100

83/83 [==============================] - 15s 172ms/step - loss: 0.2773

Epoch 81/100

83/83 [==============================] - 15s 172ms/step - loss: 0.2746

Epoch 82/100

83/83 [==============================] - 15s 172ms/step - loss: 0.2800

Epoch 83/100

83/83 [==============================] - 15s 172ms/step - loss: 0.2791

Epoch 84/100

83/83 [==============================] - 15s 172ms/step - loss: 0.2761

Бачимо, що модель зійшлась після 84 епох. Перевіримо результат роботи:

Діточки,

Мережаю, вирослаю правдивши,

Добре погуляєм! прокажив.

І окрає ми заходимось!.."

У такемний час поле

Котиться, чорноброві,

Запорожець. Ну, чудесный стан козацькая,".

"Та що мені, Тяжко вам скажі?

Ледве веселе прилине,

А вчисть у хаті и городівпанки

Петь одна, откреже!

Трохи не розважає —

Чого болите старому

На руку сихий дід

Надшу тихую само забуду.

І добро тихо твоя горе!

Ти зля

Можна побачити, що досі не всі слова є коректними, і цілісний сенс речень зазвичай відсутній. Втім, модель досить непогано навчилась наслідувати українську мову.

3.3 Трансформери

Для прикладу роботи трансформерів буде використано модель, яку нещодавно було опубліковано. Це GPT-2, що було натреновано на 4040 українських книгах.

Для використання моделі, перш за все необхідно завантажити токенізатор та модель.

```
from transformers import AlbertTokenizer, GPT2LMHeadModel

tokenizer =
AlbertTokenizer.from_pretrained("Tereveni-AI/gpt2-124M-uk-fictio
n")
model =
GPT2LMHeadModel.from_pretrained("Tereveni-AI/gpt2-124M-uk-fictio
n", tokenizer_class="Tereveni-AI/gpt2-124M-uk-fiction")

input_ids = tokenizer.encode("Були собі гайдамаки, без дому, без
віри,", add_special_tokens=False, return_tensors='pt')
```

Після того, як модель було завантажено, необхідно її використати.

Є чимало способів генерації тексту, маючи модель. Якщо використати найпростіший спосіб — використати жадібний пошук, то в результаті велика вірогідність отримати одне речення, що повторюється нескінченно:

```
outputs = model.generate(
    input_ids, max_length=500,
)
print(tokenizer.decode(outputs[0]))
```

Наталка йшла до нього, а він, не звертаючи уваги на її слова, йшов за нею. Вона не могла не помітити, що він не звертає на неї уваги. Вона не могла не помітити, що він не звертає на неї уваги. Вона не могла не помітити, що він не звертає на неї уваги. Вона не могла не помітити, що він не звертає на неї уваги.

Вона не могла не помітити, що він не звертає на неї уваги. Вона не могла не помітити, що він не звертає на неї уваги. Вона не могла не помітити, що він не звертає на неї уваги. Вона не могла не помітити, що він не звертає на неї уваги. Вона не могла не помітити, що він не звертає на неї уваги. Вона не могла не помітити, що він не звертає на неї уваги. ...

Згідно з дослідженнями [11], найкращі результати показуть наступні параметри: do_sample=True, top_k=50, top_p=0.95:

```
outputs = model.generate(
    input_ids, max_length=500,
    do_sample=True,
    top_k=50,
    top_p=0.95
)
print(tokenizer.decode(outputs[0]))
```

Наталка йшла до нього, але він не слухав. До побачення, пані, — сказав він з усмішкою. Що ж, я вже їду додому. — Вона хотіла повернутися до своєї улюбленої теми — посміхатися чи навіть поцілувати. Та вона цього не зробила. Вона була занадто схвильована, надто збентежена, щоб знову заплакати, коли він побачив її. Нарешті вони попрощались і рушили додому. Вона йшла йому назустріч. Їй хотілося тільки одного: тільки одного: щоб він міг повернутися. Вона не хотіла повертатися, вона не могла піти... Він зупинився біля ліжка, поклавши голову на подушку. Вона була така самотня, така холодна, що йому здалося, наче він заснув. Вона кохала його. І вона була така нещасна!..

Неймовірний результат! Текст є цілісним, має сенс та контекст.

Модель можна дотренувати, і, таким чином, наблизити результат до бажаного стилю чи напряму. Для прикладу, це можна зробити із датасетом із творів Шевченка.

Для цього необхідно використати скрипт, який можна знайти за посиланням https://github.com/huggingface/transformers/blob/master/examples/pytorch/language-modeling/run_clm.py. Для цього використаємо наступні параметри:

- --model_name_or_path

  Цим параметром необхідно вказати передтреновану модель, яку необхідно донавчити

- --train_file

  Цим параметром вказується шлях до файлу, на якому відбувається навчання.

- --output_dir

  За допомогою цього параметра вказуємо, в яку теку необхідно зберегти дотреновану модель

- --per_device_train_batch_size

  Цей параметр вказує кількість елементів в одному батчі. Чим більший цей параметр, тим більше відепам’яті необхідно. В дослідженні використовувалась відеокарта Nvidia GTX 1080 Ti, і на значеннях, що перевищували 2 пам’яті на ній вже не вистачало.

  Після виконання скрипта в теці, яку було вказано за допомогою параметра output_dir знаходитимуться наступні файли:

  .rw-r--r-- 65 astadnik 27 May 10:29 added_tokens.json

  .rw-r--r-- 450 astadnik 27 May 10:29 all_results.json

  .rw-r--r-- 820 astadnik 27 May 10:29 config.json

  .rw-r--r-- 510M astadnik 27 May 10:29 pytorch_model.bin

  .rw-r--r-- 241 astadnik 27 May 10:29 special_tokens_map.json

  .rw-r--r-- 1.5M astadnik 27 May 10:29 spiece.model

  .rw-r--r-- 460 astadnik 27 May 10:29 tokenizer_config.json

  .rw-r--r-- 450 astadnik 27 May 10:29 train_results.json

  .rw-r--r-- 510 astadnik 27 May 10:29 trainer_state.json

  .rw-r--r-- 2.4k astadnik 27 May 10:29 training_args.bin

3.3.1 Результат

Щоб використати дотреновану модель, спочатку завантажимо дотреновану модель:

```
tokenizer = AlbertTokenizer.from_pretrained("/tmp/test-clm")
model = GPT2LMHeadModel.from_pretrained(
    "/tmp/test-clm",
    tokenizer_class="/tmp/test-clm",
)
device = "cuda:0" if torch.cuda.is_available() else "cpu"
```

Після цього створимо початкові дані, та підготуємо їх для передачі до моделі:

```
input_ids = tokenizer.encode(
    "Наталка йшла",
    add_special_tokens=False,
    return_tensors="pt",
)
```

Нарешті, згенеруємо текст.

```
outputs = model.generate(
    input_ids, max_length=500,
    do_sample=True,
    top_k=50,
    top_p=0.95
)

print(tokenizer.decode(outputs[0], skip_special_tokens=True))
```

Наталка йшла пішки. Ішла, дивилась. А вже під селом надибали хату та й накраяли. У селі й досі з хат повитягали та закували. Аж у хаті, в полі, скрізь, скрізь. у всій хаті поличка змальована, повиті фарбами. у церкві, кругом стола укрита, і кругом стола. і на горі, і поза горою кругом стола, і на призьбі, і коло тину

на помості. У хаті аж під горою на тину, за тином стоять собі три копи та й розмовляють. Неначе й не знає, хто й що робить. Неначе й не знає, де вони беруть гроші і де вони ховаються. І не знають, де ховаються. Мов і не знають, де їх убито. Аж ось чує Петро і чує, що в хаті непорадна вечеря. Петро встав з постелі, надів на коліна, став коло тину, і його очі виглянули, заблищали. А Петро, не вміщаючись, пішов на гору до тину. Через тин ішов Петро, а потім та й пішов. Коло тину знов стала перед ним молодиця. Молодиця встала, а Петро йде та й ходить по тину. А тин іде та й іде, минає тин та й минає, минає, минає, минає, та й таки минає, аж ось він минає. Як вернесь, то й та хата буде, як і перше. "Ой боже мій, боже мій!" — і приговорює. "За що мене виганяти? За що мене виганяти?" — шепче. "За що мене виганяти?" — "За те, що я втікала.!" "Ой боже мій, боже мій, боже, боже мій... Та й іде, іде, іде... іде, та все минає... іде, та все минає". Та й іде та й іде. Заходить у хату, пристає до тину, та на тину пристає. Аж і за тобою в полі село; а з села — Петро. Іде, іде та й іде... а тин не минає, минає, а тин і з-за тину. "Із-за тину? А з-за лісу? А з поля? Чи з-за річки?!.." — і тин біжить... і тин скаче, та з-за тину біжить, та на тину біжить, та з-за тину біжить

Текст став більш схожим на стиль Шевченка — оточення, місце, втім, від цього трохи постраждав сенс. Через те, що навчання відбулось на невеликому датасеті, модель перевчилась на ньому.

РОЗДІЛ 4

ПОРІВНЯННЯ РЕЗУЛЬТАТІВ

4.1 Порівняння та аналіз моделей

Для порівняння результатів створимо програму, яка об’єднає всі методи, та надасть зручний інтерфейс для їх використання. Перш за все, необхідно отримати в якості вхідних даних від користувача бажану модель, кількість символів, яку він хоче згенерувати, та початок речення:

```
import argparse
def parse_args():
    parser = argparse.ArgumentParser()
    parser.add_argument("-l", "--length", default=100, type=int,
                        help="Length of the text")
    parser.add_argument("-i", "--input", default='Бувало',
                        help="Input text to start from")
    parser.add_argument("-m", "--model", default='GPT2',
                        choices=['markov', 'RNN', 'GPT2'],
help="Model to generate text")
    return parser.parse_args()
```

Після цього необхідно імплементувати по черзі кожен метод. Всі вони вже реалізовані, натреновані, тож досить просто їх використати.

Втім, перед цим було б корисно вимкнути попередження, які можуть показуватись фреймворками у випадку, якщо щось налаштовано некоректно, щоб не сплутати ці повідомлення з результатом роботи.

```
def disable_warnings():
    import os
```

```
    import logging
    logging.disable(logging.CRITICAL)
    os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
```

Ланцюги Маркова

```
from collections import defaultdict
import random

def markov_chain(text):
    words = text.split()
    collocation_dict = defaultdict(list)
    for current_word, next_word in zip(words[0:-1], words[1:]):
        collocation_dict[current_word].append(next_word)
    collocation_dict = dict(collocation_dict)
    return collocation_dict

def generate_sentence(chain, word_count, cur_word=None):
    cur_word = cur_word or random.choice(list(chain.keys()))
    sentence = cur_word.capitalize()
    for _ in range(word_count-1):
        next_word = random.choice(chain[cur_word])
        sentence += ' ' + next_word
        cur_word = next_word

    sentence += '.'
    return sentence

def generate_markov(input: str, length: int):
    from classical.markov_chain import generate_sentence,
markov_chain
    with open("data/Shevchenko/Shevchenko_no_latin.txt", "r") as
f:
        text = f.read()
    # text = text.translate(dict.fromkeys(map(ord,
'\n!"\'(),-.---:;?'), ' '))
    try:
```

```
        return generate_sentence(markov_chain(text), length,
input.lower())
    except KeyError:
        print('You should provide a word which is present in the
dataset')
        return ''
```

Нейронні мережі

```
def generate_rnn(input: str, length: int):
    import tensorflow as tf
    from models.RNN import embedding_dim, rnn_layers, rnn_units,
build_model, generate_text, get_vocab
    vocab = get_vocab()
    model = build_model(len(vocab), embedding_dim,
                        rnn_layers, rnn_units, batch_size=1)
    model.load_weights(tf.train.latest_checkpoint(
        'best_model_checkpoints'))
    model.build(tf.TensorShape([1, None]))
    return generate_text(model, input, vocab, length)
```

Трансформери

```
def generate_GPT2(input: str, length: int):
    import torch
    from transformers import AlbertTokenizer, GPT2LMHeadModel
    tokenizer =
AlbertTokenizer.from_pretrained("GPT2/fine_tuned")
    model = GPT2LMHeadModel.from_pretrained(
        "GPT2/fine_tuned",
        tokenizer_class="GPT2/fine_tuned",
    )
    device = "cuda:0" if torch.cuda.is_available() else "cpu"
    input_ids = tokenizer.encode(
        input,
        add_special_tokens=False,
        return_tensors="pt",
```

```
    ).to(device)
    model.to(device)
    outputs = model.generate(
        input_ids, do_sample=True, num_return_sequences=1,
max_length=length
    )
    return tokenizer.decode(outputs[0],
skip_special_tokens=True)
```

Тепер для використання програми необхідно запустити її з відповідними параметрами. Для отримання списку параметрів та можливих значень, запустимо її з параметром --help:

❯ py generate_text.py --help

usage: generate_text.py [-h] [-l LENGTH] [-i INPUT] [-m {markov,RNN,GPT2}]

optional arguments:

-h, --help show this help message and exit

-l LENGTH, --length LENGTH

Length of the text

-i INPUT, --input INPUT

Input text to start from

-m {markov,RNN,GPT2}, --model {markov,RNN,GPT2}

Model to generate text

Запустимо по черзі всі методи:

Ланцюги Маркова

Рекурентні нейронні мережі

Трансформери

Як ми можемо побачити, найкращий результат, звісно, показує модель трансформера. Текст є цілком зв’язним, слова є коректними. Втім, для використання цього методу необхідно мати досить потужну відеокарту:

```
❯ nvidia-smi
```

```
Mon May 31 20:25:33 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 465.27       Driver Version: 465.27       CUDA Version: 11.3     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA GeForce ...  Off  | 00000000:41:00.0 Off |                  N/A |
| 27%   37C    P2    82W / 250W |   3122MiB / 11176MiB |     48%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A       696      G   /usr/lib/Xorg                      11MiB |
|    0   N/A  N/A    931985      C   python3                          1513MiB |
|    0   N/A  N/A   2705393      C   ...irtualenvs/edu/bin/python     1593MiB |
+-----------------------------------------------------------------------------+
```

Використання моделі займає 1513 мегабайт відеопам'яті. Гіршим за результатами, є підхід з використанням рекурентних нейронних мереж. Чимала частина слів є некоректними, хоча це з певним наближенням має вигляд української мови. Також, саме в цьому методі найкраще відчувається стиль Шевченка, адже модель трансформерів була навчена більшою частиною на книгах інших авторів.

Своєю чергою, ланцюги Маркова дають найгірший результат, текст є найбільш схожим на випадкову послідовність слів. Втім, також, вони і є найшвидшими:

- Ланцюги Маркова — 0.091 секунда
- Рекурентні нейронні мережі — 5.23 секунди
- Трансформери — 10.02

  Відповідно, якщо важливою є не якість тексту, а обсяги та час, тоді має сенс використати модель ланцюгів Маркова.

ВИСНОВКИ

В межах цієї роботи було розроблено застосунок для генерації унікального україномовного тексту заданої довжини, що наслідує стиль та тему тексту-зразка. Надано вибір між трьома моделями для генерації, що дозволяє обирати між кращою швидкістю або якістю. Створено консольний та графічний інтерфейс для використання.

Для досягнення цілі було виконано наступні кроки:

- Проаналізовано наявні підходи до генерації тексту, обґрунтовано доцільність теми дослідження, обрано 3 моделі:
    - Ланцюги Маркова
    - Рекурентні нейронні мережі
    - Модель трансформера (GPT-2)
- Реалізовано та навчено необхідні моделі, проведено їх параметричний синтез.
- Порівняно якість та час роботи моделей, пояснення доцільність використання кожної з них.
- Створено зручний для користувача інтерфейс для використання моделей.

Описано теоретичне підґрунтя роботи, принцип роботи використаних моделей, обґрунтовано вибір гіперпараметрів, та пояснено їх вплив на результат.

Створене програмне забезпечення може бути використано для генерації тексту-замінника, наприклад, веброзробником для наповнення макетів сайту контентом, подібним до справжнього, необхідної мови та напрямленості.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Генератор українського тексту [Електронний ресурс]. – Режим доступу: https://dzhedzhalyk.com.ua/generated

2. Statistical and neural language models for the Ukrainian Language — Anastasiia Khaburska, 2020

3. Markov chain [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Markov_chain

4. Recurrent neural network [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Recurrent_neural_network

5. GPT-2 [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/GPT-2

6. GPT-2 Uk fiction [Електронний ресурс]. – Режим доступу: https://huggingface.co/Tereveni-AI/gpt2-124M-uk-fiction

7. Minsky's "And / Or" Theorem: A Single Perceptron's Limitations [Електронний ресурс]. – Режим доступу: https://alan.do/minskys-and-or-theorem-a-single-perceptron-s-limitations-490c63a02e9f

8. Multilayer perceptron [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Multilayer_perceptron

9. Attention Is All You Need — Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin

10. There is a theoretical lower bound for embedding dimension [Електронний ресурс]. – Режим доступу: https://datascience.stackexchange.com/a/65209

11. How to generate [Електронний ресурс]. – Режим доступу: https://huggingface.co/blog/how-to-generate

ДОДАТКИ

Додаток А

Код програми

```
from abc import ABC, abstractmethod
import argparse
import PySimpleGUI as sg
import tensorflow as tf
import numpy as np
from tqdm.auto import trange
from collections import defaultdict
import random

#####################################################################
################
#                                        MARKOV
#
#####################################################################
################


def markov_chain(text):
    words = text.split()
    collocation_dict = defaultdict(list)
    for current_word, next_word in zip(words[0:-1], words[1:]):
        collocation_dict[current_word].append(next_word)
    collocation_dict = dict(collocation_dict)
    return collocation_dict


def generate_sentence(chain, word_count, cur_word=None):
    cur_word = cur_word or random.choice(list(chain.keys()))
    sentence = cur_word.capitalize()
    for _ in range(word_count-1):
```

```
        next_word = random.choice(chain[cur_word])
        sentence += ' ' + next_word
        cur_word = next_word

    sentence += '.'
    return sentence


#####################################################################
################
#                                        RNN
#
#####################################################################
################


def
get_vocab(path_to_file="data/Shevchenko/Shevchenko_no_latin.txt"
):
    text = open(path_to_file,
"rb").read().decode(encoding="utf-8")
    return sorted(set(text))


def build_model(
    vocab_size, embedding_dim, rnn_layers, rnn_units,
batch_size, stateful=True
):
    model = tf.keras.Sequential(
        [
            tf.keras.layers.Embedding(
                vocab_size, embedding_dim,
batch_input_shape=[batch_size, None]
            ), *[
                tf.keras.layers.GRU(
                    rnn_units,
```

```
                    return_sequences=True,
                    stateful=stateful,
                    recurrent_initializer="glorot_uniform",
                )
                for _ in range(rnn_layers)],
tf.keras.layers.TimeDistributed(tf.keras.layers.Dense(256))]
    )
    model.build()
    return model


def generate_text(model, start_string, vocab, num_generate=400):
    char2idx = {u: i for i, u in enumerate(vocab)}
    idx2char = np.array(vocab)

    input_eval = [char2idx[s] for s in start_string]
    input_eval = tf.expand_dims(input_eval, 0)

    text_generated = []

    # Increase to choose less probable characters more often
    temperature = 1.0

    model.reset_states()
    for _ in trange(num_generate, leave=False):
        # Get predictions for the model
        predictions = model(input_eval)
        # Remove unccecessary dimension (because batch_size ==
1)
        predictions = tf.squeeze(predictions, 0)

        # Change the probabilities
        predictions = predictions / temperature
        # Choose the result
        predicted_id = tf.random.categorical(
            predictions, num_samples=1)[-1, 0].numpy()
```

```
        # Update input
        input_eval = tf.expand_dims([predicted_id], 0)

        text_generated.append(idx2char[predicted_id])

    return start_string + ''.join(text_generated)


embedding_dim = 2
rnn_layers = 3
rnn_units = 1096

################################################################
################
#                                  API CLASSES
#
################################################################
################


class Generator(ABC):
    @abstractmethod
    def generate(self, input: str, length: int):
        raise NotImplementedError(
            "You're using base class, please use a sublcass")


class Markov(Generator):
    def generate(self, input: str, length: int):
        with open("data/Shevchenko/Shevchenko_no_latin.txt",
"r") as f:
            text = f.read()
        # text = text.translate(dict.fromkeys(map(ord,
'\n!"\'(),-.---:;?'), ' '))
        try:
```

```
            return generate_sentence(markov_chain(text), length,
input.lower())
        except KeyError:
            print('You should provide a word which is present in
the dataset')
            return ''


class RNN(Generator):
    def generate(self, input: str, length: int):
        import tensorflow as tf
        vocab = get_vocab()
        model = build_model(len(vocab), embedding_dim,
                            rnn_layers, rnn_units, batch_size=1)
        model.load_weights(tf.train.latest_checkpoint(
            'best_model_checkpoints'))
        model.build(tf.TensorShape([1, None]))
        return generate_text(model, input, vocab, length)


class GPT2(Generator):
    def generate(self, input: str, length: int):
        import torch
        from transformers import (
            PreTrainedTokenizer, PreTrainedModel,
AlbertTokenizer, GPT2LMHeadModel)
        tokenizer =
AlbertTokenizer.from_pretrained("GPT2/fine_tuned")
        model = GPT2LMHeadModel.from_pretrained(
            "GPT2/fine_tuned",
            tokenizer_class="GPT2/fine_tuned",
        )
        assert(isinstance(tokenizer, PreTrainedTokenizer))
        assert(isinstance(model, PreTrainedModel))
        device = "cuda:0" if torch.cuda.is_available() else
"cpu"
```

```
        input_ids = tokenizer.encode(
            input,
            add_special_tokens=False,
            return_tensors="pt",
        ).to(device)
        model.to(device)
        outputs = model.generate(
            input_ids, do_sample=True, num_return_sequences=1,
max_length=length
        )
        return tokenizer.decode(
            outputs[0], skip_special_tokens=True)


def generator_factory(name: str):
    return {'GPT2': GPT2(), 'markov': Markov(), 'RNN':
RNN()}[name]

#####################################################################
################
#                                      TUI
#
#####################################################################
################


def parse_args():
    parser = argparse.ArgumentParser()
    parser.add_argument("-l", "--length", default=100, type=int,
                        help="Length of the text")
    parser.add_argument("-i", "--input", default='Бувало',
                        help="Input text to start from")
    parser.add_argument("-m", "--model", default='GPT2',
                        choices=['markov', 'RNN', 'GPT2'],
help="Model to generate text")
    return parser.parse_args()
```

```
# def main():
#     args = parse_args()
#     disable_warnings()
#     print(generator_factory(args.model).generate(args.input,
args.length))

#####################################################################
################
#                                        GUI
#
#####################################################################
################


def disable_warnings():
    import os
    import logging
    os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
    import tensorflow as tf
    logging.disable(logging.CRITICAL)
    tf.get_logger().setLevel('INFO')
    tf.autograph.set_verbosity(1)


def main():
    # sg.theme('DarkAmber')
    sg.theme('LightBlue3')
    # sg.theme('LightGray1')
    # sg.theme('LightGray')

    disable_warnings()
    methods = ['GPT2', 'markov', 'RNN']

    layout = [[sg.Col([[sg.T('Start of the text')],
[sg.In('бувало', key='-IN-')]]),
```

```
                sg.Col([[sg.T('Chars')],
                        [sg.Input('400', key='-NUM-',
enable_events=True, size=(5, 1))]])],
               [sg.Combo(values=methods, key='-MODEL-',
                         default_value=methods[2],
readonly=True)],
               [sg.Button('Generate', bind_return_key=True),
sg.Exit()]]

    window = sg.Window('Ukrainian text generation', layout,
                       return_keyboard_events=True)
    # window.bind('q', lambda event: window.destroy())

    # symbols_from_file =
'!"\'(),-.:;?---0123456789аАбБвВгГґҐдДеЕєЄжЖзЗиИіІїЇйЙкКлЛмМнНо
ОпПрРсСтТуУфФхХцЦчЧшШщЩъьыЫьЬэЭюЮяЯ'

    while 42:
        event, values = window.read()  # type: ignore
        if event in (sg.WINDOW_CLOSED, 'Exit', 'Escape:9'):
            break
        if (event == '-NUM-' and values['-NUM-']
                and values['-NUM-'][-1] not in ('0123456789')):
            window['-NUM-'].update(values['-NUM-'][:-1])
        if event == 'Generate':
            text = generator_factory(values['-MODEL-']
                                     ).generate(values['-IN-'],
int(values['-NUM-']))
            required_height = max((len(text) // 80) + 1,
text.count('\n') + 1)
            sg.Window('Generated text',
                      [[sg.Multiline(text, size=(80,
required_height),
                                     disabled=True)],
[sg.Exit()]]
                      ).read(close=True)
```

```
    window.close()


if__name__== "__main__":
    main()
```

Додаток Б

Результати конкурсу наукових робіт

Шифр роботи — “Цикада”.

Шифр роботи	к-ть авторів	Оцінка 1	Оцінка 2	Загальний бал
Ruva	2	100	100	200
Біодопуск	1	98	100	198
Love-miye	2	95	99	194
E-LEARNING	2	98	96	194
CDLBSW	2	98	92	190
Лідари	1	98	89	187
КОНСТРУКТОРИ	1	83	100	183
Оцінка фреймворків	1	93	90	183
Відновлення	1	81	99	180
Transport	2	91	89	180
В-ЗМЕД	1	85	94	179
Citybot	1	90	88	178
Виявлення шахраїв	1	83	85	168
Якість освіти	2	80	86	166
ZERO TWO	1	93	72	165
Cognit-Assist	1	80	80	160
Блискавки	1	83	73	156
Розповсюдження	1	74	81	155
sreifisseyab	1	67	87	154
Живі картинки	1	85	67	152
НЕЙРОМЕРЕ	1	75	77	152

ЖА				
Цикада	1	69	79	148
СУПЕР БОБЕР	1	76	68	144
Tiniovy Kit	1	73	69	142
Здоров’я населення	1	65	75	140
Великий Ух	1	89	50	139
OrgProg	1	70	68	138
Тестування геометричних знань	2	70	64	134
Рожевий фламінго	2	53	78	131
Цифрова анаконда	2	41	90	131
CryptoChat	2	75	54	129
ProjectFractal	1	58	70	128
ExchangeModule	1	62	65	127
Паводок	2	69	58	127
Межа-Сонце	1	45	67	112
Автоматизація олімпіад	1	67	40	107
Спеціальні мережі	1	68	39	107
Стемінг	1	50	56	106
Marchen	2	57	48	105
РОБОТ ТІМА	1	64	39	103
Облік продажів	1	38	60	98
Система контролю	1	46	49	95
GPS-трекінг	1	41	46	87
Birdy	2	41	38	79
Todo List	1	48	27	75
Enkel	1	30	40	70
Crowd buy	1	31	39	70
20/21-24	1	15	54	69
Bad Teleport	1	35	30	65
Web-сервіс	1	0	60	60

20/21-26	1	25	26	51

Додаток В

Акт впровадження

ЗАТВЕРДЖУЮ
Директор Державного підприємства
Український центр
розвитку інформаційних технологій
Міністерства освіти і науки України

В.Б.ПОЛІЩУК

17 грудня 2020 року

АКТ
впровадження результатів дослідження
студентів Київського національного університету імені Тараса Шевченка
Стадніка Андрія Валерійовича
на тему:
"Програмне забезпечення захисту авторських прав на зображення текстових матеріалів шляхом керованого шуму"

Комісія у складі:
голова комісії – Директор ДП УкрНЦ РІТ МОНУ Поліщук В.Б.;
члени комісії:
Провідний науковий співробітник Нетесін І.Є.,
Головний інженер Коконцев С.О.

Встановила та цим актом засвідчує, що нижчеперелічені результати досліджень:

1. Розроблена технологія ручного приховування тексту на важливих зображеннях за допомогою керованого шуму.
2. Розроблена технологія автоматичного приховування ключових слів на зображеннях за допомогою керованого шуму.

впроваджені в проекті стандарту компанії з «Захист інформації» в розділі «Захист текстової інформації».

Голова комісії
кандидат технічних наук В.Б..ПОЛІЩУК
Члени комісії
Кандидат фізико-математичних наук І.Є.НЕТЕСІН
С.О.КОКОНЦЕВ

17 грудня 2020 року

Додаток Г

Список публікацій

За результатами наукових досліджень, проведених у роботі, було підготовлено 2 публікації в збірнику матеріалів VII Східно-Європейська конференція “Математичні та програмні технології Internet of Everything”, 22-23.12.2020, Київ.

- Стаднік А.В., Шевченко В.Л. Захист зображення текстових матеріалів шляхом керованого шуму
- Стаднік А.В. Захист зображення текстових матеріалів шляхом керованого шуму