НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
«____» _____2021 р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Комп'ютерні системи та мережі»
спеціальності 123 «Комп'ютерна інженерія»

на тему: «Система генерації тематичних текстів на основі нейронних мереж»

Виконала: студентка 4 курсу, групи ІВ-71
Корнійчук Ольга Платонівна _____

Керівник Волокита А. М. _____

Консультант (нормоконтроль) Симоненко В. П. _____

Рецензент _____ _____

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.

Студентка Корнійчук О.П_____

Київ – 2021 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

«Комп'ютерні системи та мережі»

Спеціальність 123 «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ Сергій СТІРЕНКО
«__» _____2021 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студентки

Корнійчук Ольги Платонівни

1. Тема проекту *Система генерації тематичних текстів на основі нейронних мереж*

керівник проекту *Волокита Артем Миколайович*

Затверджені наказом по університету від «___» _____ 2021 року №_____

2. Термін здачі студентом закінченого проекту (роботи) _____

3. Вихідні дані до проекту *технічна документація, теоретичні та статистичні дані;*

4. Зміст розрахунково-пояснювальної записки а*наліз предметної області, огляд існуючих підходів та експериментів, розробка системи, результати розробки та тестування моделі*

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо): *структурна схема нейронної мережі, принципова схема алгоритму роботи програми, функціональна схема класів*

6. Консультанти проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Симоненко В.П.		

7. Дата видачі завдання_____

Календарний план

№ п/п	Найменування етапів дипломного проекту	Терміни виконання етапів проекту	Примітки
1.	*Затвердження теми проєкту*	*10.12.2020-15.12.2020*	
2.	*Вивчення та аналіз завдання*	*15.12.2020-15.03.2021*	
3.	*Написання вступної частини та огляду предметної частини*	*15.03.2021-25.03.2021*	
4.	*Розробка архітектури системи*	*25.03.2021-5.04.2021*	
5.	*Програмна реалізація системи*	*5.04.2021-15.04.2021*	
6.	*Виправлення помилок*	*15.04.2021-20.05.2021*	
7.	*Оформлення документації дипломної роботи*	*25.04.2021*	
8.	*Передзахист*	*22.05.2021*	

9.	*Захист*	*16.06.2021*	

Студентки Корнійчук Ольги Платонівни _____

Керівник проєкту Волокита Артем Миколайович _____

Анотація

У бакалаврському дипломному проєкті реалізовано систему генерації тематичного тексту на основі нейронних мереж, що здатна генерувати тематичний текст на базисі п’ятикнижжя Мойсеєвого.

Програма дозволяє створювати нові текствоі послідовності заданої довжини. Програмний продукт був створений на мові програмування Python 3.9 в інтегрованому середовищі розробки Jupyter.

Annotation

The bachelor's degree project implements a system of thematic text generation based on neural networks, which is able to generate a thematic text based on the Pentateuch of Moses.

The program allows you to create new text sequences of a given length. The software product was created in the Python 3.9 programming language in the Jupyter integrated development environment.

Опис альбому

до дипломного проєкту

на тему: «Система генерації тематичних текстів на основі нейронних мереж»

Київ – 2021

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів1	№ екземпляру	Примітки
			Документація загальна			
			Розроблена заново			
1	А4	ІАЛЦ.467200.001 ОА	Опис альбому	1		
2	А4	ІАЛЦ.467200.002 ТЗ	Технічне завдання	4		
3	А4	ІАЛЦ.467200.003 ПЗ	Пояснювальна записка	77		
4	А4	ІАЛЦ.467200.004 Д1	Додаток А	1		
5	А4	ІАЛЦ.467200.005 Д2	Додаток Б	1		
6	А4	ІАЛЦ.467200.006 Д3	Додаток В	1		
7	А4	ІАЛЦ.467200.007 Д4	Додаток Г	8		

Зм	Арк.	№ докум	Підп.	Дата	***ІАЛЦ.467200.001 ОА***		
Розроб.		*Корнічук О.П.*			*Система генерації тематичних текстів на основі нейронних мереж* *Опис альбому*	**Літ.**	**Арк.** 1 / **Аркушів** 1
Первір.		*Волокита А.М.*					
Н.Контр.		*Симоненко В.П.*				***КПІ ім. Ігоря Сікорського ФІОТ, ІВ-71***	
Затверд.		*Волокита А.М.*					

Технічне завдання
до дипломного проєкту

на тему: «Система генерації тематичних текстів на основі нейронних мереж»

Київ — 2021

ЗМІСТ

1. НАЙМЕНУВАННЯ І ОБЛАСТЬ ЗАСТОСУВАННЯ

Найменування: система генерації тематичних текстів на основі нейронних мереж.

Область застосування: альтернатива існуючим системам и програмам генерації тематичних текстів.

2. ПІДСТАВИ РОЗРОБКИ

Підставою для розробки є завдання на виконання роботи кваліфікаційно-освітнього рівня «бакалавр комп'ютерної інженерії», затверджене кафедрою обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут».

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою цього проєкту є розгляд різних видів систем генерації тематичнх текстів на основі природніх мов та створення власної системи, що використовує сучасні розробки в області обробки природніх мов за допомогою нейронних мереж.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом розроки є науково-технічна література з теорії та практики основ операційних систем, технічна документація, публікації в періодичних виданнях, довідники, публікації в Інтернеті з даних питань.

5. ТЕХНІЧНІ ВИМОГИ

5.1 Вимоги до розробленого продукту

Далі надано основні вимоги до проєкту:

- Система має надавати можливості генерувати текст.
- Помилка при навчанні має бути менша за 50%.
- Система має надавати можливість створити нейронну мережу з нуля.
- Система має надавати можливість завантажувати модель з диска.

5.2 Вимоги до програмного забезпечення

- Операційна система Linux;
- Мова програмування Python 3.9.
- Встановлена бібліотека tensorflow-gpu

5.3 Вимоги до апаратної частини

- Операційна система Linux 5.12.5 або новіша;
- Вільної пам’яті на комп’ютері більше ніж 1024 МБ;
- Відеокарта NVIDIA з підтримкою CUDA.

6. ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Вивчення літератури	15.12.2020
Складання і узгодження технічного завдання	25.01.2021
Розробка компонентів цільової системи	20.02.2021
Тестування компонентів роботи	15.03.2021
Виправлення помилок	15.04.2021
Оформлення документації до дипломної роботи	19.05.2021

Пояснювальна записка

до дипломного проєкту

на тему: «Система генерації тематичних текстів на основі нейронних мереж»

Київ – 2021

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

NLP	(Natural Language Processing) Обробка натуральної мови
NLG	(Natural Language Generation) Генерація природної мови
NTG	(Neural Text Generation) Генерування нейронного тексту
NN	(Neural Network) Нейронна мережа
NNLM	(Neural Network Language Model) Мовна модель нейронної мережі
RNNLM	(Recurrent Neural Network Language Model) Рекурентна мовна модель нейронної мережі
RNN	(Recurrent Neural Network) Рекурентна нейронна мережа
LSTM	(Long Short-term Memory) Довга короткочасна пам'ять
GRU	(Gated Recurrent Unit) Керований рекурентний блок
SS	(Scheduled Sampling) Запланована вибірка
MDP	(Markov Decision Process) Процес прийняття рішення Маркова
BRA	(Bootstrapped Ranking Activation) Презавантажена градаційна активація
DCNN	(Deep Convolutional Neural Network) Глибока згорткова нейронна мережа
ANN	(Artificial Neural Networks) Штучні нейронні мережі
MLE	(Maximum Likelihood Estimation) Максимальна оцінка ймовірності
GAN	(Generative Adversarial Network) Генеративна конкурентна мережа
API	(Application Programming Interface) Інтерфейс програмування програм

ВСТУП

З плином часу розвиток нейронних мереж займає дедалі більшу частину сучасних розробок у галузі штучного інтелекту. Значну частину цього розвитку становлять генеративні нейронні мережі. Однією з таких мереж є система генерації тексту на основі природніх мов.

Метою цього дипломного проєкту є:

1) Аналіз предметної області, а саме існуючих підходів до генерації тематичного тексту на основі природніх мов.
2) Огляд вже створених та функціонуючих систем, що використовують сучасні методики та алгоритми.
3) Розробка нової системи генерації тематичних текстів за допомогою нейронних мереж.
4) Огляд і тестування створеної системи.

Дипломний проєкт складається зі вступу, чотирьох розділів, висновків до кожного з них, і з загального висновку до роботи.

У першому розділі проведено детальний аналіз предметної області, оглянуто принципи роботи нейронних мереж та підходів до генерації тематичних текстів.

У другому розділі оглянуто вже існуючі системи генерації текстів на основі нейронних мереж та описано існуючі дослідження в цій темі, а також надано й обґрунтовано вибір засобів реалізації проєкту.

У третьому розділі наведено інформацію щодо деталей розробленої системи та оглянуто її структуру.

У четвертому розділі приведено опис обраних алгоритмів та отриманиї результатів від експериментів зі створеною системою, а також надано інформацію щодо використаних мов та бібліотек.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

У даному розділі буде проведено аналіз в області нейронних мереж. А саме: розглянуто різні види генерації природного тексту, їх переваги та недоліки. Також буде проведений загальний огляд нейронних мереж типу LSTM.

1.1. Загальний огляд нейронних мереж

Штучні нейронні мережі - це дуже потужні обчислювальні моделі, натхненні мозком. Вони застосовуються в різних сферах, таких як обчислювальна техніка, медицина, інженерія, економіка та багато інших. Штучна нейронна мережа базується на теорії оптимізації.

Штучна нейронна мережа - це обчислювальна модель, натхненна функціонуванням людського мозку. Така складається з набору штучних нейронів (відомих як одиниці обробки), які взаємопов'язані з іншими нейронами, які залежать від ваги нейронної мережі. Ці ваги представляють зв'язок між нейронами, які визначають вплив одного нейрона на інший [1].

Світ обчислень може багато виграти від нейронних мереж. Їх здатність вчитися на прикладах робить їх дуже гнучкими та потужними. Крім того, немає необхідності розробляти алгоритм для виконання конкретного завдання; тобто немає необхідності розуміти внутрішні механізми цього завдання. Вони також дуже добре підходять для систем реального часу через їх швидку реакцію та обчислювальний час, що обумовлено їх паралельною архітектурою.

Перший штучний нейрон був вперше запропонований в офіційній моделі в 1943 році МакКаллоком і Піттсом. Вони довели, що ця модель нейрона здатна виконувати будь-яку обчислювальну функцію,

використовуючи кінцеве число штучних нейронів та синаптичних ваг, що регулюються. Дані математично обробляються та передають результат нейронам наступного шару. Вихід мережі забезпечується нейронами останнього шару.

J-нейрон у прихованому шарі обробляє вхідні дані шляхом: обчислення зваженої суми та додавання терміна «упередження» відповідно до:

$$\text{net}_j = \sum_{i=1}^{m} X_i * w_{ij} + \theta_j \quad j=(1,2,3,n)[2] ,$$

де xi -- вхідні дані;

wij -- вага;

θj -- упередження.

Нейронні мережі здатні адаптуватися до даних; нейронні мережі здатні узагальнювати навіть тоді, коли вхідні дані містять шум або відсутні значення (навчена мережа має можливість правильно заповнювати значення, не впливаючи на прогноз); нейронні мережі діють як універсальне наближення для довільної неперервної функції з довільною точністю.

Рис. 1.1 Будова штучного нейрона

ANN визначається трьома характеристиками:

1. Архітектура із зазначенням кількості шарів та вузлів у кожному з шарів.

2. Механізм навчання, застосований для оновлення ваг з'єднань.

3. Функції активації, що використовуються в різних шарах [2].

Розглянуто згладжений варіант вищезазначеного нейрона, для якого замість функції sgm використовуємо плавну монотонно зростаючу функцію sgm, яка змінюється від нуля до одиниці.

1.1.1. Навчання в ANN

Існує три основні парадигми навчання; навчання під наглядом, навчання без нагляду та навчання з підкріпленням. Зазвичай їх можна використовувати в будь-якому типі штучної архітектури нейронних мереж. Кожна парадигма навчання має безліч навчальних алгоритмів.

Нейронні мережі за кількістю шарів можуть поділятись на два основних типа:

1. Розмірний шар (одношаровий персептрон) є функцією від N дійсних змінних форми:

$$f(x_1, \ldots\ldots, x_N) = sgn(\sum_{i=1}^{N} w_i x_i - \theta) \quad ,$$

де x -- вхідні дані;

N -- кількість вхідних даних;

w -- ваги;

θ -- упередження.

2. Багатошаровий перцептрон:

Багатошаровий, що складається з трьох послідовних шарів: вхідного, прихованого та вихідного шарів. Кожна система в основному

являє собою тришарову систему, яка є вхідним шаром, прихованим шаром та вихідним шаром. Вхідний рівень має вхідні нейрони, які передають дані через синапси до прихованого шару, і аналогічно прихований рівень передає ці дані до вихідного шару через більше синапсів. У синапсах зберігаються значення, звані вагами, що допомагає їм маніпулювати вхідними та вихідними даними на різні рівні.

1.1.2. Контрольоване навчання

Контрольоване навчання - це техніка машинного навчання, яка встановлює параметри штучної нейронної мережі на основі навчальних даних. Завданням вивчення штучної нейронної мережі є встановлення значення її параметрів для будь-якого дійсного вхідного значення після побаченого вихідного значення.

Навчальні дані складаються з пар вхідних та бажаних вихідних значень, які традиційно представлені у векторах даних. Контрольоване навчання також можна назвати класифікацією, де ми маємо широкий спектр класифікаторів, кожен зі своїми сильними і слабкими сторонами.

Для вирішення даної проблеми контрольованого навчання слід розглянути різні етапи:

- На першому кроці визначається тип навчальних прикладів [3].
- На другому кроці створюється набір навчальних даних, який задовільно описує дану проблему.
- На третьому кроці описується зібраний набір навчальних даних у формі, зрозумілій для обраної штучної нейронної мережі.
- На четвертому кроці відбувається навчання, і після навчання можливо перевірити ефективність вивченої штучної нейронної мережі за допомогою тестового (валідаційного) набору даних. Тестовий набір даних складається з даних, які не були введені до штучної нейронної мережі під час навчання.

1.1.3. Неконтрольоване навчання

Неконтрольоване навчання - це техніка машинного навчання, яка встановлює параметри штучної нейронної мережі на основі даних та функції витрат, яку слід мінімізувати.

Функцією витрат може бути будь-яка функція, і вона визначається формулюванням завдання.

Неконтрольоване навчання здебільшого використовується в додатках, що вирішують проблеми статистичного моделювання, стиснення, фільтрації, поділу сліпих джерел та кластеризації. Під час неконтрольованого навчання необхідно визначити, як організовані дані. Цей спосіб відрізняється від контрольованого навчання та навчання з підкріпленням тим, що штучна нейронна мережа отримує на вхід лише приклади без маркування. Однією з поширених форм неконтрольованого навчання є кластеризація, коли ми намагаємось класифікувати дані в різних кластерах за їх подібністю.

Серед описаних вище моделей штучних нейронних мереж, самоорганізаційна карта Кохонена найчастіше використовується алгоритмом неконтрольованого навчання.

1.1.4. Навчання з підкріпленням

Навчання з підкріпленням - це техніка машинного навчання, яка встановлює параметри штучної нейронної мережі так, що дані зазвичай не вводяться, а генеруються в результаті взаємодії з навколишнім середовищем. Навчання з підкріплення стосується того, як штучна нейронна мережа повинна діяти в оточенні, щоб максимізувати довгострокову винагороду.

Більше того, навчання з підкріпленням часто використовується як частина загального алгоритму навчання нейронних мереж. Після того,

як визначена функція ефективності навчання, яку потрібно максимізувати, навчання з підкріпленням використовує декілька алгоритмів для пошуку методу, який забезпечує максимальну ефективність.

Разом з тим навчання з підкріпленням особливо добре підходить для проблем, які включають довгострокову винагороду. Цей спосіб був успішно застосований до різних проблем, включаючи управління роботами, телекомунікації та ігри, такі як шахи та інші послідовні завдання щодо прийняття рішень.

1.2. Застосування нейронних мереж

1.2.1. Класифікація

Нейронні мережі широко застосовується для класифікації мозкових пухлин на основі багатьох джерел інформації. Kailash D. та інші запропонували метод класифікації пухлин головного мозку шляхом аналізу зображень магнітно-резонансної та даних магнітно-резонансної спектроскопії пацієнтів з доброякісними та злоякісними пухлинами для визначення типу пухлин [4].

1.2.2. Індексація та пошук зображень.

SooBeom Park та інші запропонували метод класифікації зображень на основі вмісту, який відображає форму предмета на основі візерунка текстурної ознаки [5].

1.2.3. Багатокласне розпізнавання об’єктів

Критично важливою здатністю є можливість інтелектуального робота сприймати своє оточення. Yuhua Zheng і Yan Meng

запропонували модель, яка поєднувала ряд модульних нейронних мереж для розпізнавання декількох класів об'єктів для роботизованої системи.

Кількість класів модульних нейронних мереж залежить від кількості класів об'єктів, що розпізнаються, і кожна модульна мережа фокусується лише на вивченні одного класу об'єктів. Для кожної модульної нейронної мережі шляхи «знизу вгору» (керовані сенсорами) та «зверху вниз» (керовані очікуваннями) з'єднані разом, а керований алгоритм навчання застосовується для оновлення відповідних ваг обох шляхів. Також оцінюються дві різні навчальні стратегії: підготовка лише для позитиву та позитивно-негативна підготовка [6].

1.2.4. Використання нейронних мереж для дослідження позаземних об'єктів

Роботизований транспортний засіб для дослідження космосу та вивчення землі планети. Він має можливість подорожувати по поверхні ландшафту та інших космічних тіл. Штучні нейронні мережі мають багато переваг у космічних програмах завдяки: ефективності, адаптації, низькому енергоспоживанню.

1.3. Рекурентні нейронні мережі

Рекурентні нейронні мережі (RNN) - це тип архітектури нейронної мережі, який в основному використовується для виявлення закономірностей у послідовності даних. Такими даними можуть бути рукописний текст, геноми, текст або числові часові ряди [4, 6]. Однак вони також застосовуються до зображень, якщо вони відповідно розкладаються на блоки та обробляються як послідовність [6].

На вищому рівні мережі RNN знаходять застосування в моделюванні мови та формуванні тексту, розпізнаванні мови, формуванні описів зображень або відео.

Що відрізняє рекурентні нейронні мережі від стандартних нейронних мереж, це те, як інформація проходить через мережу. У той час як стандартні мережі передають інформацію через мережу без циклів, RNN має цикли і передає інформацію назад у себе. Це дозволяє їм розширити функціональність стандартних мереж, а також враховувати попередні входи X0: t – 1, а не лише поточний вхід Xt.

Ця різниця доступно зображена на рис. 1.2 та рис. 1.3. Тут варіант наявності декількох прихованих шарів агрегується до одного блоку прихованих шарів. Цей блок, очевидно, може бути розширений до декількох прихованих шарів.

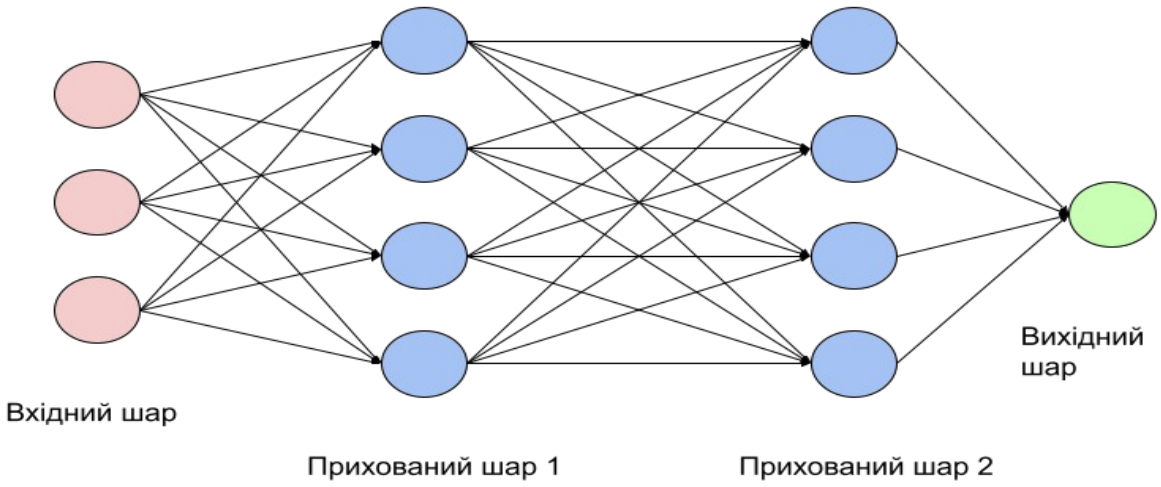


Рис. 1.2 Загальна структура нейронної мережі з двома прихованими шарами

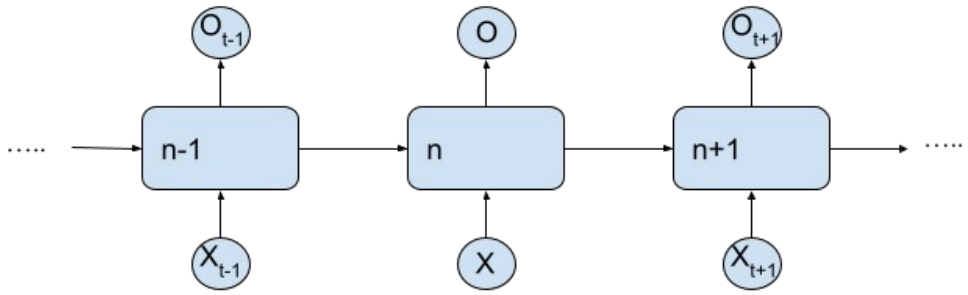


Рис. 1.3 Загальна структура рекурентної нейронної мережі

Як легко помітити, рекурентна мережа відрізняється від будь якої мережі з призованими шарами тим, що на кожній її ітерації їй як вхідні дані подається результат попередньої ітерації.

1.3.1. Проблема вибуху та зникнення градієнтів

Існують дві широко відомі проблеми, які виникають під час навчання рекурентних нейронних мереж: проблеми зникнення та вибуху, що детально описані в [7].

У першому випадку цей термін наближається до нуля експоненціально швидко, що ускладнює вивчення деяких довготривалих залежностей. Ця проблема називається зникаючим градієнтом.

У другому випадку цей термін експоненційно швидко переходить у нескінченність, і його значення стає NaN через нестабільний процес. Ця проблема називається вибуховим градієнтом.

Існує декілька простих методів, за допомогою яких можна визначити, чи не страждає ваша модель від проблеми вибухаючих чи зникаючих градієнтів.

Для проблеми зникаючого градієнту це такі показники:

1) В процесі тренування дельта зміни ваги моделі зменшуються експоненційно і приймає дуже малі значення.

2) Модель буде вдосконалюватися дуже повільно під час тренувального етапу.

3) Навчання припиняється дуже рано, тобто будь-яке подальше навчання не покращує модель.

4) Ваги ближче до вихідного шару моделі будуть змінюватись сильніше, тоді як ваги, які знаходяться ближче до вхідного шару, не будуть сильно змінюватися (якщо взагалі змінюватимуться).

Для проблеми зникаючого градієнту це такі показники:

1) Дельта зміни ваги моделі приймає значення NaN в процесі тренування.
2) В процесі тренування ваги моделі ростуть експоненційно і приймають дуже великі значення.
3) Функція витрат приймає значення NaN в процесі тренування
4) Модель матиме великі зміни у показниках функції витрат під час кожного оновлення через нестабільність.
5) Модель не вивчає багато даних при навчання, тому призводить до збільшення значень функції витрат.

Нижче перелічено 3 підходи для вирішення проблеми вибухаючих та зникаючих градієнтів:

1. Зменшення кількості шарів

   Це рішення можна використовувати в обох сценаріях (вибуховий та зникаючий градієнт). Однак, зменшуючи кількість шарів у мережі, доводиться відмовитись від більш складних моделей, оскільки модель може апроксимувати складні функції лише за наявності більшої кількості шарів.

2. Відсікання градієнтів

   Іншим рішенням є перевірка та обмеження розміру градієнтів, поки модель тренується. Відсікання градієнтів - це техніка, яка

вирішує проблему вибухових градієнтів. Ідея відсікання градієнта дуже проста: якщо градієнт стає занадто великим, потрібно масштабувати його, щоб він залишався в межах норми. Це допомагає градієнтному спуску мати адекватну поведінку, навіть якщо похідна функції витрат моделі нестабільна.

3. Ініціалізація ваги

   Більш ретельний вибір випадкової ініціалізації для вашої мережі, як правило, є частковим рішенням, однак це не вирішує проблему повністю [9].

1.4. Нейронна мережа LSTM як приклад рекурентних мереж

В останні роки, із застосуванням рекурентних нейронних мереж (RNN) при генерації тексту, якість сформованого тексту значно покращилася. На відміну від інших глибоких нейронних мереж, RNN - це спеціальна штучна нейронна мережа, яка має безліч прихованих шарів [10]. Найпростіша RNN може мати лише один прихований шар. RNN добре підходить для задач моделювання послідовностей. Вона містить з'єднання зворотного зв'язку на кожному часовому кроці, тому є можливість розширити її у часовому вимірі і сформувати глибоку нейронну мережу.

На кожному часовому кроці t RNN приймає вхідний вектор xt і вектор прихованого стану ht – 1 для створення наступного вектора прихованого стану ht за такими рівняннями:

$$\begin{cases} h_t = f\left(W \cdot x_t + U \cdot h_{t-1} + b_h\right) \\ o_t = V \cdot h_t + b_o \end{cases}$$

,

де W, U, V -- вивчені матриці ваги;

bh і bo -- прихований вектор зміщення та вектор вихідного зміщення;

f -- функція прихованого шару, яка є нелінійною функцією і зазвичай є функцією tanh або softmax.

Теоретично RNN може обробляти вхідну послідовність будь-якої довжини. Однак на практиці через проблему зникаючого градієнта вона не може ефективно справлятися з довготривалими залежностями. Однак довга короткочасна пам'ять (LSTM), запропонована Hochreiter і Schmidhuber [11], є кращою, ніж RNN, для пошуку та використання довготривалого контексту шляхом додавання вектора клітин пам'яті Ct. За одну ітерацію мережа LSTM приймає вхідні дані sxt, ht – 1, Ct – 1, а потім виробляє sht, Ct, які обчислюються як така складена функція:

$$\begin{cases} I_t = \sigma\left(W_i \cdot x_t + U_i \cdot h_{t-1} + b_i\right) \\ F_t = \sigma\left(W_f \cdot x_t + U_f \cdot h_{t-1} + b_f\right) \\ C_t = F_t \cdot C_{t-1} + I_t \cdot tanh\left(W_c \cdot x_t + U_c \cdot h_{t-1} + b_c\right) \\ O_t = \sigma\left(W_o \cdot x_t + U_o \cdot h_{t-1} + b_o\right) \\ h_t = O_t \cdot tanh(C_t) \end{cases},$$

де I_t -- вхідний порт;

F_t -- забутий порт;

O_t -- вихідний порт;

C_t - вектор активації клітин;

W_i, W_f, W_c, W_o, U_i, U_f, U_c, U_o - матриці ваги, що підлягають вивченню;

b_i, b_f, b_c, b_o - вектори упередженості, яких слід навчити;

σ - логістична сигмоїдна функція.

Ці чотири вектори і прихований вектор h_t мають однаковий розмір. При t = 1, h_0, C_0 ініціалізуються нульовим вектором.

У RNN короткочасна пам'ять h буде продовжувати примножуватися, а потім градієнт зникає. У LSTM накопичення використовується замість множення, таким чином вирішуючи проблему зникнення градієнта [12]. В даний час LSTM перевершив RNN за багатьма завданнями, включаючи моделювання мови [13].

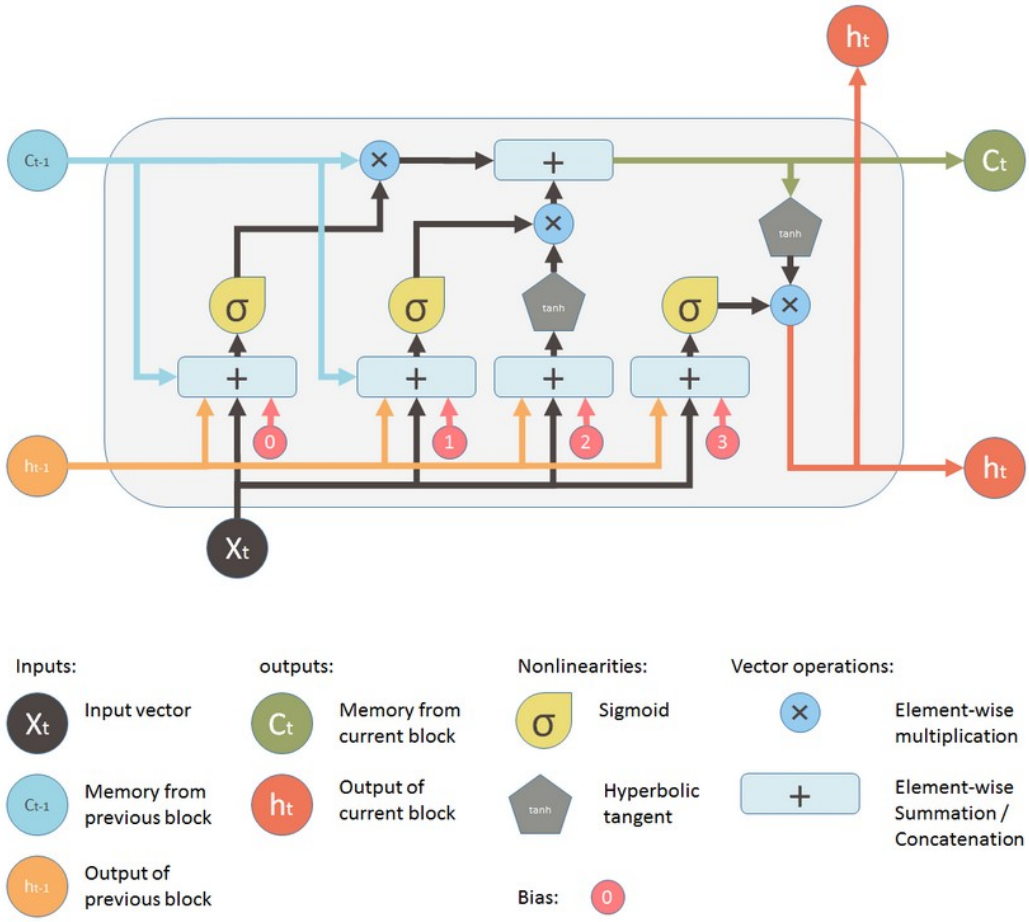


Рис. 1.4 Загальна структура LSTM [14]

Тобто є сенс використовувати LSTM замість RNN для обробки тексту.

1.4.1. Застосування LSTM в стеганографії

Ключовим моментом процесу приховування інформації є генерація тексту стего за допомогою моделі мови на рівні символів на основі LSTM [15]. Текст стего, який можливо згенерувати, є своєрідним

сигналом послідовності, в той час як RNN дуже добре підходить для моделювання послідовності.

Однак RNN не може вирішити довгострокову залежність, яка може призвести до зникнення градієнта та призвести до того, що параметри не оновлюються. Додаючи комірку пам'яті, LSTM успішно вирішує цю проблему. Тому, нарешті, було використано LSTM для побудови моделі для створення стего тексту.

Скориставшись перевагами моделі Маркова для генерації тексту, були представлені деякі нові лінгвістичні стеганографічні методи. Більше того, [16] докладав зусиль для спрощення процедури оцінки генерації тексту. Передбачається, що всі ймовірності переходу з даного стану в інший стан рівні. Однак у процесі генерації тексту вони ігнорували ймовірність переходу кожного слова, що призвело до того, що сформований текст мав низьку якість.

Подібним чином [17] запропонував стеганографічний метод, заснований на моделі Маркова, який зосереджується на тому, як забезпечити вбудовування кожного згенерованого речення з фіксованою кількістю секретних бітів, але отриманий результат не був задовільним через ігнорування різниці ймовірності переходу. [18] використовував модель ланцюга Маркова для генерування конкретного навчання ци-поезії з даного корпусу для приховування секретної інформації. Щоб подолати погіршення якості сформованого стего тексту, спричинене кодуванням фіксованої довжини, [19] поєднав модель Маркова та кодування Хаффмана, запропонувавши метод автоматичного генерування стего тексту. Під час генерації тексту кожного разу для слів-кандидатів будували дерево Хаффмана відповідно до їх умовних ймовірностей. Слово, код якого відповідає секретним бітам, було обрано для створення як стего слово. Хоча лінгвістичні стеганографічні методи,

засновані на моделі Маркова, покращили якість та надійність сформованого тексту порівняно з попередніми методами, все ще існують деякі проблеми; наприклад, створені тексти недостатньо природні, щоб уникнути їх виявлення за допомогою стеганалізу через обмеження марковської моделі.

З недавніми досягненнями нейронних мереж, моделювання мови на основі нейронних мереж почало демонструвати задовільні показники [20]. В результаті з'явилися деякі лінгвістичні стеганографічні методи, засновані на нейронних мережах. [21] вперше представив LSTM для вивчення статистичної мовної моделі природного тексту та дослідив систему стеганографії на основі LSTM, яка може створювати тексти з різними жанрами та високою якістю, навчаючи різні моделі. [22] запропонував лінгвістичну стеганографію на основі рекурентної нейронної мережі. Він використовував повне двійкове дерево та дерево Хаффмана для динамічного кодування умовного розподілу ймовірності кожного слова-кандидата, щоб секретна інформація була вбудована у вибране слово відповідно до кодів слів-кандидатів.

Тим часом деякі дослідники звертали увагу на генерацію конкретних семантичних текстів. Ло і Хуан [23] запропонували метод стеганографії для створення класичної китайської поезії за допомогою рамки кодера-декодера. Тонг та ін. [24] представив модель кодера-декодера RNN для створення текстів китайської поп-музики, щоб приховати секретну інформацію.

Сформовані специфічні тексти, вбудовані в секретну інформацію, були у певній формі, відповідаючи візуальним вимогам та вимогам вимови. Лінгвістична стеганографія на основі нейронних мереж значно покращила якість сформованих стеготекстів.

Варто зазначити, що у всіх вищезазначених методах для генерації тексту використовуються моделі мови на рівні слова. Для методів генерації тексту на рівні слова зазвичай потрібні великі словникові запаси, щоб зберігати всі слова у великомасштабному корпусі, так що введення та вихідні дані завжди надзвичайно складні, а модель вимагає мільярди параметрів.

Моделі посимвольного кроку намагалися подолати цю проблему [24]. Експериментальні результати в [25] також показали, що навіть представлення контексту може бути сформовано для охоплення характеристик морфології та семантики. Оскільки кількість символів у мові невелика, вхід і вихід мовної моделі на рівні символів є простими, а модель мови на рівні символів має перевагу в моделюванні слів, що не містять словникового запасу.

1.5. Нейронна мережа Word2Vec для обробки тексту

Word2vec - це двошарова нейронна мережа, яка обробляє текст, “векторизуючи” слова [26]. Його вхід - це текстовий корпус, а його вихід - набір векторів: вектори функцій, що представляють слова в цьому корпусі. Хоча Word2vec не є глибокою нейронною мережею, він перетворює текст у числову форму, яку глибокі нейронні мережі можуть зрозуміти.

Програми Word2vec виходять за рамки розбору речень у дикій природі. Він може бути застосований так само добре до генів, коду, лайків, списків відтворення, графіків соціальних мереж та інших словесних чи символічних серій, в яких можна розпізнати шаблони.

Оскільки слова - це просто дискретні стани, як інші дані, згадані вище, і ми просто шукаємо перехідні ймовірності між цими станами: ймовірність їх спільного виникнення. Тож gene2vec, like2vec та

follower2vec - це все можливо. Маючи це на увазі, підручник нижче допоможе вам зрозуміти, як створити нейронні вбудовування для будь-якої групи дискретних та спільних станів.

Призначення та корисність Word2vec полягає у групуванні векторів подібних слів у просторі векторів. Тобто виявляє схожість математично. Word2vec створює вектори, які розподіляють числові подання ознак слова, таких функцій, як контекст окремих слів. Це робиться без втручання людини.

Враховуючи достатню кількість даних, використання та контексти, Word2vec може робити дуже точні здогади про значення слова на основі минулих появ.

Ці здогадки можна використовувати для встановлення асоціації слова з іншими словами (наприклад, «чоловік» - це «хлопчик», що «жінка» - «дівчина»), або кластеризувати документи та класифікувати їх за темами. Ці кластери можуть лягти в основу пошуку, аналізу настроїв та рекомендацій у таких різноманітних галузях, як наукові дослідження, юридичні відкриття, електронна комерція та управління відносинами з клієнтами.

Результатом роботи нейромережі Word2vec є словниковий запас, до якого кожен елемент має приєднаний вектор, який можна подати в мережу або просто запитати, щоб виявити взаємозв'язок між словами.

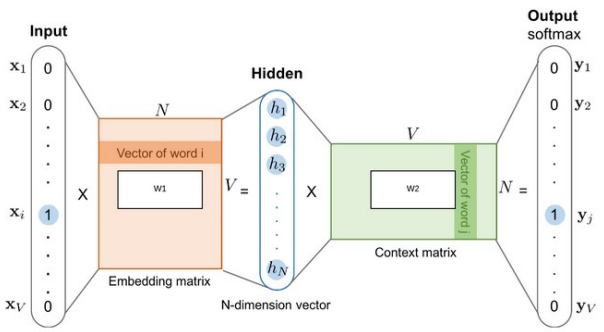


Рис. 1.5 Загальна структура Word2Vec [27]

Висновок до розділу 1

Вище було розглянуто і порівняно різні види існуючих нейронних мереж, їх переваги та недоліки. А саме: особливості генерації природного тексту з контрольованим навчанням, з навчальним підкріпленням, та з конкурентним навчанням.

Також був проведений загальний огляд нейронних мереж типу LSTM. Крім цього було розглянуто нейронну мережу для генерації тексту Word2Vec.

Розвиток RNNLM було детально описано в трьох навчальних парадигмах, а саме контрольованому навчанні, навчанні з підкріплення та змагальному навчанні.

Таким чином актуальною є задача створення системи генерації природнього тексту на основі нейронних мереж, що дозволить створювати тематично забарвлений текст та використовувати дану мінімалістичну систему для різних цілей, наприклад, для автоматизованого створення тексту для веб сайтів замість стадартного текстового набору lorem ipsum.

РОЗДІЛ 2. ОГЛЯД ІСНУЮЧИХ ПІДХОДІВ ТА ЕКСПЕРИМЕНТІВ

У цьому підрозділі буде представлений огляд класичних та нещодавно розроблених нейронних моделей генерації тексту.

Контрольовані методи навчання з метою MLE є найбільш широко прийнятим рішенням для генерації природних мов, але вони, ймовірно, спричиняють проблему упередженості щодо даних на яких проводилось тренування. Таким чином, для поліпшення таких проблем пропонуються на розгляд різні техніки, включаючи повторне масштабування шкали та ієрархічну архітектуру. Також буде розглянуто моделі на основі MLE та навчання з підкріпленням.

2.1. Огляд нейронних мереж для обробки природних мов

Проблеми обробки природної мови [28] (NLP), особливо генерації природних мов (NLG), давно вважаються одними з найскладніших обчислювальних завдань [29].

Методи генерації природних мов широко використовуються як основний модуль у різних завданнях, таких як:

- генерація речень або віршів [30]
- Контекстно зумовлена генерація тексту, така як підпис зображення[31]
- генерація емоційно забарвлених речень [32] та ін.

Проблема є складною з кількох причин. Зазвичай, в цих завданнях існує контекстна різниця між вводом і виводом [33], особливо для випадків з нетекстовим введенням. Семантика введення, як правило, конкретна і чітка, тоді як компоненти природної мови часто неоднозначні. Цей факт змушує моделі генерації природних мов (NTG)

знаходити загальні закономірності цільової мови та виражати вхідну інформацію з неоднозначністю шляхом побудови відповідних контекстів.

Під час цієї процедури в основному виникають дві труднощі. Першою із них є граматична складність природної мови. Інша -- труднощі під час отримання, спрощення та трансформації вхідної інформації.

Для вирішення граматичних проблем дослідники розробляють загальні підходи до побудови складних систем, заснованих на знаннях, як це обговорюється в [34]. Важливо зазначити, що, хоча ця парадигма потребує значних зусиль людини, вона все ще широко використовується у багатьох комерційних продуктах сьогодні, оскільки вона є зрозумілою та надійною, якщо вона добре розроблена.

Протягом останнього десятиліття нейронні мережі (NN) показали багатообіцяючі результати у багатьох сферах. З метою створення тексту мовною моделлю нейронної мережі (NNLM) [35], було вперше запропоновано використати переваги NN для генерації тексту. NNLM можна розглядати як пряме продовження n-грамової парадигми з узагальнюючими можливостями NN. Однак теоретично неможливо вирахувати n-грамову парадигму для довготривалих залежностей, згідно з деякими попередніми зауваженнями [36]. Для вирішення цієї проблеми розроблена рекурентна модель мови нейронної мережі (RNNLM) [37]. Вона є більш загальним втіленням мовної моделі з властивостями Маркова. Типова RNNLM використовує рекуррентну нейронну мережу (RNN) для кодування результату попереднього обчислення у “прихований” вектор, який потім використовується під час виведення наступного токену.

Генерування нейронного тексту посилено вивчалося з часів RNNLM. Наприклад, прийнявши вдосконалені варіанти RNN, наприклад, довгу короткочасну пам'ять (LSTM) [38] та керований рекурентний блок (GRU) [39], RNNLM показують прийнятні результати у відстеженні довготривалих залежностей в тексті. Однак, як зазначає [40], якщо розподіл згенерованого моделлю тексту співпадає з розподілом вхідних даних це не означає, що згенерований текст буде задовільним, через проблему упередженості щодо даних, на яких проводилось тренування. Пізніше пропонуються різні види рішень, включаючи моделі на основі навчання з підкріпленням, генеративні конкурентні мережі (GAN) [41; 42] та основні методи повторної параметризації [43].

У цьому підрозділі представлено систематичний огляд цих нещодавно запропонованих моделей нейронного генерування тексту (NTG). Також, розглянуто різні властивості цих моделей та відповідні методи для вирішення їх загальних проблем, такі як градієнтне зникнення під час тренувань та різноманітність згенерованих даних. Нарешті, проведено експеримент порівняльного аналізу з різними типами нейронних моделей генерування тексту на двох відомих наборах даних та розглянуто емпіричні результати разом із вищезазначеними властивостями моделей.

2.2. Навчальні парадигми RNNLM

У цьому підрозділі, в основному представлено три навчальні парадигми RNNLM, а саме контрольоване навчання, прийоми навчання з підкріпленням та схеми конкурентних тренувань.

2.2.1. Генерація природного тексту з контрольованим навчанням

Хоча генерування тексту насправді є непідконтрольним навчальним завданням, існує певна контрольована метрика, яка є задовільним наближенням до базової істини за певних обмежень. Ці алгоритми спрямовані на безпосередню оптимізацію певної контрольованої метрики. Деякі з них можуть містити корисні прийоми, які допоможуть полегшити деякі конкретні проблеми.

Першим прийомом є оцінка максимальної правдоподібності. Як правило, класичні нейронні моделі навчаються шляхом оцінки максимальної ймовірності (Контрольоване навчання) [44]. Контрольоване навчання є природним для RNNLM, оскільки він розглядає проблему генерації як послідовну класифікацію з декількома мітками, а потім безпосередньо оптимізує перехресну ентропію з кількома мітками.

Важливо зазначити, що на сьогоднішній день більшість існуючих сучасних моделей генерації природних мов приймають контрольоване навчання як свою навчальну основу [31, 32]. Контрольоване навчання має кращий коефіцієнт конвергенції та надійність навчання порівняно з іншими алгоритмами.

Однак теоретично контрольоване навчання страждає від так званого упередженого впливу [45], що обумовлено невід'ємною різницею між етапом навчання та етапом умовиводу мовних моделей, що навчаються за допомогою контрольованого навчання. Тобто, мовна модель навчена генерувати відповідні наступні лексеми з урахуванням префіксу основної істини під час навчання.

Однак на етапі висновку вільного генерування модель повинна передбачити наступний маркер з урахуванням сформованого префікса.

Немає гарантії того, що модель все одно буде працювати адекватно в тих випадках, коли префікси трохи відрізняються від тих, що містяться в тренувальних даних. Ефект зміщення експозиції стає більш очевидним і серйозним, коли послідовність стає довшою, що робить контрольоване навчання менш корисним, коли модель застосовується для довгого генерування тексту.

Наступним способом покращення ефективності є планова вибірка. Для полегшення проблеми пропонується запланований відбір зразків (SS) [46]. Він вводить випадкову величину є для узгодження примусового навчання та вільної генерації, щоб закрити різницю між етапом навчання та етапом прийняття рішень. На кожному кроці процедури навчання SS результат оцінки є визначає, чи виконує модель вільне генерування. Результати SS виглядають реалістично і демонструють помітні покращення порівняно з початковим MLE.

2.2.2. Генерація природного тексту з навчальним підкріпленням

Формування тексту за допомогою RNNLM можна розглядати як процес прийняття рішень Маркова (MDP), оптимальна місцева функція якого може бути знайдена завдяки навчанню з підкріпленням [47].

Просте рішення полягає у використанні функцій заснованих на алгоритмі навчання з підкріпленням (наприклад, REINFORCE) [48] для оптимізації деяких недиференційованих показників. Класичним вибором є оптимізація BLEU [49], метрика рівня n-грам для оцінки мовної моделі. Потім ця ідея представлена як PG-BLEU.

З точки зору навчання з підкріпленням, MLE можна розглядати як навчання, що не відповідає правилам, з епізодами, відібраними з буфера відтворення, який містить основні (істинні) дані. На кожному кроці навчального процесу винагорода Rt встановлюється рівною 1,0.

Оскільки імітаційне навчання також допомагає зменшити дисперсію у багатьох сценаріях навчання з підкріпленням, це пояснює, чому такі алгоритми, як PG-BLEU, зазвичай включають попередню підготовку за допомогою MLE [42].

2.2.3. Ефективніша генерація природного тексту з конкурентним навчанням

Успіх генеративних змагальних мереж (GAN) [41] надихнув дослідників вирішити проблему шляхом створення моделей, що змагаються між собою.

Більш прямим і природним підходом є послідовна генеративна змагальна мережа (SeqGAN) [42]. SeqGAN показує багатообіцяючі емпіричні результати [50, 51]. Метод конкуренції, який походить від GAN [41], нещодавно перетворився на нову парадигму навчання без нагляду.

Конкурентне навчання показало багатообіцяючі результати у багатьох завданнях з навчанням без нагляду, таких як взаємне усунення інформації [52] та надвисока роздільна здатність [53]. Іншим цікавим прикладом є генеративне змагальне імітаційне навчання [54], яке має на меті дати оцінку навколишнього середовища шляхом спостереження епізодів ґрунтової істини, щоб модель була навчена імітувати особливості поведінки наданих основних істин. Такий процес існує також у SeqGAN.

Незважаючи на переваги, SeqGAN все ще страждає від двох основних проблем:

1. Проблема зникнення градієнта у випадку, коли дискримінатор має значно вищу ефективність за генератор, генератору стає надзвичайно важко мати будь-які фактичні оновлення, оскільки будь-які

вихідні екземпляри генератора будуть оцінені як майже 0. Це відбувається тому що навчання припиняється раніше, перш ніж настане справжня конвергенція або рівновага Неша.

2. Проблема колапсу режиму мовлення, спричинена алгоритмом REINFORCE, що збільшує передбачувану ймовірність вибірки певних токенів, що отримують високу оцінку від дискримінатора. В результаті генератору вдається лише імітувати обмежену частину цільового розподілу, що значно зменшує різноманітність згенерованих текстів.

2.3. Про обмеження RNNLM та подальший розвиток

Незважаючи на успішний розвиток, згаданий вище, більшість сучасних методів мають деякі загальні властивості, які, в деяких випадках, обмежують їх ефективність.

Зауважимо, що хоча генеративне правило за Марковим вказує на те, що варіанти RNN, такі як LSTM, виявляються повноцінними за Тьюрінгом [61], мовні моделі все ще можна розглядати як узагальнену версію регулярної граматики. Оскільки регулярна граматика є найпростішою серед чотирьох типів граматик в ієрархії Хомського, якщо трапиться щось, що перевищує можливості регулярної граматики, вся ефективність RNNLM під час моделювання природних мов повинна покладатися на узагальнюючу здатність RNN. Узагальнювальні здібності NN не завжди є надійними, тому для таких моделей важко передбачити ефективність після навчання.

З іншого боку, деякі ефективні архітектури, такі як глибока згорткова нейронна мережа (DCNN або просто CNN) [62], не були ефективно навчені для генерування нейронного тексту, особливо для безумовного генерування тексту. Однак, як показано в роботі WGAN-GP

[63], можливо, CNN може бути застосована до цього завдання за допомогою конкурентного навчання.

Особливо очікувані прориви в цьому напрямку з кількох причин. По-перше, CNN використовують природну локальність мов і одночасно обробляють довготривалі залежності, перетворюючи глобальні залежності в локальні залежності на вищих рівнях мережі. По-друге, CNN можна обчислювати паралельно. Навчання, зближення та висновки CNN зазвичай вважаються у десять разів швидшими, ніж RNN, що особливо важливо для практичного використання.

2.3.1. Полегшення вирішення проблеми зникаючого градієнта

Існує особливість, що дві моделі створюють однаковий порядок для згенерованих речень, але розподіл ймовірностей в моделі А набагато реалістичніше ніж в моделі В. Тобто правило для градієнта в моделі В призводить до набагато швидшої та стабільнішої конвергенції моделі. Інтуїтивно це дивно, оскільки краще навчений дискримінатор повинен призвести до кращої оцінки прихованого розподілу даних, а не навпаки.

Для вирішення цієї проблеми існує в основному два основних типи методів. Перший метод полягає у використанні результатів масштабування як сигналів нагороди. Типовою роботою, створеною за цим методом є (MaliGAN) [55]. Крім того, для прискорення конвергенції MaliGAN включає базовий метод винагороди, який полягає у збереженні рухомого середнього значення та дисперсії розрахованої винагороди, а потім за їх допомогою лінійно масштабує винагороду. Цей метод не тільки допомагає полегшити проблему зникнення градієнта, але й певною мірою покращує різноманітність генерації тексту моделі.

RankGAN показує багатообіцяючі результати у покращенні ефективності конвергенції SeqGAN у багатьох випадках. Однак,

оскільки вона вимагає додаткової вибірки з вхідних даних, обчислювальні витрати на неї також вищі, ніж на інші моделі.

Ranking List		Case A (Gradient Vanishes)	Case B (No Gradient Vanishing)
	I have a pen which writes well.	1.0	0.9
	I have a pen.	0.99999	0.6
Generated Samples	I have pen a.	1e-10	0.5
	pen have a I am.	1e-15	0.4
	pen a pen pen I am.	1e-20	0.3

Рис. 2.1 Приклад проблеми зникаючого градієнта

Натхненні обома ідеями, дослідники пропонують Bootstrapped Ranking Activation (BRA) [56]. BRA не вимагає змін в архітектурі SeqGAN. BRA не вимагає великих додаткових обчислень, і його легко включити в інші моделі, що робить його конкурентоспроможним стабілізатором градієнта.

2.3.2. Посилення різноманітності поколінь

Особливий випадок проблеми колапсу режиму мовлення полягає в тому, що SeqGAN завжди прагне генерувати короткі послідовності, оскільки ці послідовності легко засвоїти, щоб отримати вищі бали.

Для підвищення спроможності моделювати довготривалу залежність пропонуються ієрархічні методи, такі як LeakGAN [56]. LeakGAN продемонстрував багатообіцяючі результати в покращенні надійності генерації довшого тексту, проте, впроваджуючи інші види колапсу режиму мовлення.

2.3.3. Повторна параметризація

Окрім методів, заснованих на навчанні з підкріпленням, є спроби застосувати спосіб повторної параметризації до RNNLM, щоб обійти проблему розрахунку градієнта над дискретними лексемами. Як типовий приклад, GAN для генерації тексту за допомогою функції Gumbel softmax [57] використовує розподіл Gumbel, щоб уникнути явної вибірки, що дає можливість проводити спільне навчання шляхом зворотного розповсюдження. Змагальна генерація природної мови [58] вводить випадкові дані шляхом декодування гауссового шуму на кожному кроці, щоб уникнути явної вибірки.

Однак, надійність вищезазначеної роботи, реалізованої шляхом повторної параметризації, значно нижча, ніж у варіантів SeqGAN. Сформований текст не схожий на природну мову і має значний колапс режиму мовлення. Така проблема виявляється на платформі оцінки ефективності, а саме Texygen [59], яка проводить оцінку стандартних моделей та справедливе порівняння між різними моделями генерації природної мови.

2.3.4. Інші методи

MaskGAN [51] - перша безумовна генеративна модель за допомогою навчання послідовності до послідовності (Seq2Seq). Базова версія MaskGAN дотримується подібних ідей із запланованою вибіркою (SS) [46], проте додає конкурентне навчання з метою усунення проблеми неконсистентності SS.

Однак, маючи інші корисні доповнення, такі як механізм уваги [Bahdanau et al., 2014], MaskGAN має потенціал вийти далеко за межі ефективності SS. Зіставлення функцій конкурентного навчання для

генерації тексту (TextGAN) [60] включає конкурентне навчання шляхом мінімізації витрат на реконструкцію оцінки конкурентних ознак сформованого тексту.

2.4. Емпіричне дослідження

Було проведене емпіричне дослідження [28] типових нейронних моделей генерації тексту. Більшість з них оцінюються на основі Texygen [59]. Це платформа порівняльного аналізу, особливо для завдань з генерації тексту з багатьма добре впровадженими базовими моделями та різними показниками оцінки. Крім моделі, яку вже інтегрував Texygen, було оцінено MaskGAN.

2.4.1. Набори даних

Набір даних COCO3 пропонується для титрування зображень. В експерименті використано лише анотації підписів до зображень, де відібрано 10 000 речень як навчальний набір і ще 10 000 як тестовий набір. Він містить 4682 різних слів, а максимальна довжина речення - 37. Речення в цьому наборі даних мають порівняно короткі та прості шаблони.

Набір даних EMNLP2017 WMT News4 містить дані у формі речень зі статей про новини. Беручи до уваги той факт, що більшість речень містять професіоналізми, збережено лише речення, що містять лише найбільш часто вживані слова. В результаті підрахунку було з’ясовано що ці дані складаються з 5700 слів.

Після попереднього процесу обирано 200 000 речень як навчальний набір, 10000 речень як тестовий набір. Максимальна довжина речення - 51, і його можна розглядати як набір даних для довгого тексту.

Таблиця 2.1 Набори даних COCO3 та EMNLP2017 WMT News4

	COCO3	EMNLP2017 WMT News4
Навчальний набір	10 000	200 000
Тестовий набір	10 000	10 000
Максимальна довжина речення	37	51
Вибірка слів	4 682	5 700

2.4.2. Метрики

BLEU [Papineni et al., 2002] та NLL-test [59] використовуються для оцінки подібності між документами або здатності генератора відповідати реальним даним. Крім того, Self-BLEU [59], який обчислює оцінку BLEU між сформованими реченнями, використовується для моніторингу тяжкості режиму колапсу.

2.4.3. Деталі навчання

В експерименті SeqGAN, RankGAN, MaliGAN, TextGAN, LeakGAN та MaskGAN вибираються як порівнювані алгоритми. Результати стандартного MLE також додаються в якості довідки.

2.4.4. Налаштування експерименту

У експериментах усі параметри моделей GAN ініціалізуються відповідно до стандартного розподілу Гауса N (0, 1). Перед змагальним тренуванням спочатку проведено попередню підготовку генератора та дискримінатора кожної моделі, використовуючи навчання MLE 80 епох відповідно, а потім проведено змагальне навчання 100 епох. У

навчальному процесі LeakGAN використано схему переплетення, де 5 епох MLE будуть проводитись через кожні 10 змагальних епох.

2.4.5. Результати експерименту

Оцінки BLEU за тестовими даними EMNLP2017 WMT News4 та Image COCO наведені у таблицях 2.2 та 2.3 відповідно. LeakGAN демонструє велику перевагу в цій метриці, особливо коли завдання - довге створення тексту. Серед інших моделей, при генерації короткого тексту, SeqGAN перевершує інші моделі, тоді як MaliGAN має невеликий недолік, але він досить добре працює при довгій генерації тексту. MaskGAN та TextGAN погано працюють за цією метрикою.

На рисунках 2.2 і 2.3 зображені криві NLL-test в процесі тренувань. Вертикальна пунктирна лінія являє собою кінець процесу попередньої підготовки. MaskGAN виключається в цій частині, оскільки не може бути оцінений NLL-test. Оскільки MLE безпосередньо оптимізує NLL-test (за даними тренувань), найкращі результати завжди досягаються в кінці попередньої підготовки, крім LeakGAN. NLL-test втрата LeakGAN має тенденцію до сходження до ще нижчого значення після досягнення мінімуму в кінці попередньої підготовки, що може бути результатом його процесу чергування. TextGAN забезпечує найнижчу продуктивність за цим показником серед усіх моделей, оскільки його навчальна мета полягає не у ймовірності, а в дистанції розподілу функцій.

Інші моделі мають майже однакові криві навчання під час попередньої підготовки, оскільки всі вони використовують стандартний MLE в цьому процесі.

Оцінки Self-BLEU наведені в таблицях 2.4 і 2.5. Можна помітити, що при генерації короткого тексту MaskGAN має найменш серйозне

згортання режиму. З іншого боку, TextGAN страждає від дуже серйозної проблеми колапсу режиму, особливо коли навчальним набором є довгий текст.

Враховуючи той факт, що TextGAN має досить високий показник BLEU, коли n-грам великий, це може бути наслідком його колапсу режиму, що означає, що він буде генерувати високочастотні фрази у великій кількості та багаторазово.

Як легко бачити з експерименту, серед інших моделей, MaliGAN має найнижчий рівень колапсу режиму, що дає йому значнуперевагу над іншими моделями, але ця перевага зникає, коли справа доходить до створення довгих текстів, адже з цією задачею дана модель справляється не так успішно.

Далі будуть наведені порівняльні таблиці з результатами усіх моделей на різних датасетах.

Таблиця 2.2 Оцінка BLEU на EMNLP2017 WMT News

Модель	BLEU2	BLEU3	BLEU4	BLEU5
SeqGAN	0.724	0.416	0.178	0.086
MaliGAN	0.755	0.436	0.168	0.077
RankGAN	0.686	0.387	0.178	0.086
LeakGAN	0.835	0.648	0.437	0.271
MaskGAN	0.265	0.165	0.094	0.057
TextGAN	0.205	0.173	0.153	0.133
MLE	0.771	0.481	0.249	0.133

Таблиця 2.3 Оцінка BLEU на Image COCO

Модель	BLEU2	BLEU3	BLEU4	BLEU5
SeqGAN	0.745	0.498	0.294	0.180
MaliGAN	0.673	0.432	0.257	0.159
RankGAN	0.743	0.467	0.264	0.156
LeakGAN	0.744	0.517	0.327	0.205
MaskGAN	0.539	0.328	0.209	0.143
TextGAN	0.593	0.463	0.277	0.207
MLE	0.731	0.497	0.305	0.189

Таблиця 2.4 Оцінка Self-BLEU на Image COCO

Модель	BLEU2	BLEU3	BLEU4	BLEU5
SeqGAN	0.950	0.840	0.670	0.489
MaliGAN	0.918	0.781	0.606	0.437
RankGAN	0.9590.618	0.882	0.762	0.618
LeakGAN	0.934	0.818	0.663	0.510
MaskGAN	0.752	0.516	0.378	0.293
TextGAN	0.942	0.931	0.804	0.746
MLE	0.916	0.769	0.583	0.408

Таблиця 2.5 Оцінка Self-BLEU на EMNLP2017 WMT News

Модель	BLEU2	BLEU3	BLEU4	BLEU5
SeqGAN	0.907	0.704	0.463	0.265
MaliGAN	0.909	0.718	0.470	0.252
RankGAN	0.897	0.677	0.448	0.298
LeakGAN	0.938	0.821	0.668	0.510
MaskGAN	0.448	0.244	0.140	0.091
TextGAN	0.999	0.975	0.967	0.962
MLE	0.851	0.572	0.316	0.171

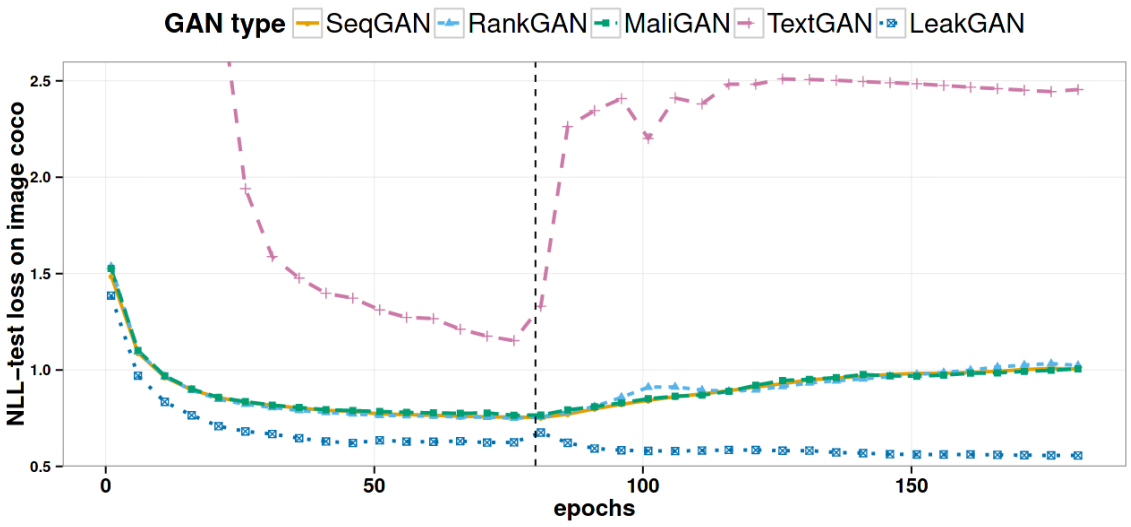


Рис. 2.2 NLL-test на датасеті зображень COCO [28]

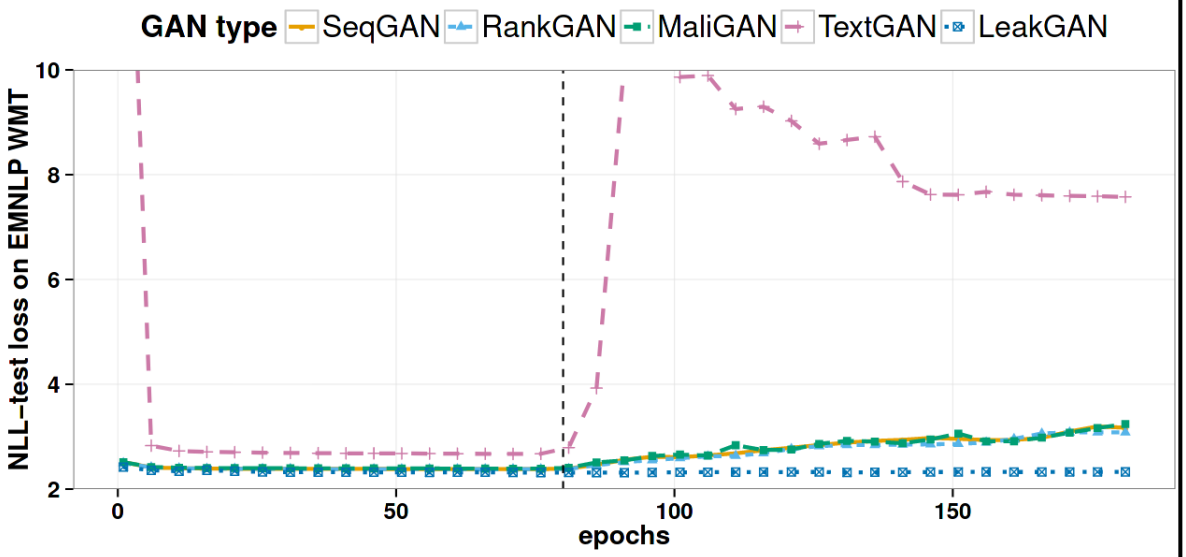


Рис. 2.3 NLL-test на датасеті EMNLP2017 WMT News [28]

На основі приведених графіків можна зробити висновки щодо сфер застосування кожної з моделей в залежності їх успішності в тій чи іншій галузі.

2.5. Вибір інструментів реалізації

Першочерговою задачею розробника нейронної мережі є вибір зручної мови програмування. В нашому випадку Python розглядається для розвитку машинного навчання, в першу чергу, через простоту на фоні решти мов програмування.

По-перше, синтаксис програмування та структури даних Python дуже прості і легко засвоюються. Відповідно, в ньому можна ефективно виконувати численні алгоритми машинного навчання.

Також Python потребує значно менше часу на освоєння на відміну від інших мов програмування, таких як Lisp, Java, C#, C, C++. Він підтримує об'єктно-орієнтований, а також процедурно-орієнтований стилі програмування.

Крім того, Python має багато бібліотек, які значно допомагають у вирішенні поставлених завдань у області машинного навчання. Наприклад: Numpy - це бібліотека для Python, яка надає можливість швидко виконувати велику кількість числових розрахунків.

Іншим важливим інструментом для розробки є бібліотеки для обраної мови програмування. Поставлені вимоги добре задовольняє бібліотека Matplotlib, яка застосовується для генерації графіків і діаграм. Matplotlib напряму може працювати з типами даних з NumPy.

Ця бібліотека надає об'єктно-орієнтований API для вбудовування графіків у документ з використанням загальних інструментів графічного інтерфейсу, таких як Tkinter, wxPython, Qt або GTK.

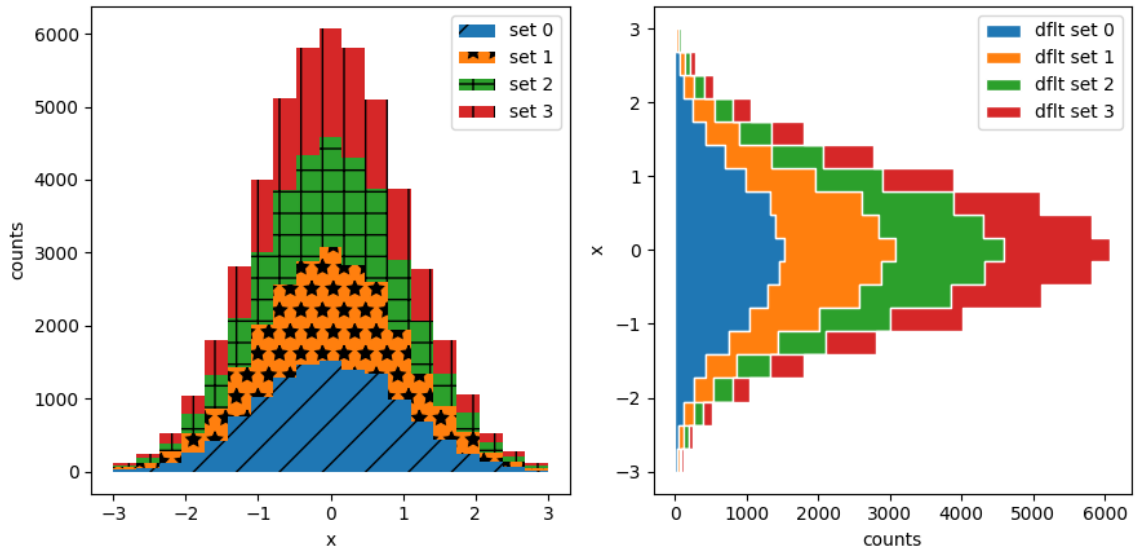


Рис. 2.4 Приклад графіка, згенерованого Matplotlib [64]

Не менш важливим фактором є середовище розробки. Jupyter -- це веб-програма з відкритим кодом, за допомогою якої можна створювати та обмінюватися документами, що містять виконуваний код, рівняння, візуалізації та текст. Назва Jupyter походить від назв мов програмування, які вона підтримує: Julia, Python та R. Jupyter Notebook підтримується людьми з проєкту Jupyter.

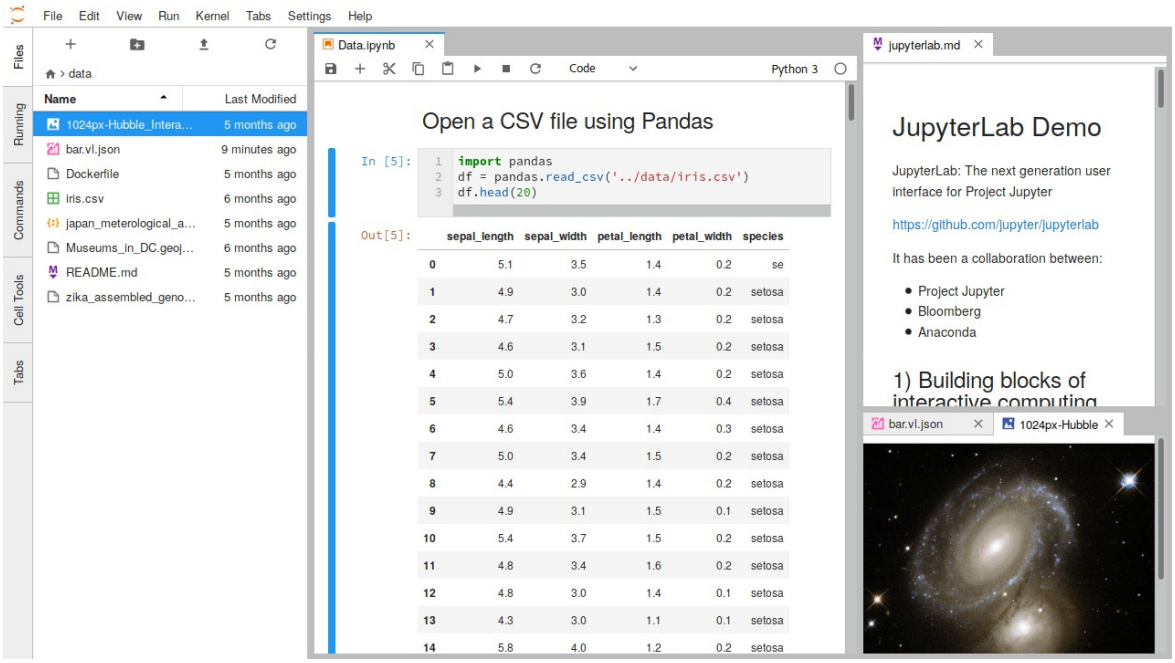


Рис. 2.5 Приклад інтерфейсу Jupiter [65]

Jupyter -- це виділений проєкт із проєкту IPython, який раніше мав сам проєкт IPython Notebook. Jupyter постачається з ядром IPython, що дозволяє писати програми на Python, але на даний момент існує понад 100 інших ядер, які ви також можна використовувати.

Висновок до розділу 2

У цьому розділі представлений огляд класичних та нещодавно запропонованих нейронних моделей генерації тексту. Контрольовані методи навчання з цілями MLE є найбільш широко прийнятим рішенням для генерації природних мов, але вони, ймовірно, спричиняють проблеми зі зміщенням експозиції.

Таким чином, для полегшення проблем розглядаються різні методи, включаючи масштабування винагороди та ієрархічні архітектури. У цьому підрозділі представлено систематичний огляд цих нещодавно запропонованих моделей нейронного генерування тексту (NTG).

Також, розглянуто різні властивості моделей та відповідні методи для вирішення їх загальних проблем, такі як градієнтне зникнення під час тренувань та різноманітність згенерованих даних.

Нарешті, проведено експеримент порівняльного аналізу з різними типами нейронних моделей генерування тексту на двох відомих наборах даних та розглянуто емпіричні результати разом із вищезазначеними властивостями моделей.

РОЗДІЛ 3. РОЗРОБКА СИСТЕМИ

Метою цього розділу є детальне пояснення процесу розробки системи. У цьому розділі буде проведено детальний аналіз обраного датасету, а також надано статистичну інформацію щодо цього датасету.

Крім того, в розділі 3 буде детально описано розроблену систему та створені програмні компоненти, що до неї відносяться. Створена система налічуватиме 5 класів, структуру та методи кожного з яких буде пояснено нижче.

3.1. Датасет для тренування нейронної мережі

Набір даних є необхідною складовою процесу тренування нейронної мережі. Потрібно обрати базу однорідного відформатованого тексту. Крім цього потрібно використовувати датасет на який не накладено ніяких ліцензійних обмежень щодо його використання у напрямку дослідження нейронних мереж. Також бажано використовувати відносно розповсюджений набір даних, щоб інші розробники також могли мати до нього доступ у випадку необхідності та для дослідницьких цілей.

У нашому випадку для тренування нейронної моделі був обраний датасет що складається з П'ятикнижжя Мойсеєвого — першої частина Танаху, або, як його називають християни, Старого Заповіту Біблії. П’ятикнижжя - це перша частина Біблії, що складається з книг Буття, Виходу, Левиту, Числа і Повторення Закону. Цей датасет задовільняє всім наведеним вище критеріям — ці дані не мають накладеної обмежуючої ліцензії, їх легко обробляти, бо їх можна знайти в великій кількості форматів даних, та вони безкоштовні.

Нижче наведено аналіз складу датасета.

Таблиця 3.1 Кількість символів і слів у кожній з книжок

Книжка	Кількість слів	Кількість символів
Буття	36299	187684
Виходу	32683	170381
Левиту	24791	128699
Числа	33349	178711
Повторення Закону	28351	147094

У процесі розробки було підраховано загальну кількість слів у використаному датасеті. Було отримано наступні результати: загальна кількість слів -- 155473, а загальна кількість символів -- 812569.

3.2. Опис програмних елементів

3.2.1. Клас Utils

Клас Utils містить у собі допоміжну функцію sample.

Її аргументами є preds, temperature. Ця функція використовується для того, щоб вибрати індекс зразка з масиву ймовірностей.

```
class Utils:
    @staticmethod
    def sample(preds, temperature=1.0):
        """Helper function to sample an index from a probability array"""
        preds = np.asarray(preds).astype("float64")
        preds = np.log(preds) / temperature
        exp_preds = np.exp(preds)
        preds = exp_preds / np.sum(exp_preds)
        probas = np.random.multinomial(1, preds, 1)
        return np.argmax(probas)
```

Рис. 3.1 Функція sample

Таким чином ми можемо бачити структуру класу Utils.

3.2.2. Клас Dataset

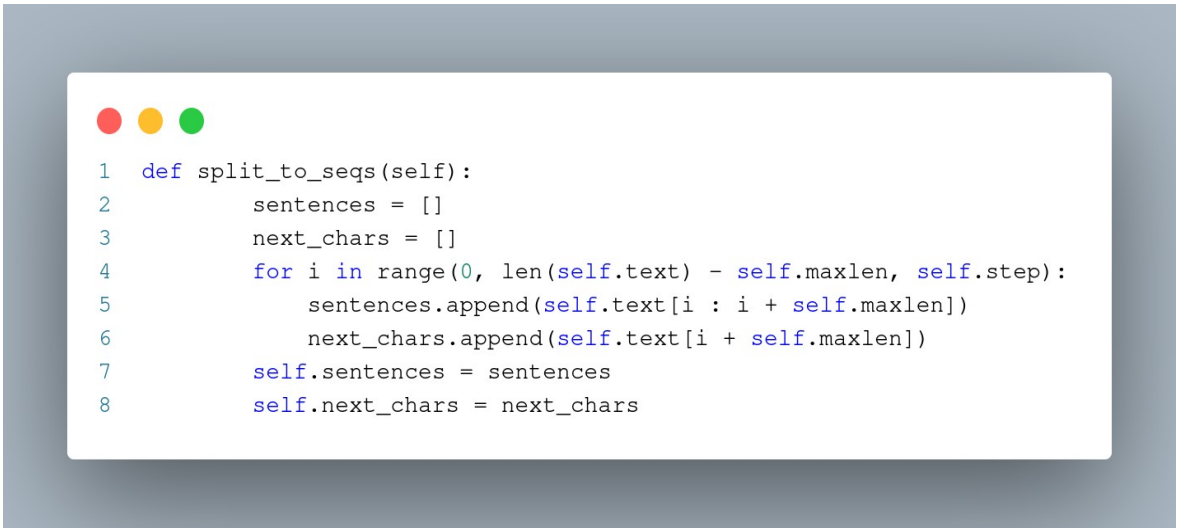
Цей клас включає в себе всю інформацію стосовно даних на яких тренується модель. В першу чергу в ньому виконується зчитування даних з диска. В цьому класі проходять всі необхідні маніпуляції щодо перетворення даних в потрібний формат. Цей клас інкапсулює в собі перетворення даних з текстового представлення даних в числове та навпаки, тому що мережі обробляють тільки числове введення даних.

Функція __init__ . ЇЇ аргументи: self, text_path, maxlen, step. Функція реалізує зчитування з файлу. Вона трансформує букви в числа і навпаки для полегшення обробки нейронними мережами. Як результат роботи даної функції ми маємо датасет тексту для нейронної мережі.

```
def __init__(
        self, text_path="bible/combined_bible.txt", maxlen=40, step=3
    ):
        with open(text_path) as file:
            text = file.read().lower()

        # We remove newlines chars for nicer display
        text = text.replace("\n", " ")

        chars = sorted(list(set(text)))
        char_indices = dict((c, i) for i, c in enumerate(chars))
        indices_char = dict((i, c) for i, c in enumerate(chars))

        self.text = text
        self.chars = chars
        self.char_indices = char_indices
        self.indices_char = indices_char
        self.maxlen = maxlen
        self.step = step

        self.split_to_seqs()
```

Рис. 3.2 Функція __init__

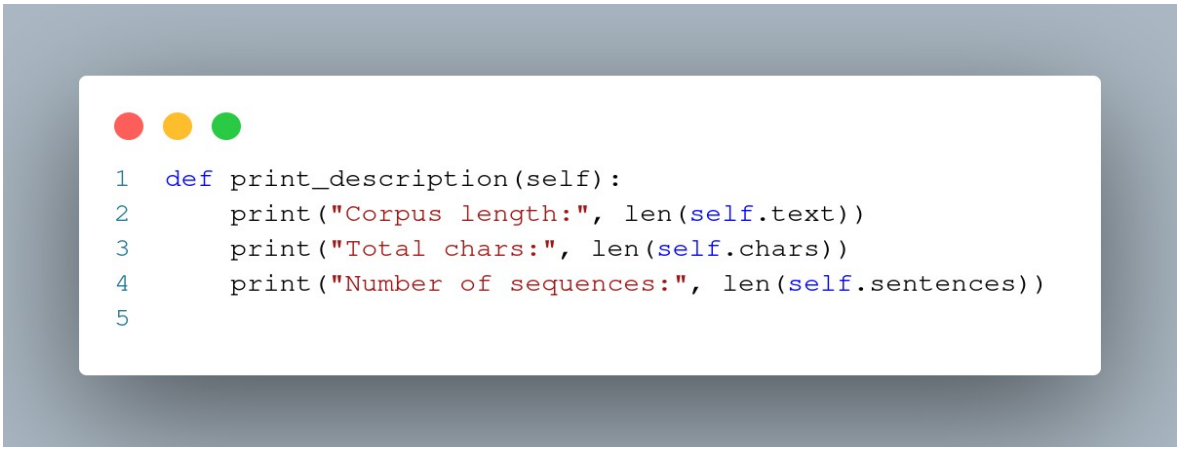
Функція split_to_seqs. У цієї функції аргументи відсутні. Функція split_to_seqs була застосована для розділення тексту на послідовності однакової довжини. Зведення послідовностей до однакової довжини необхідно для нормального сприйняття їх нейронною мережею.



```
def split_to_seqs(self):
        sentences = []
        next_chars = []
        for i in range(0, len(self.text) - self.maxlen, self.step):
            sentences.append(self.text[i : i + self.maxlen])
            next_chars.append(self.text[i + self.maxlen])
        self.sentences = sentences
        self.next_chars = next_chars
```

Рис. 3.3 Функція split_to_seqs

Функція print_description. У цієї функції аргументи відсутні. Функція print_description використовується з ціллю отримати інформацію про датасет в потік стандартного виводу програми. Разом з тим, ця функція надає дані про кількість символів і послідовностей у заданому наборі даних. Також виводиться довжина тексту. Ці дані необхідні для тренування нейронних мереж, адже потрібно мати набори даних фіксованої розмірності.



```
def print_description(self):
    print("Corpus length:", len(self.text))
    print("Total chars:", len(self.chars))
    print("Number of sequences:", len(self.sentences))

```

Рис. 3.4 Функція print_description

Функція get_x_y. У цієї функції аргументи відсутні. Функція get_x_y створює дані x та y на основі набору даних для розрахунку успішності тренування моделі. Де x - вхідні дані для нейронної мережі, а y - очікувані відповіді нейронної мережі.

```
def get_x_y(self):
        x = np.zeros(
            (len(self.sentences), self.maxlen, len(self.chars)), dtype=np.bool
        )
        y = np.zeros((len(self.sentences), len(self.chars)), dtype=np.bool)
        for i, sentence in enumerate(self.sentences):
            for t, char in enumerate(sentence):
                x[i, t, self.char_indices[char]] = 1
                y[i, self.char_indices[self.next_chars[i]]] = 1
        return x, y
```

Рис. 3.5 Функція get_x_y

З цього опису можна зрозуміти структуру класу Dataset.

3.2.3. Клас Model

Клас Model містить в собі функції __init__ , print_description, setup_optimizer, train, generate.

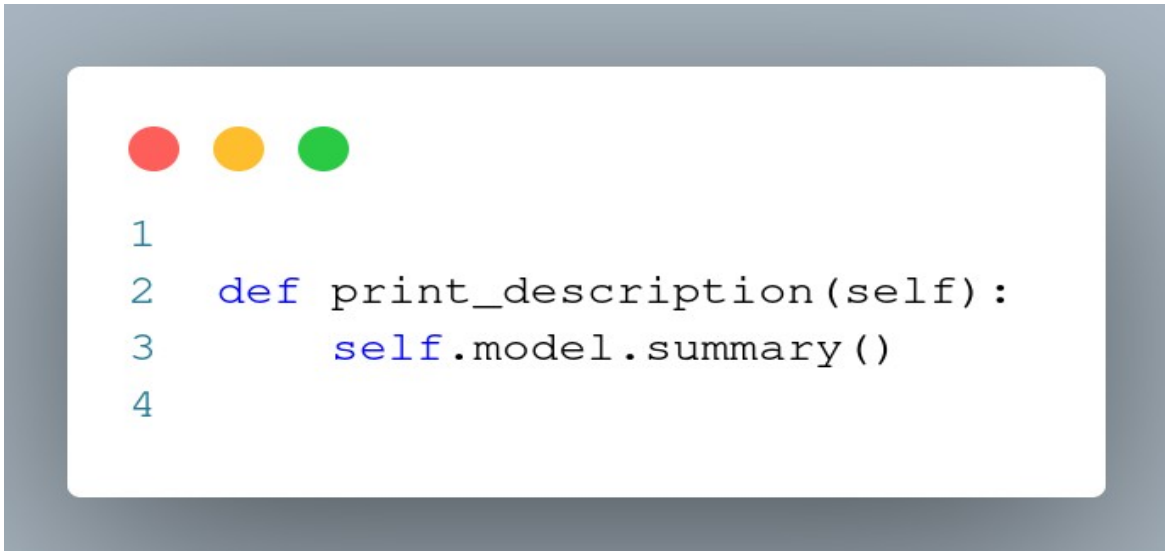
Функція __init__ . Її аргументи: self, dataset, load_folder. В залежності від стані аргументу load_folder визначаються подальші дії програми. А саме: коли стан аргументу функції load_folder дорівнює None, то функція створює нову нейронну модель, порівняно з тим, у випадку, коли цей аргумент функції load_folder не дорівнює None, функція завантажує вже існуючу нейронну модель із папки, шлях до якої передається саме у цьому аргументі load_folder. За замовчуванням використовується значення None. Зазвичай у програмі при необхідності завантаження нейронної мережі з диску, передається шлях checkpoint,

що вказує на підпапку у поточній директорії користувача, де розташована збережена модель.

```
def __init__(self, dataset, load_folder=None):
        if load_folder is None:
            chars_len = len(dataset.chars)
            shape = (dataset.maxlen, chars_len)
            self.model = keras.Sequential(
                [
                    keras.Input(shape=shape),
                    layers.LSTM(128, return_sequences=True),
                    layers.LSTM(128, return_sequences=True),
                    layers.LSTM(64),
                    layers.Dense(chars_len, activation="softmax"),
                ]
            )
        else:
            self.model = keras.models.load_model(load_folder)
```

Рис. 3.6 Функція __init__

Функція print_description. У цієї функції аргументи відсутні. Ця функція використовується для виводу даних про модель. Дані про модель включають: кількість шарів моделі, їх розмірності, входи-виходи моделі, а також її структуру.



```

def print_description(self):
    self.model.summary()

```

Рис. 3.7 Функція print_description

Функція setup_optimizer. Її аргументи: self, metrics, optimizer, loss. Функція setup_optimizer виконує задачу компіляції моделі. Разом з тим, вона задає функцію оптимізації Adam. Також, у цій функції була застосована категоріальна крос-ентропія як функція помилки.

```
1
2  def setup_optimizer(
3      self, metrics, optimizer="adam", loss="categorical_crossentropy"
4  ):
5      self.model.compile(
6          loss=loss,
7          optimizer=optimizer,
8          metrics=metrics,
9      )
10
```

Рис. 3.8 Функція setup_optimizer

Далі описаний принцип роботи функції train. Її аргументи: self, dataset, callbacks, batch_size, epochs. Функція запускає та контролює тренування моделі. Одна епоха дорівнює одній ітеерації нейронної мережі над повним датасетом. Функція тренування встановлює кількість епох рівну 40 для найбільш ефективного тренування нейронної мережі. Вона використовує переданий датасет в якості тренувального.

Під час тренування моделі, після кожної епохи відбувається перевірка ефективності моделі. У випадку, коли вона не підвищується протягом декількох ітерацій, функція ініціює дострокове завершення процесу тренування нейронної мережі.

Окрім того, у випадку зростання ефективності моделі під час моніторингу, поточний варіант нейронної моделі зберігається на диск, де вона зберігається у папці checkpoint, що розташована у поточній директорії користувача.

```
1
2  def train(self, dataset, callbacks, batch_size=32, epochs=40):
3      x, y = dataset.get_x_y()
4      self.model.fit(
5          x,
6          y,
7          batch_size=batch_size,
8          epochs=epochs,
9          callbacks=callbacks,
10     )
11
```

Рис. 3.9 Функція train

Наступна функція generate. ЇЇ аргументи: self, dataset, diversity. Основна задача функції -- створення тексту з натренованою моделлю. Таким чином, ми можемо мати доступ до результатів навчання нашої нейронної мережі.

```
1  def generate(
2          self,
3          dataset,
4          diversity=1.0,
5      ):
6          start_index = random.randint(0, len(dataset.text) - dataset.maxlen - 1)
7          generated = ""
8          seed = dataset.text[start_index : start_index + dataset.maxlen]
9          for i in range(400):
10             x_pred = np.zeros((1, dataset.maxlen, len(dataset.chars)))
11             for t, char in enumerate(seed):
12                 x_pred[0, t, dataset.char_indices[char]] = 1.0
13             preds = self.model.predict(x_pred, verbose=0)[0]
14             next_index = Utils.sample(preds, diversity)
15             next_char = dataset.indices_char[next_index]
16             seed = seed[1:] + next_char
17             generated += next_char
18         return seed, generated
```

Рис. 3.10 Функція generate

В загальному, ця функція використовується безпосередньо для генерації тексту.

3.2.4. Клас ArgumentParser

Цей клас містить у собі функцію parse_command_line_argument.

Функція parse_command_line_argument. У цієї функції аргументи відсутні.

```
def parse_command_line_arguments():
        parser = argparse.ArgumentParser()
        parser.add_argument(
            "--train", help="Train the model.", action="store_true"
        )
        parser.add_argument(
            "--load",
            help="Load a pretrained model and generate a bit of text.",
        )
        args = parser.parse_args()
        if bool(args.train) == bool(args.load):
            parser.error("You must specify either --train or --load <path>")
        return args
```

Рис. 3.11 Функція parse_command_line_argument

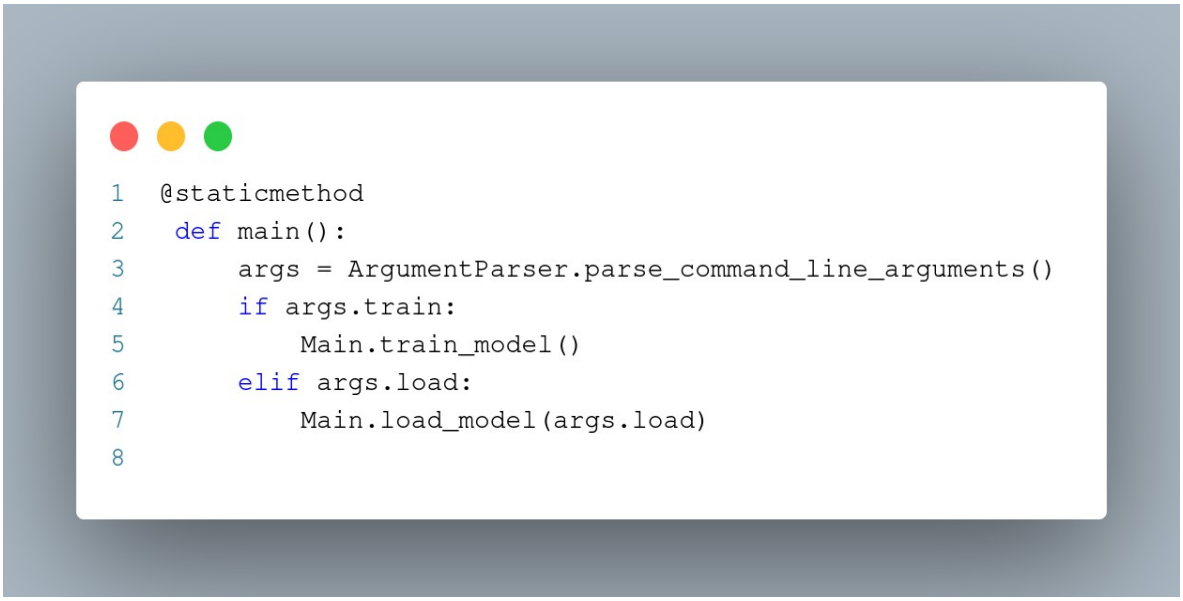
Функція parse_command_line_argument зчитує аргументи, що користувач передає у програму через командний рядок. Після того, вона аналізує їх і повертає запит.

3.2.5. Клас Main

Містить в собі функції main, load_model, train_model. Необхідно розглянути клас Main. У даному класі проходять всі процеси навчання нейронної мережі з початку до кінця.

Функція main. ЇЇ аргументи: self, dataset, diversity. Основною задачею функції є запуск програми тренування нейронної мережі. Особливо важливо, що у цій функції визначається чи буде нейронна

мережа завантажена з диску, чи створювати нову і тренувати її спочатку. Тобто ця функції визначає, який з описаних сценаріїв потрібно запустити, щоб вдовільнити запити користувача.



```
@staticmethod
 def main():
     args = ArgumentParser.parse_command_line_arguments()
     if args.train:
         Main.train_model()
     elif args.load:
         Main.load_model(args.load)

```

Рис. 3.12 Функція main

Наступна функція load_model. ЇЇ аргумент model_path. Ця функція завантажує нейронну модель з диску на я кому вона зберігається. Разом із тим, функція load_model генерує текст за допомогою щойно завантаженої моделі.

```

@staticmethod
def load_model(model_path):
    dataset = Dataset()
    model = Model(dataset=dataset, load_folder=model_path)
    seed, generated = model.generate(dataset=dataset, diversity=1.0)
    print("Seed:", seed)
    print("Generated:", generated)

```

Рис. 3.13 Функція load_model

Функція train_model. У цієї функції аргументи відсутні. Функція train_model активується коли у функції main було вирішено створити нову нейронну мережу. У такому випадку відбувається завантаження нового набору даних. Після цього функція створює нову нейронну модель. Далі щойно завантажена модель проходить процес тренування.

```python
def train_model():
        dataset = Dataset()
        dataset.print_description()

        model = Model(dataset=dataset)
        model.print_description()
        model.setup_optimizer(
            loss="categorical_crossentropy",
            optimizer="adam",
            metrics=["accuracy"],
        )

        epochs = 40
        batch_size = 128

        callback = keras.callbacks.EarlyStopping(
            monitor="accuracy", patience=5
        )
        checkpoint = keras.callbacks.ModelCheckpoint(
            "checkpoint", monitor="accuracy", save_best_only=True, mode="max"
        )
        model.train(
            dataset=dataset,
            callbacks=[callback, checkpoint],
            batch_size=batch_size,
            epochs=epochs,
        )
        seed, generated = model.generate(
            dataset=dataset,
            diversity=1.0,
        )
        print("Seed:", seed)
        print("Generated:", generated)
```

Рис. 3.14 Функція train_model

В результаті виникає можливість генерувати текст за допомогою нової нейронної моделі.

Висновок до розділу 3

В даному розділі було наведено детальне пояснення процесу розробки системи. Більш того, у розділі 3 було проведено детальний аналіз обраного датасету, що складається з п’ятикнижжя Мойсеєвого, а також надано статистичну інформацію щодо цього датасету.

Нарешті, в розділі 3 було детально описано розроблену систему та створені програмні компоненти, що до неї відносяться. Створена система налічує 5 класів, структуру та методи кожного з яких було детально описано та пояснено у розділі.

РОЗДІЛ 4. РЕЗУЛЬТАТИ РОЗРОБКИ ТА ТЕСТУВАННЯ МОДЕЛІ

Мета цього розділу -- детальний розгляд процесів навчання моделі, а також деталей її тестування. Крім того, необхідно надати детальний опис використаних у тренуванні моделі алгоритмів, таких як алгоритм оптимізації Adam, а також функції нелінійного активування softmax і функії визначення помилки нейронної мережі категоріальної крос-ентропії.

Більше того, у цьому розділі буде описано обрану структуру нейронної мережі та процесів її навчання і тестування.

Також буде розглянуто залежності програми і їх версії.

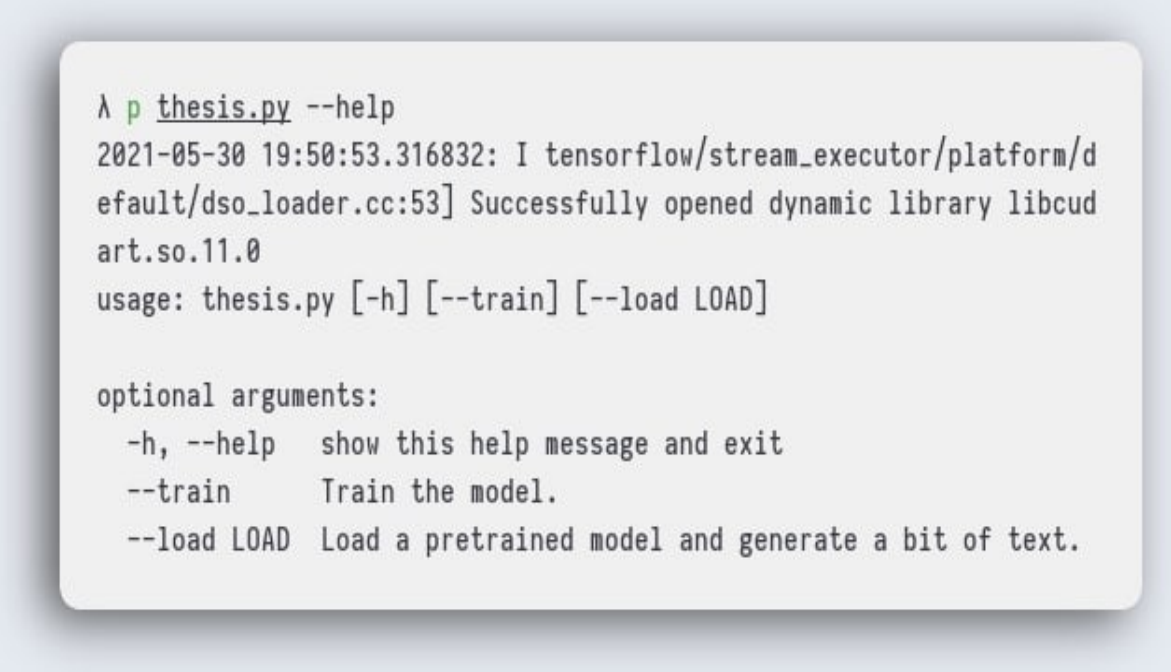


Рис. 4.1 Вивід допоміжного повідомлення

Також створена програма підтримує керування за допомогою командного рядка.

4.1. Опис обраних алгоритмів, мови і бібліотек

4.1.1. Обрана мова та бібліотеки

Далі наведено конкретні версії залежностей програмного забезпечення які були використані при розробці нейронної мережі.

1. Версія 3.9.5. мови програмування Python.
2. Бібліотека Numpy, версія 1.19.5.
3. Бібліотека Tensorflow+gpu версія 2.5.0.
4. Бібліотека Keras версія 2.4.3.

Далі буде розглянуто обрані алгоритми та принципи їх роботи.

4.1.2. Алгоритм оптимізації Adam

Adam — це алгоритм оптимізації, який можна використовувати замість класичної стохастичної процедури градієнтного спуску для оновлення вагових коефіцієнтів мережі ітеративно на основі навчальних даних. Швидкість навчання підтримується для кожної ваги мережі (параметра) і окремо адаптується в міру розгортання навчання. Adam, метод ефективної стохастичної оптимізації, який вимагає лише градієнтів першого порядку з невеликими вимогами до пам'яті.

Метод обчислює індивідуальні показники адаптивного навчання для різних параметрів на основі оцінок першого та другого моментів градієнтів; ім'я Adam походить від адаптивного оцінювання моменту. Метод розроблений для поєднання переваг двох останніх популярних методів: AdaGrad, який добре працює з розрідженими градієнтами, та RMSProp, який добре працює в Інтернеті та з нестаціонарним налаштуванням.

Деякі переваги Adam полягають у тому, що величини оновлень параметрів незмінні до масштабування градієнта, його кроки приблизно обмежені гіперпараметром кроку, він не вимагає стаціонарної мети, він

працює з розрідженими градієнтами, і це природно виконує кроковий розмір відпалу.

Зокрема, алгоритм обчислює експоненціальну рухому середню градієнта та квадратичний градієнт, а параметри beta1 та beta2 контролюють швидкість занепаду цих рухомих середніх. Початкове значення рухомих середніх та значень beta1 та beta2, близьких до 1,0 (рекомендовані), призводять до зміщення оцінок моменту до нуля. Це упередження долається спочатку обчисленням упереджених оцінок, а потім обчисленням, скоригованим із зміщенням.

4.1.3. Функція перехресної ентропії

Функція перехресної ентропії - це функція оптимізації, яка використовується для навчання моделей класифікації машинного навчання, які класифікують дані, передбачаючи ймовірність (значення від 0 до 1) того, чи належать дані до одного класу чи іншого класу.

Якщо передбачувана ймовірність класу значно відрізняється від фактичної мітки класу (0 або 1), величина перехресних ентропійних втрат велика. У випадку, якщо передбачувана ймовірність класу наближається до мітки класу (0 або 1), втрата перехресної ентропії буде меншою. Перехресні ентропійні втрати зазвичай використовуються як функція втрат для моделей, які мають вихід softmax.

Зокрема, функція перехресної ентропії або втрата журналу використовується як функція витрат для моделей логістичної регресії або моделей із вихідним коефіцієнтом softmax (мультиноміальна логістична регресія або нейронна мережа) для оцінки параметрів моделі логістичної регресії.

4.1.4. Функція softmax

Функція softmax - це функція, яка перетворює вектор K дійсних значень на вектор K дійсних значень, що складають до 1. Вхідні значення можуть бути додатними, від'ємними, нульовими або більшими, ніж одиниця, але softmax перетворює їх у значення між 0 і 1, щоб їх можна було інтерпретувати як ймовірності. Якщо один із входів занадто малий або від'ємний, softmax перетворює його на дуже малу ймовірність, а якщо вхід занадто великий, тоді він перетворює його на велику ймовірність, але він завжди залишатиметься між 0 і 1.

Функцію softmax іноді називають функцією softargmax або багатокласовою логістичною регресією. Це пояснюється тим, що softmax - це узагальнення логістичної регресії, яке може бути використано для багатокласової класифікації, і його формула дуже схожа на сигмоїдну функцію, яка використовується для логістичної регресії.

Функцію softmax можна використовувати в класифікаторі лише тоді, коли класи взаємовиключні. З цієї причини, як правило, додають функцію softmax як кінцевий шар нейронної мережі. Нижче наведена формула функції softmax:

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}, \quad (4.1)$$

де $\vec{z}$ -- вхідний вектор;

z_i -- елементи вхідного вектору;

K -- кількість класів.

З інформації про деталі алгоритму роботи функції активації softmax, яку було наведено вище, можна легко побачити що ця функція

активації чудово задовільняє потреби розробників як функція, яку зручно використовувати в останньому шарі нейронної мережі. Це пов'язано з тим фактом, що для нейронної мережі, які вирішують завдання категоризації зручно використовувати функції вивід яких в сумі дає 1.

Багато багатошарових нейронних мереж закінчуються передостаннім шаром, який виводить реальні оцінки, які не зручно масштабувати і з якими може бути важко працювати.

Тут алгоритм активації softmax є дуже корисним, оскільки він перетворює вхідні значення у нормалізований розподіл ймовірностей, який може відображатися користувачеві або використовуватися як вхід для інших систем.

Таким чином цей алгоритм добре підходить для задач з генеруванням тематичного тексту, бо нейронні мережі такого типу виконують завдання класифікації під час вибору наступного символу.

Таблиця 4.1 Детальний опис складових рівняння (4.1)

$\vec{z}$	Вхідний вектор для функції softmax, що складається з (z0, ... zK).
z_i	Усі значення zi є елементами вхідного вектора функції softmax, і вони можуть приймати будь-яке дійсне значення, додатне, нульове або від'ємне. Наприклад, нейронна мережа могла б вивести такий вектор, як (-0,62, 8,12, 2,53), що не є дійсним розподілом ймовірностей, ось чому необхідний softmax.

Таблиця 4.1 (закінчення)

$\vec{z}$	Вхідний вектор для функції softmax, що складається з (z0, ... zK).
e^{z_i}	Стандартна експоненціальна функція застосовується до кожного елемента вхідного вектора. Це дає позитивне значення вище 0, яке буде дуже малим, якщо вхідне значення було від'ємним, і дуже великим, якщо вхідне значення було великим. Однак він все ще не фіксований у діапазоні (0, 1), що вимагається від ймовірності.
$\sum_{j=1}^{K} e^{z_j}$	Термін внизу формули є терміном нормування. Це гарантує, що всі вихідні значення функції складатимуть до 1 і кожне знаходитиметься в діапазоні (0, 1), таким чином, складаючи дійсний розподіл ймовірностей.
K	Кількість класів у багатокласному класифікаторі.

4.2. Структура нейронної мережі

Необхідно зазначити що наша система в загальному використовує 276467 параметрів для тренування, через те що вона містить достатню кількість LSTM шарів які потребують великих витрат на обчислення.

Структура нейронної мережі заснована на передбаченні наступного символу. Наш датасет має 51 унікальний символ. Нам необхідно визначити довжину згенерованого тексту.

У процесі розробки було прийняте рішення встановити максимальну довжину тексту в 40 символів. З цієї причини розроблена нейронна мережа має вхідну розмірність 40x51.

Таблиця 4.2 Структура нейронної мережі

Шар	Вихідна форма	Кількість параметрів
LSTM	40x128	92160
LSTM	40x128	131584
LSTM	64	49408
Повнозв’язний	51	3315

Як видно з таблиці, наша нейронна мережа має чотири шари. Основною структурною одиницею приведеної нейронної мережі було обрано модифікований підтип рекурентних нейронних мереж LSTM. Для останнього шару нейронної мережі було обрано повнозв’язний шар що використовує softmax як активаційний алгоритм.

4.3. Процес навчання та тренування моделі

4.3.1. Навчання моделі

Для повноцінної реалізації процесу тренування нейронної мережі необхідне обов'язкове виконання наступних умов:

- На комп'ютері користувача встановлена мова Python версії 3.9.
- Встановлені залежності Numpy, версія 1.19.5.; Tensorflow+gpu версія 2.5.0.; Бібліотека Keras версія 2.4.3.
- У підпапці bible містяться книги для тренування з розширенням .txt

Далі необхідно створити датасет який буде використовуватися для тренування нейронної мережі. Для цього потрібно запустити файл create_dataset.py. Ця підпрограма об'єднає всі необхідні дані в один файл. Таким чином буде створено файл combined_bible.txt у підпапці bible.

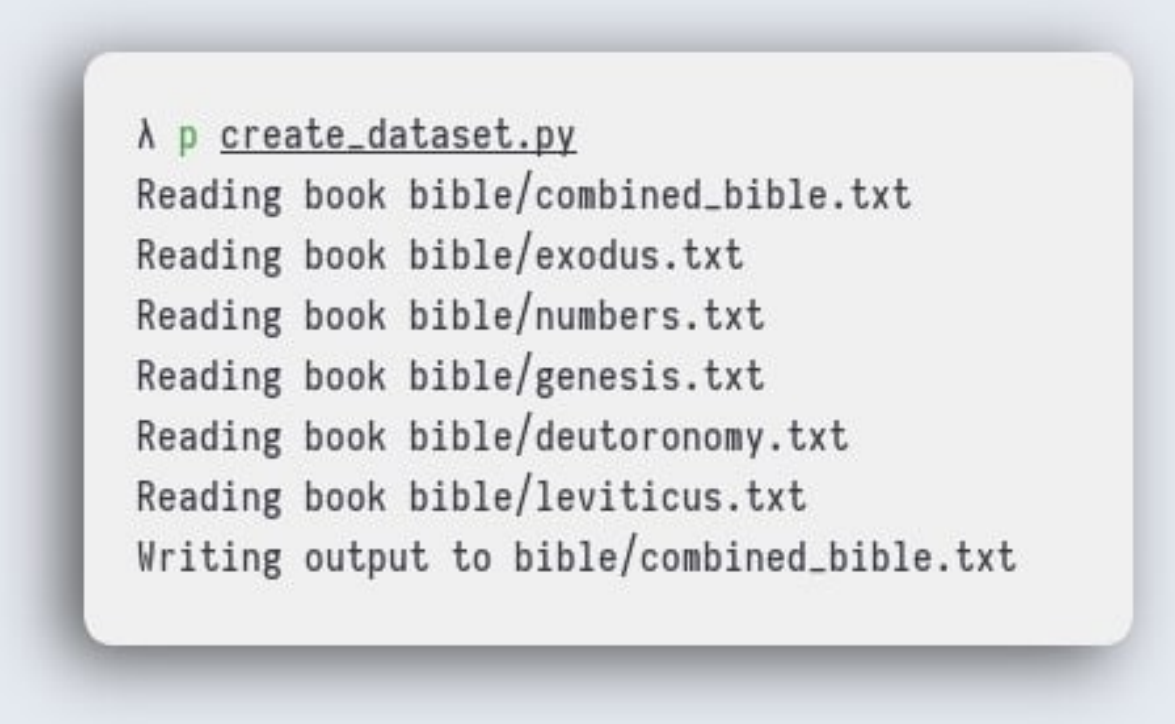


Рис. 4.2 Створення файлу combined_bible.txt

На наступному етапі починаємо безпосередньо навчання моделі. Для цього необхідно виконати файл thesis.py, передавши до нього аргумент командного рядку --train. Це ініціює надання відповідної інформації до програми, що необхідно почати тренування нової моделі. Функція main класу Main викличе функцію Main.train_model.

Ця функція створить об'єкт класу Dataset, який зчитає необхідні дані у своєму конструкторі та обробить ці дані для тренування нейронної мережі.

Після цього буде виведена статистична інформація про зчитаний датасет.

Наступним кроком буде створено об'єкт класу Model. Даний клас ініціює нову нейронну мережу, де розмірність вхідного шару задається заданим датасетом.

Основна функція train_model виведе інформацію про створену модель та налаштує функції помилки, оптимізації, та встановить необхідні метрики. Після цього будуть обрані параметри для тренування та створені функції, що моніторять ефективність навчання моделі після кожної епохи.

Далі відбувається суто тренування моделі.

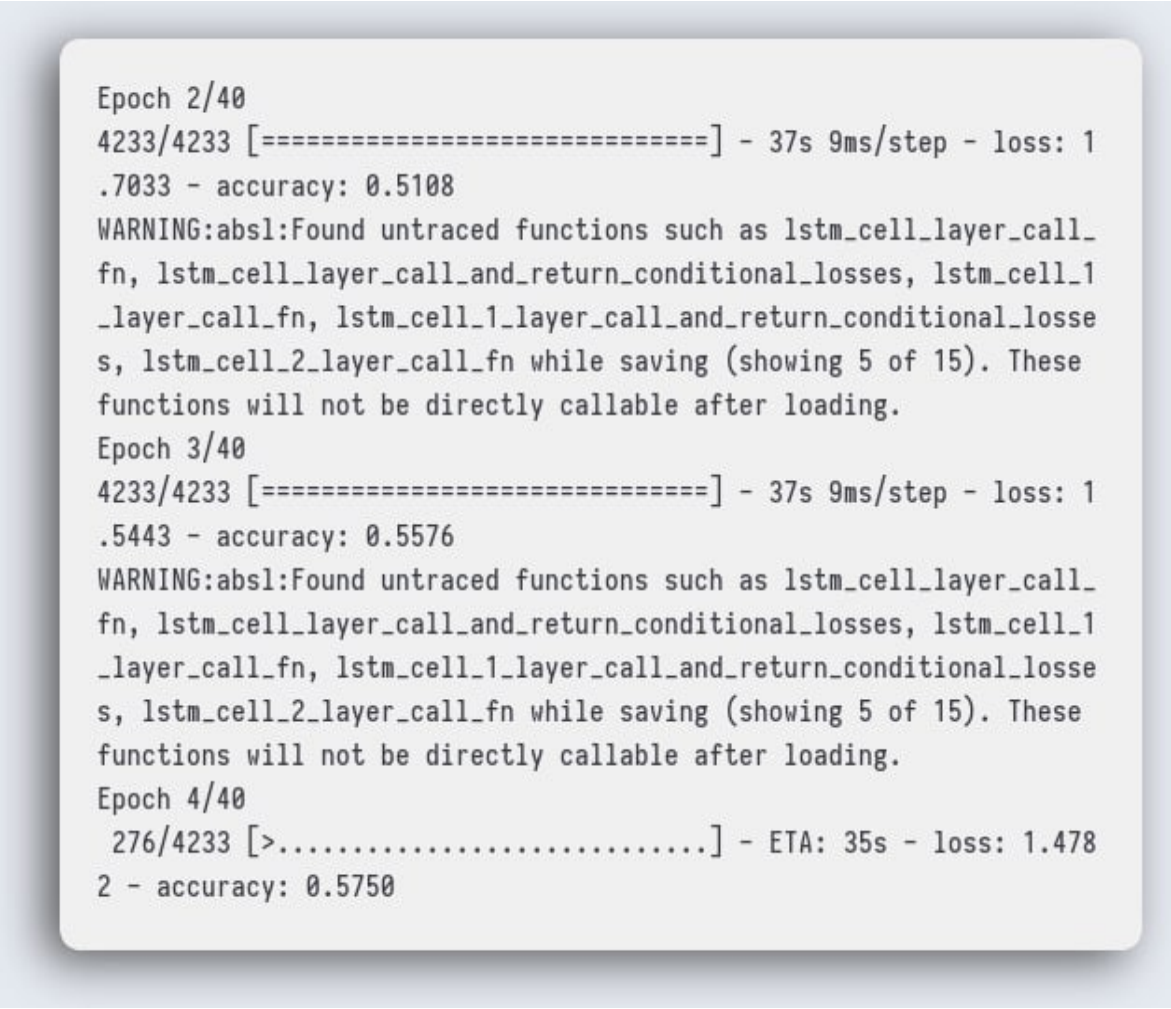


Рис 4.3 Процес тренування моделі

В середньому можливо провести обрахунок часу який займає тренування моделі. В даному випадку це 40 епох по 37 секунд кожна, отже маємо приблизно 25 хвилин.

Після завершення процесу тренування модель з найкращим показником ефективності буде записана на диск.

4.3.2. Тестування моделі

Перед початком процесу тренування нейронної моделі програма виставляє метрику ефективності, що визначається функцією категоріальної крос-ентропії. Ця метрика надає можливість слідкувати за ефективністю моделі під час її тренування. Таким чином ми отримуємо зріз ефективності моделі після кожної епохи тренування.

Таблиця 4.2 Оцінка ефективності моделі під час тренування

Епоха	Помилка	Точність
1	2.3264	0.3502
2	1.8766	0.4694
3	1.7218	0.5085
...	...	...
38	0.9793	0.7051
39	0.9708	0.7070
40	0.9693	0.7091

Таким чином легко помітити, що модель досягає точності у 71% після завершення 40 епохи. Однак, з ростом кількості епох підвищення точності сповільнюється.

Пререквізитом для тестування моделі є створена та натренована модель, яка була записана у підпапку checkpoint. Якщо ця умова виконана, користувач може виконати файл thesis.py з переданим аргументом командного рядка --load checkpoint.

Це ініціює надання відповідної інформації до програми, що необхідно зчитати натреновану модель з диску, що розташована у підпапці checkpoint.

Далі, функція main класу Main викличе функцію Main.load_model, та передасть до неї шлях до підпапки зі збереженою моделлю.

Після цього буде викликано підпрограму, яка створить новий об’єкт класу Dataset, який завантажить тренувальний датасет з підпапки bible, а після цього буде створено новий об’єкт класу Model, яка завантажить збережену нейронну мережу з заданої підпапки.

Наступним кроком буде вибір випадкової ключ-послідовності з завантаженого датасету, на базі якої модель згенерує тематичне продовження.

Потім модель створить 30 випадкових послідовностей тексту.

4.3.3. Результати екперименту

Нижче наведено декілька прикладів згенерованого моделлю тексту.

1. Ключ-послідовність: covered thee, and the ram be shildren a

   Згенерований моделлю текст: ates. thus creise his wames on own his peace and mine heshbon and out of her about. and they said, "shall speak and spake unto see in whom the hand of the children wherewulled and her, and the flesh.. what he that all the ground: in the

hooted, all his seir to be seeth my covenants came now figh this way, shephere unto them, behold, and cherounce it. were covered thee, and the ram be shildren a

2. Ключ-послідовність: rvan ephraise here the spoil unto me one

   Згенерований моделлю текст: as a sabblespe thou hasah toward the people of thy brother, and of the house of through you their fathers, and the remeate it upon the old uncleanness vancle. but lamb without you adly to you have set in any: if you give thysembe in the lord were before the lord. that ye will see with the children of israel out of the congregation. and the feap would be a servan ephraise here the spoil unto me one

3. Ключ-послідовність: from our hundred. so thy bove to the blo

   Згенерований моделлю текст: the words very generation, ye will blow; so he came went years; that ye suach, and put it was in the rams, and jachie, the god shalt put as unclean, whether ye melce as i commanded thousand and of thou shalt speak unto hood, who had serve one sister, and went out of all my hangim wood, and have not let no son by the congregation. anked the sou, and they day from our hundred. so thy bove to the blo

Як можна бачити, модель генерує осмислений текст, що нагадує текст із зразка. Модель має мінімалістичну структуру і завдяки цьому займає вагу всього 5.6 Мб. Така особливість є значною перевагою в контексті вбудовування в готовий проект з обмеженою кількістю ресурсів.

Висновок до розділу 4

У цьому розділі було детально проаналізовано процес тренування та тестування моделі. Таким чином, було показано як отримана точність моделі та проміжні результати під час тренувань. Крім цього були перераховані використані мови і бібліотеки. Також було визначено, що збільшення кількості епох тренування призводить до значного уповільнення росту ефективності нейронної мережі.

У результаті було досягнуто точності моделі, що дорівнює 71%. Незважаючи на середній показник точності моделі, слід звернути увагу, що мета створення даної нейронної мережі полягає не в тому, щоби передбачувати вже написаний текст з початкового датасету, а в тому, щоб генерувати новий тематично забарвлений текст на основі цього набору даних. У такому контексті досягнутий показник точності в 71% становить собою гарний компроміс між новизною згенерованого тексту та його змістовністю.

ВИСНОВОК

У першому розділі було проведено детальний аналіз предметної області, оглянуто принципи роботи нейронних мереж та підходів до генерації тематичних текстів.

У другому розділі було оглянуто вже існуючі системи генерації текстів на основі нейронних мереж та описано існуючі дослідження в цій темі, а також надано й обґрунтовано вибір засобів реалізації проєкту.

У третьому розділі було наведено інформацію щодо деталей розробленої системи та оглянуто її структуру, а також детально оглянуто обраний датасет для тренування нейронної мережі.

У четвертому розділі було приведено опис обраних алгоритмів та отриманиї результатів від експериментів зі створеною системою, а також надано інформацію щодо використаних мов та бібліотек.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] Kozyrev, S. V. "Classification by ensembles of neural networks." *P-Adic Numbers, Ultrametric Analysis, and Applications* 4.1 (2012): 27-33

[2] Swain, Madhusmita, et al. "An approach for iris plant classification using neural network." *International Journal on Soft Computing* 3.1 (2012): 79.

[3] Sibi, P., S. Allwyn Jones, and P. Siddarth. "Analysis of different activation functions using back propagation neural networks." *Journal of theoretical and applied information technology* 47.3 (2013): 1264-1268.

[4] Zakaria, Magdi, M. Al-Shebany, and Shahenda Sarhan. "Artificial neural network: a brief overview." *International Journal of Engineering Research and Applications* 4.2 (2014): 7-12.

[5] Park, Soo Beom, Jae Won Lee, and Sang Kyoon Kim. "Content-based image classification using a neural network." *Pattern Recognition Letters* 25.3 (2004): 287-300.

[6] Zheng, Yuhua, and Yan Meng. "Modular neural networks for multi-class object recognition." *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011.

[7] Bengio, Yoshua, Patrice Simard, and Paolo Frasconi. "Learning long-term dependencies with gradient descent is difficult." *IEEE transactions on neural networks* 5.2 (1994): 157-166.

[8] Toward Data Science [Електронний ресурс] Режим доступу — https://towardsdatascience.com/the-vanishing-exploding-gradient-problem-in-deep-neural-networks-191358470c11 (дата звернення 23.05.2021)

[9] Zakaria, Magdi, M. Al-Shebany, and Shahenda Sarhan. "Artificial neural network: a brief overview." *International Journal of Engineering Research and Applications* 4.2 (2014): 7-12.

[10] Xiang, Lingyun, et al. "Novel linguistic steganography based on character-level text generation." *Mathematics* 8.9 (2020): 1558.

[11] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." *Neural computation* 9.8 (1997): 1735-1780.

[12] Wang, Baowei, et al. "Air quality forecasting based on gated recurrent long short term memory model in Internet of Things." *IEEE Access* 7 (2019): 69524-69534.

[13] Sundermeyer, Martin, Ralf Schlüter, and Hermann Ney. "LSTM neural networks for language modeling." *Thirteenth annual conference of the international speech communication association*. 2012.

[14] Medium [Електронний ресурс] Режим доступу — https://blog.mlreview.com/understanding-lstm-and-its-diagrams-37e2f46f1714 (дата звернення 14.05.2021)

[15] Xiang, Lingyun, et al. "Novel linguistic steganography based on character-level text generation." *Mathematics* 8.9 (2020): 1558.

[16] Dai, Weihui, Yue Yu, and Bin Deng. "BinText steganography based on Markov state transferring probability." *Proceedings of the 2nd international conference on interaction sciences: information technology, culture and human*. 2009.

[17] Xiang, Lingyun, et al. "Novel linguistic steganography based on character-level text generation." *Mathematics* 8.9 (2020): 1558.

[18] Luo, Yubo, et al. "Text steganography based on ci-poetry generation using Markov chain model." *KSII Transactions on Internet and Information Systems (TIIS)* 10.9 (2016): 4568-4584.

[19] Yang, Zhongliang, et al. "Automatically generate steganographic text based on Markov model and Huffman coding." *arXiv preprint arXiv:1811.04720* (2018).

[20] Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks." *arXiv preprint arXiv:1409.3215* (2014).

[21] Fang, Tina, Martin Jaggi, and Katerina Argyraki. "Generating steganographic text with lstms." *arXiv preprint arXiv:1705.10742* (2017).

[22] Yang, Zhong-Liang, et al. "RNN-stega: Linguistic steganography based on recurrent neural networks." *IEEE Transactions on Information Forensics and Security* 14.5 (2018): 1280-1295.

[23] Luo, Yubo, and Yongfeng Huang. "Text steganography with high embedding rate: Using recurrent neural networks to generate chinese classic poetry." *Proceedings of the 5th ACM workshop on information hiding and multimedia security*. 2017.

[24] Sutskever, Ilya, James Martens, and Geoffrey E. Hinton. "Generating text with recurrent neural networks." *ICML*. 2011.

[25] Tong, Yongju, et al. "Text steganography on RNN-Generated lyrics." *Mathematical Biosciences and Engineering* 16.5 (2019): 5451-5463.

[26] Word2Vec [Електронний ресурс] — режим доступу до https://wiki.pathmind.com/word2vec (дата звернення 31.04.2021).

[27] Church, Kenneth Ward. "Word2Vec." Natural Language Engineering 23.1 (2017): 155-162.

[28] Lu, Sidi, et al. "Neural text generation: Past, present and beyond." *arXiv preprint arXiv:1803.07133* (2018).

[29] Murty, Katta G., and Santosh N. Kabadi. *Some NP-complete problems in quadratic and nonlinear programming*. 1985.

[30] Zhang, Xingxing, and Mirella Lapata. "Chinese poetry generation with recurrent neural networks." *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2014.

[31] Karpathy, Andrej, and Li Fei-Fei. "Deep visual-semantic alignments for generating image descriptions." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015.

[32] Hu, Zhiting, et al. "Toward controlled generation of text." *International Conference on Machine Learning*. PMLR, 2017.

[33] Shapiro, Stuart C. *Encyclopedia of artificial intelligence second edition*. John, 1992.

[34] Siddharthan, Advaith. "Ehud Reiter and Robert Dale. Building Natural Language Generation Systems. Cambridge University Press, 2000. $64.95/£ 37.50 (Hardback). 234 pages." *Natural Language Engineering* 7.3 (2001): 271.

[35] Bengio, Yoshua, Réjean Ducharme, and Pascal Vincent. "A neural probabilistic language model." *Advances in Neural Information Processing Systems*. 2001.

[36] Rosenfeld, Ronald. "Two decades of statistical language modeling: Where do we go from here?." *Proceedings of the IEEE* 88.8 (2000): 1270-1278.

[37] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan ˇCernock`y, and Sanjeev Khudanpur. Recurrent neural network based language model. In 11th INTERSPEECH, 2010.

[38] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Neural computation, 9(8):1735–1780, 1997.

[39] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv:1406.1078, 2014.

[40] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for se-uence prediction with recurrent neural networks. In NIPS, pages 1171–1179, 2015.

[41] Goodfellow, Ian J., et al. "Generative adversarial networks." *arXiv preprint arXiv:1406.2661* (2014).

[42] Yu, Lantao, et al. "Seqgan: Sequence generative adversarial nets with policy gradient." *Proceedings of the AAAI conference on artificial intelligence*. Vol. 31. No. 1. 2017.

[43] Kusner, Matt J., and José Miguel Hernández-Lobato. "Gans for sequences of discrete elements with the gumbel-softmax distribution." *arXiv preprint arXiv:1611.04051* (2016).

[44] Williams, Ronald J., and David Zipser. "A learning algorithm for continually running fully recurrent neural networks." *Neural computation* 1.2 (1989): 270-280.

[45] Huszár, Ferenc. "How (not) to train your generative model: Scheduled sampling, likelihood, adversary?." *arXiv preprint arXiv:1511.05101* (2015).

[46] Bengio, Samy, et al. "Scheduled sampling for sequence prediction with recurrent neural networks." *arXiv preprint arXiv:1506.03099* (2015).

[47] Sutton, Richard S., and Andrew G. Barto. "Reinforcement learning: An introduction." (2011).

[48] Sutton, Richard S., et al. "Policy gradient methods for reinforcement learning with function approximation." *NIPs*. Vol. 99. 1999.

[49] Papineni, Kishore, et al. "Bleu: a method for automatic evaluation of machine translation." *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. 2002.

[50] Guo, Jiaxian, et al. "Long text generation via adversarial training with leaked information." *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. No. 1. 2018.

[51] Fedus, William, Ian Goodfellow, and Andrew M. Dai. "Maskgan: better text generation via filling in the_." *arXiv preprint arXiv:1801.07736* (2018).

[52] Liu, Pengfei, Xipeng Qiu, and Xuanjing Huang. "Adversarial multi-task learning for text classification." *arXiv preprint arXiv:1704.05742* (2017).

[53] Ledig, Christian, et al. "Photo-realistic single image super-resolution using a generative adversarial network." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017.

[54] Ho, Jonathan, and Stefano Ermon. "Generative adversarial imitation learning." *arXiv preprint arXiv:1606.03476* (2016).

[55] Che, Tong, et al. "Maximum-likelihood augmented discrete generative adversarial networks." *arXiv preprint arXiv:1702.07983* (2017).

[56] Guo, Chuan, et al. "On calibration of modern neural networks." *International Conference on Machine Learning*. PMLR, 2017.

[57] Kusner, Matt J., and José Miguel Hernández-Lobato. "Gans for sequences of discrete elements with the gumbel-softmax distribution." *arXiv preprint arXiv:1611.04051* (2016).

[58] Rajeswar, Sai, et al. "Adversarial generation of natural language." *arXiv preprint arXiv:1705.10929* (2017).

[59] Zhu, Yaoming, et al. "Texygen: A benchmarking platform for text generation models." *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. 2018.

[60] Zhang, Yizhe, et al. "Adversarial feature matching for text generation." *International Conference on Machine Learning*. PMLR, 2017.

[61] Siegelmann, Hava T., and Eduardo D. Sontag. "On the computational power of neural nets." *Journal of computer and system sciences* 50.1 (1995): 132-150.

[62] Bottou, Leon, et al. "High quality document image compression with." *Journal of Electronic Imaging* 7.3 (1998): 410-425.

[63] Gulrajani, Ishaan, et al. "Improved training of wasserstein gans." *arXiv preprint arXiv:1704.00028* (2017).

[64] MatPlotLib [Електронний ресурс] Режим доступу — https://matplotlib.org/stable/gallery/lines_bars_and_markers/filled_step.html#sphx-glr-gallery-lines-bars-and-markers-filled-step-py (дата звернення 20.04.2021)

[65] Jupiter [Електронний ресурс] Режим доступу — https://jupyter.org/ (дата звернення 11.05.2021)

Додаток А

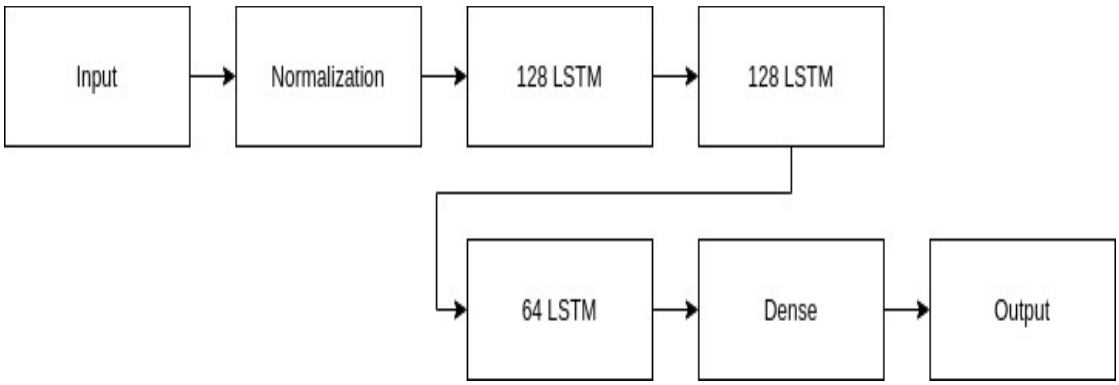
СТРУКТУРНА СХЕМА РОБОТИ

до дипломного проєкту

на тему: «Система генерації тематичних текстів на основі нейронних мереж»

Київ – 2021

СТРУКТУРНА СХЕМА РОБОТИ



					ІАЛЦ.467200.004 Д1			
Зм.	Арк.	№ докум	Підп.	Дата				
Розробив		Корнійчук О.П.			Система генерації тематичних текстів на основі нейронних мереж	Літ.	Аркуш	Аркушів
Перевірив		Волокита А.М.				Т	1	1
					Структурна схема роботи	НТУУ «КПІ ім. Ігоря Сікорського» ФІОТ Група ІВ-71		
Н.контр.		Симоненко В.П.						
Затв.		Волокита А.М.						

Додаток Б

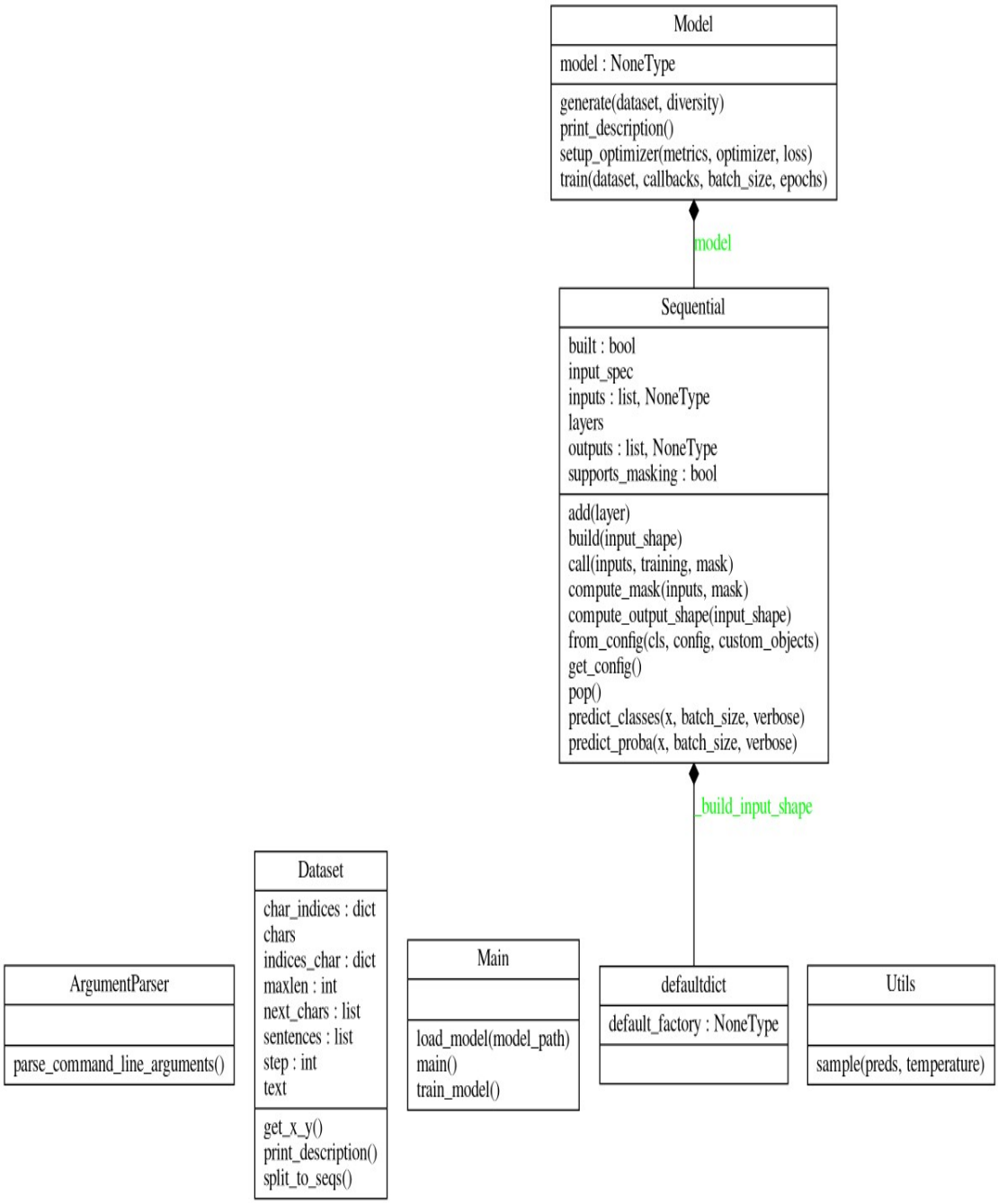
ФУНКЦІОНАЛЬНА СХЕМА ПОТОКУ ДАНИХ

до дипломного проєкту

на тему: «Система генерації тематичних текстів на основі нейронних мереж»

Київ – 2021

ФУНКЦІОНАЛЬНА СХЕМА КЛАСІВ



					ІАЛЦ.467200.005 Д2		
Зм.	*Арк.*	*№ докум*	*Підп.*	*Дата*			
Розробив		*Корнійчук О.П.*			*Система генерації тематичних текстів на основі нейронних мереж*	*Літ.*	*Аркуш*	*Аркушів*
Перевірив		*Волокита А.М.*				T	*1*	*1*
					Функціональна схема потоку даних	*НТУУ «КПІ ім. Ігоря Сікорського» ФІОТ Група ІВ-71*		
Н.контр.		*Симоненко В.П.*						
Затв.		*Волокита А.М.*						

Додаток В

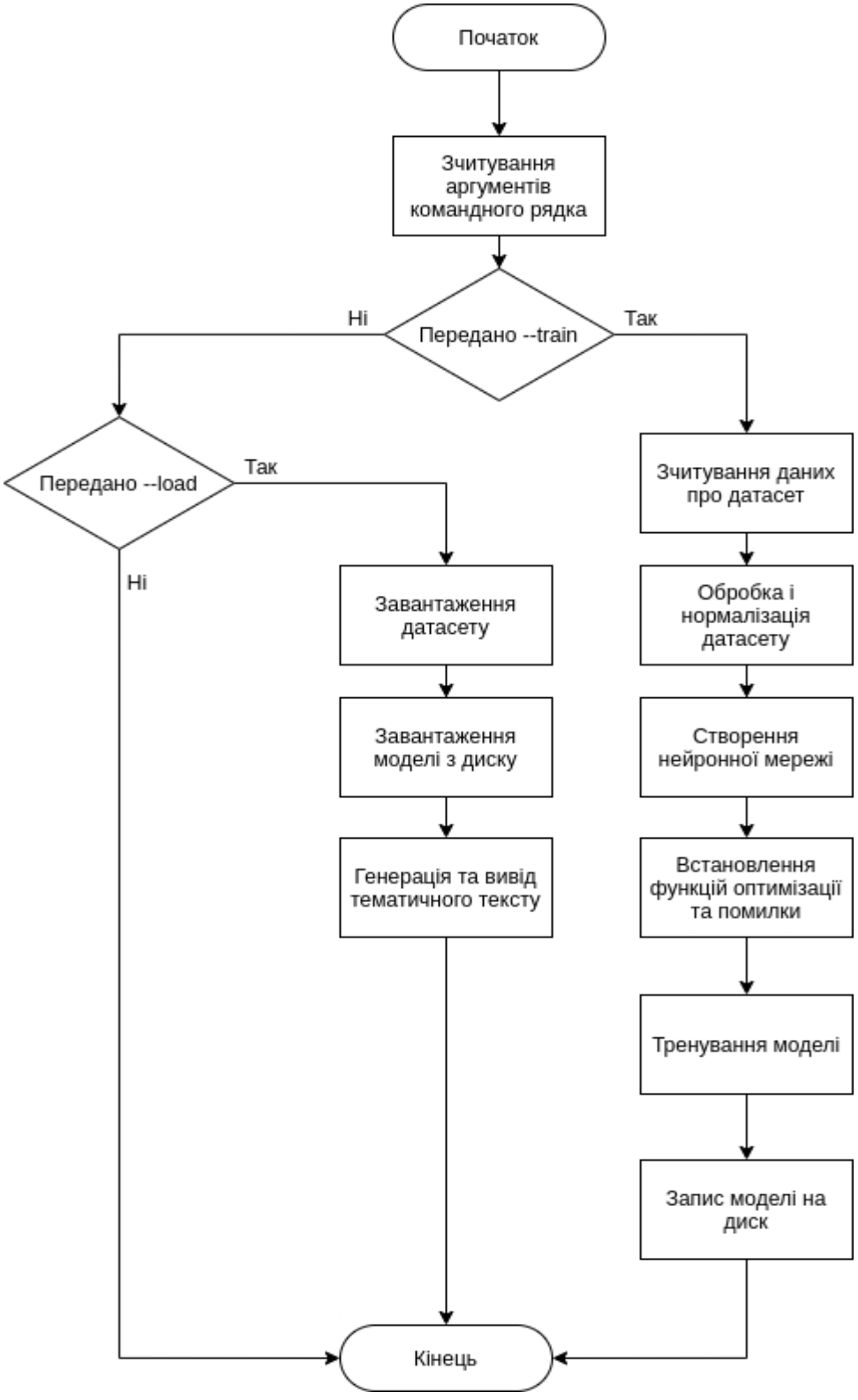
ПРИНЦИПОВА СХЕМА РОБОТИ

до дипломного проєкту

на тему: «Система генерації тематичних текстів на основі нейронних мереж»

Київ – 2021

ПРИНЦИПОВА СХЕМА РОБОТИ



					ІАЛЦ.467200.006 ДЗ	
Зм.	*Арк.*	*№ докум*	*Підп.*	*Дата*		
Розробив		*Корнійчук О.П.*			*Система генерації тематичних текстів на основі нейронних мереж*	*Літ.* T / *Аркуш* 1 / *Аркушів* 1
Перевірив		*Волокита А.М.*				
					Принципова Схема Роботи	*НТУУ «КПІ ім. Ігоря Сікорського» ФІОТ Група ІВ-71*
Н.контр.		*Симоненко В.П.*				
Затв.		*Волокита А.М.*				

Додаток Г

ЛІСТИНГ ПРОГРАМИ

до дипломного проєкту

на тему: «Система генерації тематичних текстів на основі нейронних мереж»

Київ – 2021

ЛІСТИНГ ПРОГРАМИ

Файл thesis.py:

```
import argparse

import random


import numpy as np

from tensorflow import keras

from tensorflow.keras import layers



class Utils:

    @staticmethod

    def sample(preds, temperature=1.0):

        """Helper function to sample an index from a probability array"""

        preds = np.asarray(preds).astype("float64")

        preds = np.log(preds) / temperature

        exp_preds = np.exp(preds)

        preds = exp_preds / np.sum(exp_preds)

        probas = np.random.multinomial(1, preds, 1)

        return np.argmax(probas)



class Dataset:

    def __init__(
```

```
        self, text_path="bible/combined_bible.txt", maxlen=40, step=3
    ):
        with open(text_path) as file:
            text = file.read().lower()

        # We remove newlines chars for nicer display
        text = text.replace("\n", " ")

        chars = sorted(list(set(text)))
        char_indices = dict((c, i) for i, c in enumerate(chars))
        indices_char = dict((i, c) for i, c in enumerate(chars))

        self.text = text
        self.chars = chars
        self.char_indices = char_indices
        self.indices_char = indices_char
        self.maxlen = maxlen
        self.step = step

        self.split_to_seqs()

    def split_to_seqs(self):
        sentences = []
        next_chars = []
        for i in range(0, len(self.text) - self.maxlen, self.step):
            sentences.append(self.text[i : i + self.maxlen])
            next_chars.append(self.text[i + self.maxlen])
        self.sentences = sentences
```

```
        self.next_chars = next_chars


    def print_description(self):
        print("Corpus length:", len(self.text))
        print("Total chars:", len(self.chars))
        print("Number of sequences:", len(self.sentences))


    def get_x_y(self):
        x = np.zeros(
                    (len(self.sentences),  self.maxlen,  len(self.chars)),
dtype=np.bool
        )
                y  =  np.zeros((len(self.sentences),  len(self.chars)),
dtype=np.bool)
        for i, sentence in enumerate(self.sentences):
            for t, char in enumerate(sentence):
                x[i, t, self.char_indices[char]] = 1
                y[i, self.char_indices[self.next_chars[i]]] = 1
        return x, y



class Model:
    def __init__(self, dataset, load_folder=None):
        if load_folder is None:
            chars_len = len(dataset.chars)
            shape = (dataset.maxlen, chars_len)
            self.model = keras.Sequential(
                [
                    keras.Input(shape=shape),
```

```
                    layers.LSTM(128, return_sequences=True),
                    layers.LSTM(128, return_sequences=True),
                    layers.LSTM(64),
                    layers.Dense(chars_len, activation="softmax"),
                ]
            )
        else:
            self.model = keras.models.load_model(load_folder)

    def print_description(self):
        self.model.summary()

    def setup_optimizer(
        self, metrics, optimizer="adam", loss="categorical_crossentropy"
    ):
        self.model.compile(
            loss=loss,
            optimizer=optimizer,
            metrics=metrics,
        )

    def train(self, dataset, callbacks, batch_size=32, epochs=40):
        x, y = dataset.get_x_y()
        self.model.fit(
            x,
            y,
            batch_size=batch_size,
            epochs=epochs,
```

```
            callbacks=callbacks,
        )


    def generate(
        self,
        dataset,
        diversity=1.0,
    ):
                start_index = random.randint(0, len(dataset.text) -
dataset.maxlen - 1)
        generated = ""
        seed = dataset.text[start_index : start_index + dataset.maxlen]
        for i in range(400):
            x_pred = np.zeros((1, dataset.maxlen, len(dataset.chars)))
            for t, char in enumerate(seed):
                x_pred[0, t, dataset.char_indices[char]] = 1.0
            preds = self.model.predict(x_pred, verbose=0)[0]
            next_index = Utils.sample(preds, diversity)
            next_char = dataset.indices_char[next_index]
            seed = seed[1:] + next_char
            generated += next_char
        return seed, generated



class ArgumentParser:
    @staticmethod
    def parse_command_line_arguments():
        parser = argparse.ArgumentParser()
```

```
        parser.add_argument(
            "--train", help="Train the model.", action="store_true"
        )
        parser.add_argument(
            "--load",
            help="Load a pretrained model and generate a bit of text.",
        )
        args = parser.parse_args()
        if bool(args.train) == bool(args.load):
                parser.error("You must specify either --train or --load
<path>")
        return args



class Main:
    @staticmethod
    def main():
        args = ArgumentParser.parse_command_line_arguments()
        if args.train:
            Main.train_model()
        elif args.load:
            Main.load_model(args.load)


    @staticmethod
    def load_model(model_path):
        dataset = Dataset()
        model = Model(dataset=dataset, load_folder=model_path)
        seed, generated = model.generate(dataset=dataset, diversity=1.0)
```

```
        print("Seed:", seed)
        print("Generated:", generated)


    @staticmethod
    def train_model():
        dataset = Dataset()
        dataset.print_description()


        model = Model(dataset=dataset)
        model.print_description()
        model.setup_optimizer(
            loss="categorical_crossentropy",
            optimizer="adam",
            metrics=["accuracy"],
        )


        epochs = 40
        batch_size = 128


        callback = keras.callbacks.EarlyStopping(
            monitor="accuracy", patience=5
        )
        checkpoint = keras.callbacks.ModelCheckpoint(
                    "checkpoint", monitor="accuracy", save_best_only=True,
mode="max"
        )
        model.train(
            dataset=dataset,
```

```
            callbacks=[callback, checkpoint],
            batch_size=batch_size,
            epochs=epochs,
        )
        seed, generated = model.generate(
            dataset=dataset,
            diversity=1.0,
        )
        print("Seed:", seed)
        print("Generated:", generated)


if __name__ == "__main__":
    Main.main()
```

Файл create_dataset.py:

```
from pathlib import Path


def create_dataset():
    data = ""
    for book in Path("bible").iterdir():
        if book.suffix == ".txt" and book.name != "combined_bible":
            print("Reading book", book)
            data += book.read_text()
    print("Writing output to bible/combined_bible.txt")
    Path("bible/combined_bible.txt").write_text(data)
if __name__ == "__main__":
    create_dataset()
```