МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РЕАЛІЗАЦІЯ ВИДАЛЕННЯ ІЗ НЕФОРМАТОВАНОГО ТЕКСТУ ВИПАДКОВИХ ГРУП СИМВОЛІВ»**

Виконала: студентка 5 курсу, групи 6.1228-з
спеціальності 122 комп'ютерні науки
(шифр і назва спеціальності)
освітньої програми комп'ютерні науки
(назва освітньої програми)
В.І. Копосова
(ініціали та прізвище)
Керівник старший викладач кафедри комп'ютерних наук, к.т.н. Добровольський Г.А.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент завідувач кафедри програмної інженерії, к.ф.-м.н, доцент Лісняк А.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти бакалавр

Спеціальність 122 комп'ютерні науки
(шифр і назва)

Освітня програма комп'ютерні науки

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

_____ Чопоров С.В.
(підпис)

" ____ " _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТЦІ

Копосовій Валерії Ігорівні
(прізвище, ім'я та по-батькові)

1. Тема роботи Реалізація видалення із неформатованого тексту випадкових груп символів

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

затверджені наказом ЗНУ від « 26 » грудня 2023 року № 103-с

2. Строк подання студентом роботи 23.05.2023

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

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

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

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	Добровольський Г.А. старший викладач кафедри комп'ютерних наук, к.т.н.		
	Лісняк А.О. завідувач кафедри програмної інженерії, к.ф.-м.н, доцент		

7. Дата видачі завдання 27.01.2023

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	27.01.2023	
2.	Аналіз предметної області.	05.02.2023	
3.	Обробка методичних джерел	16.02.2023	
4.	Розробка першого та другого розділу.	02.03.2023	
5.	Розробка третього розділу.	20.04.20023	
6.	Оформлення та нормоконтроль	23.05.2023	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	25.05.2023	

Студент _____ В.І. Копосова
(підпис) (ініціали та прізвище)

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

Нормоконтроль пройдено

Нормоконтролер _____ О.Г.Спиця

(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Реалізація видалення із неформатованого тексту випадкових груп символів»: 66 с., 9 рис., 0 табл., 14 джерел, 2 додатків.

ОБРОБКА ПРИРОДНОГО ТЕКСТУ, МЕТОД ВИЯВЛЕННЯ ПОМИЛОК, МОВНІ МОДЕЛІ, LSTM, PERPLEXITY.

Об’єкт дослідження – текст природної мови з помилковими словами.

Предмет дослідження - застосування символьної мовної моделі для обчислення функції належності слова у мові.

Мета роботи: створення та оцінка якості мовної моделі на рівні символів для покращення видобутого тексту.

Метод дослідження – теоретичний, експериментальний.

Створено нейромережеву мовну модель LSTM, навчену на англійському тексті як послідовності символів. Метою моделі є розділення слів які належать мові, та не належать їй. Розроблена модель призначена для виявлення у вхідних даних помилкових слів з метою покращення якості англійського тексту перед його аналізом іншими методами машинного навчання. Точність отриманої моделі 74-89%, але її можна покращити за рахунок інших навчальних вибірок та оптимізації параметрів нейронної мережі. Символьні мовні моделі дозволяють впізнавати слова, яких не було в навчальних даних.

SUMMARY

Master’s qualifying paper «Implementation of Random Character Groups Cleaning from a Plain Text»: 66 pages, 9 figures, 0 tables, 14 references, 2 supplements.

NATURAL LANGUAGE PROCESSING, ERROR DETECTION TECHNIQUES, LANGUAGE MODELS, LSTM, PERPLEXITY.

Object of the study – natural language text with misspelled words.

Subject of the study – applying a symbolic language model to calculate the word membership function in a language.

Aim of the study: creating and evaluating the quality of the language model at the character level to improve the extracted text.

Methods of research – theoretical, experimental .

An LSTM neural network language model trained on English text as a sequence of characters is created. The goal of the model is to separate words that belong to the language from those that do not. The developed model is designed to detect false words in the input data in order to improve the quality of the English text before analyzing it with other machine learning methods. The accuracy of the resulting model is 74-89%, but it can be improved by using other training samples and optimizing the neural network parameters. Symbolic language models allow you to recognize words that were not in the training data.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

PDF	Portable document format
HTML	Hypertext markup language
NLP	Neuro-linguistic programming
LSTM	Long short-term memory
RNN	Recurrent neural network
SLM	Statistical language model
NLTK	Natural language Toolkit

ВСТУП

Формат PDF [1] є одним із найрозповсюдженіших форматів документів в Інтернеті, після HTML. Документи PDF призначено для візуалізації, але, для успішного застосування засобів штучного інтелекту потрібно мати чистий текст. Чистим текстом тут називається рядок на природній мові, у якому є тільки символи та послідовності символів, що мають сенс із точки зору природної мови. Але часто текст, добутий із PDF є “забрудненим”. Наведемо приклад на різних об’єктах:

формули

а) Оригінал в PDF.

$$f(z) = \frac{1}{1+e^{-z}}, \quad (3)$$

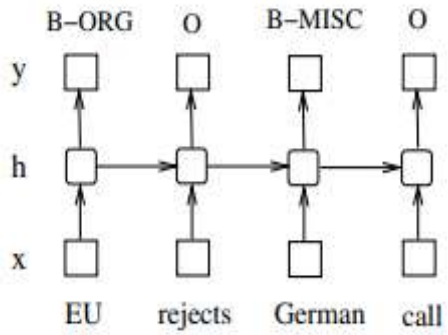
$$g(z_m) = \frac{e^{z_m}}{\sum_k e^{z_k}}. \quad (4)$$

б) Добутий текст.

```
f(z) = 1 1 + e−z , (3) g(zm) = e
zm P k e zk . (4)
```

або діаграми

а) Оригінал в PDF



б) Добутий текст

```
B−ORG O B−MISC O EU
rejects German call x h
Figure 1: A simple RNN
model.
```

або таблиці

а) Оригінал в PDF

б) Добутий текст.

Table 1: Size of sentences, tokens, and labels for training, validation ar

		POS	CoNLL2000	CoNLL20
training	sentence #	39831	8936	14987
	token #	950011	211727	204567
validation	sentence #	1699	N/A	3466
	token #	40068	N/A	51578
test	sentences #	2415	2012	3684
	token #	56671	47377	46666
	label #	45	22	9

Table 1: Size of sentences, tokens, and labels for training, validation and test sets. POS CoNLL2000 CoNLL2003 training sentence # 39831 8936 14987 token # 950011 211727 204567 validation sentence # 1699 N/A 3466 token # 40068 N/A 51578 test sentences # 2415 2012 3684 token # 56671 47377 46666 label # 45 22 9

Як можна побачити з ілюстрацій, видобутий із PDF текст, містить послідовності символів (наприклад «CoNLL2000»), що не належать до словника англійської мови. Такі послідовності знижують якість аналізу текстів засобами NLP.

Тому важливою є задача виявлення у добутому з PDF тексті послідовностей символів, що не належать до природної мови. Названа задача ускладнюється наявністю орфографічних помилок, а також неологізмів, термінів, жаргонізмів - слів, які виконують закони словотворення, схожі на слова із академічного словника, але відсутні у цьому словнику. Складні системи розпізнавання текстів на зображеннях вирішують аналогічну задачу за допомогою статистичних або нейромережевих мовних моделей. Але необхідність обробки великої кількості документів та обмеженість ресурсів ставлять під питання можливість застосування систем розпізнавання зображень для масового видобування тексту із файлів PDF.

Задачами роботи є:

а) пошук визначень основних понять;

б) огляд технологій, які можуть створити мовну модель символьного рівня;

в) застосування LSTM для створення мовної моделі символьного рівня;

г) реалізація обчислення здивування (perplexity) для виявлення незвичайних послідовностей символів;

д) перевірка якості отриманого рішення.

1 ОГЛЯД НАЯВНИХ РІШЕНЬ

Розглянемо, для початку, пакет textract. Цей пакет надає єдиний інтерфейс для вилучення вмісту з будь-якого типу файлу без будь-якої невідповідної розмітки. Textract надає дві основні можливості для цього: інтерфейс командного рядка та пакет засобів розробки ПЗ.

Лістинг 1.1 - пакет textract.

```
extract path/to/file.extension

або пакет python

# деякий файл python
імпортувати текст
text = textract.process("path/to/file.extension")
```

Крім textract існують інші проекти, такі як: pypdfium2, PyMuPDF, Tika, pypdf, pdftotext, pdfminer.six, pdfplumber, Borb. Вони пропонують функцію видобування тексту, забезпечують високу якість видобування[2], але не пропонують перевірки якості добутого тексту, подальшої очистки та застосування мовних моделей.

1.1 Мовні моделі

Мовна модель [2] (Language Models) - це модель, що розподіляє ймовірність слів і послідовностей слів (речень) за допомогою аналізу текстових даних. Зазвичай такі моделі використовуються для того, щоб передбачити

наступне слово у реченні, враховуючи всі попередні слова. На практиці це дає ймовірність того, що певна послідовність слів є «дійсною».

Нас часто цікавить ймовірність того, що наша модель надає повній пропозиції W, що складається з послідовності слів.

$$P(W) = P(w_1, w_2, \dots, w_N), \tag{1.1}$$

де w_1, w_2, ..., w_N - послідовність слів,

P - навчена мовна модель.

Наприклад, ми хотіли б, щоб модель надавала більш високі ймовірності пропозиціям, які є реальними та синтаксично правильними.

Модель unigram працює лише на рівні окремих слів. Враховуючи послідовність слів W, модель unigram виведе ймовірність:

$$P(W) = P(w_1)P(w_2) \dots, P(w_N), \tag{1.2}$$

де w_1, w_2, ..., w_N - послідовність слів,

P - навчена мовна модель.

Індивідуальні ймовірності $P(w_i)$ можуть, наприклад, оцінюватись на основі частоти слів у навчальному корпусі.

Натомість n-грамна модель розглядає попередні слова, щоб оцінити наступне. Наприклад, модель триграми розглядатиме попередні 2 слова, так що:

$$P(W) = P(w_1)P(w_1|w_2)P(w_3|w_2, w_1) \dots P\left(w_N|w_{N-1}, w_{N-2}\right), \tag{1.3}$$

де w_1, w_2, w_3, w_N- послідовність слів,

w_{N-1}, w_{N-2}- попередні слова,

P - навчена мовна модель.

Мовні моделі можуть бути вбудовані у складніші системи, щоб

допомогти у виконанні мовних завдань, таких як переклад, класифікація, розпізнавання мови тощо.

1.1.1 Символьні мовні моделі

Символьна мовна модель (Character Language Model) - модель, основне завдання якої - передбачити наступний символ, беручи до уваги всі попередні символи у послідовності даних. Тобто, ця модель генерує текст посимвольно.

Перевагою мовних моделей є їхній невеликий словниковий запас та гнучкість в обробці будь-яких слів, розділових знаків та іншої структури документа.

Тим не менш, в області CLM пропонують перспективні можливості для загального, гнучкого і потужного підходу до мовного моделювання.

Кількість символів, що використовуються як вхідні дані, також визначатиме кількість символів, які необхідно надати моделі, щоб отримати перший передбачений символ.

Після того, як перший символ згенерований, його можна додати до вхідної послідовності та використовувати як вхідні дані для моделі для генерації наступного символу.

Більш довгі послідовності надають більше контексту моделі, щоб дізнатися, який символ виводити наступним, але вимагають більше часу для навчання і накладають велике навантаження на заповнення моделі при генерації тексту.

1.1.2 Статистичні мовні моделі

Статистична мовна модель (Statistical Language Model) - це математична модель, яка визначає ймовірність послідовності слів. Вона використовує дані з

великих корпусів тексту для оцінки частоти використання слів та їх взаємодії в мовленні. Статистичні мовні моделі використовуються в багатьох застосунках, таких як машинний переклад, розпізнавання усної мови, генерація тексту та аналіз тексту.

Статистична мовна модель описується із використанням ймовірностей для представлення взаємодії між словами у мові. Основні види моделей, що використовуються в SLM, є:

а) N-грамна модель (література на цю тему [3],[7],[9]):

в даній моделі ймовірність слова визначається відносно його N попередніх слів. Ймовірність визначається як:

$$P(w_i|w_1, w_2 \ldots, w_{i-N+1}), \tag{1.4}$$

де w_i - це *i*-те слово,

w_1, w_2, ..., w_{i-N+1} - це попередні *N-1* слів

P - навчена мовна модель.

б) Модель Маркова, (більше про цю модель [4],[6],[8]),

у цій моделі ймовірність кожного слова визначається відносно його попереднього слова. Ймовірність визначається як:

$$P(w_i|w_{i-1}), \tag{1.5}$$

де w_i - це *i*-те слово,

w_{i-1} - це попереднє слово

P - навчена мовна модель.

в) Рекурентна нейронна мережа (RNN): у цій моделі ймовірність кожного слова визначається відносно всіх попередніх слів.

Статистичні мовні моделі символьного рівня — це тип мовної моделі, яка працює на рівні символів, на відміну від рівня слів. У цих моделях мета полягає в тому, щоб передбачити наступний символ у послідовності, враховуючи контекст попередніх символів. Моделі на рівні символів мають кілька переваг порівняно з моделями на рівні слів: вони можуть обробляти орфографічні помилки та друкарські помилки, можуть обробляти текст, написаний різними шрифтами та алфавітами, і можуть працювати зі словами поза словниковим запасом.

Моделі символьного рівня можна створювати за допомогою різних алгоритмів машинного навчання, таких як моделі Маркова, рекурентні нейронні мережі (RNN) і згорткові нейронні мережі (CNN). У цих моделях вхідними даними є послідовність символів, і модель вчиться передбачати розподіл ймовірностей за наступним символом, враховуючи послідовність попередніх символів. Результати моделі можна використовувати для таких завдань, як створення тексту, виправлення орфографії та класифікація тексту.

Моделі рівня символів застосовувалися в різних областях, таких як аналіз настроїв, визначення авторства та класифікація тексту, серед іншого. Ці моделі показали хороші результати і стають все більш популярними в області обробки природної мови.

Статистичні моделі на рівні символів є, відносно, нещодавньою розробкою в області природної мови, і розглядаються в таких книгах: «Deep Learning for Natural Processing» [5], та інших [10,11].

1.1.3 RNN

Люди не починають своє мислення з нуля щосекунди. Коли людина читає текст, вона розуміє кожне слово з урахуванням розуміння попередніх слів. Думки людини мають сталість.

Традиційні нейронні мережі не можуть цього зробити, і це видається серйозним недоліком. Наприклад, якщо ви хочете класифікувати, яка подія відбувається у кожний момент фільму. Незрозуміло, як традиційна нейронна мережа може використовувати свої міркування про попередні події у фільмі, щоб інформувати про пізніші.

Рекурентні нейронні мережі вирішують проблему. Це мережі з петлями (див. рис. 1.1), які дозволяють зберігати інформацію.

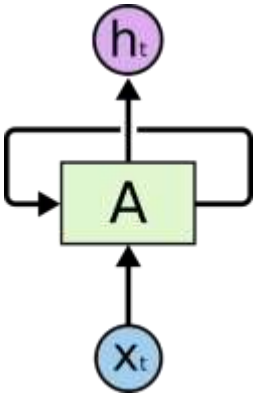


Рисунок 1.1 – Цикл рекурентної нейронної мережі.

На наведеній вище діаграмі фрагмент нейронної мережі “A” дивиться на деякі вхідні дані Ікс, і виводить значення Час. Цикл дозволяє передавати інформацію від одного кроку мережі до іншого.

Рекурентну нейронну мережу можна розглядати як декілька копій однієї мережі, кожна з яких передає повідомлення наступній. Розглянемо, що станеться, якщо ми розгорнемо цикл (рис. 1.2):

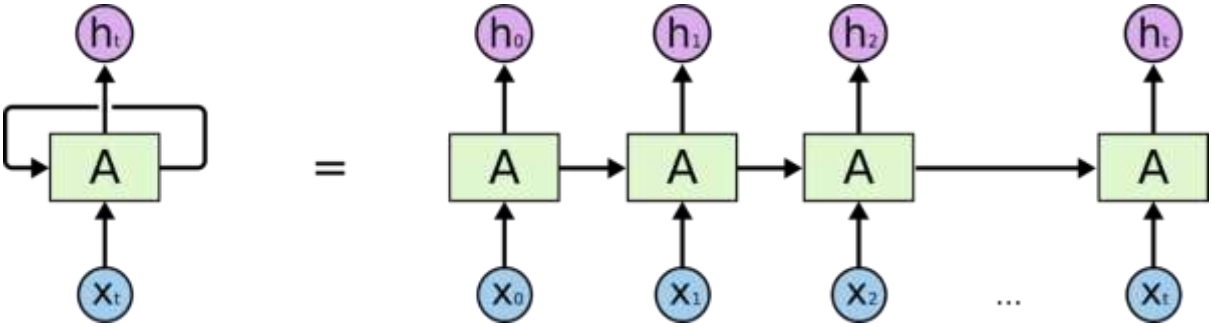


Рисунок 1.2 -Розгорнутий цикл рекурентної нейронної мережі.

Ця ланцюгова природа показує, що рекурентні нейронні мережі тісно пов'язані з послідовностями та списками. Це - природна архітектура нейронної мережі таких даних.

І вони часто використовуються. За останні кілька років RNN досягли неймовірного успіху у вирішенні найрізноманітніших завдань: розпізнавання мови, мовне моделювання, переклад, субтитри до зображень тощо.

Істотним для цих успіхів є використання «LSTM», особливого виду рекурентної нейронної мережі, яка для багатьох завдань працює набагато краще, ніж стандартна версія. Майже всі результати, засновані на нейронних рекурентних мережах, досягаються з їх допомогою.

Однак, однією з привабливих сторін RNN є ідея, що вони можуть пов'язувати попередню інформацію з поточним завданням, наприклад, використання попередніх відеокадрів може сприяти розумінню поточного кадру. Якби RNN могли це робити, вони були б надзвичайно корисними. Але чи можуть вони? І так, і ні.

Іноді потрібно лише переглянути останню інформацію, щоб виконати поточне завдання. Наприклад, якщо розглянути мовну модель, яка намагається передбачити наступне слово з урахуванням попередніх. Якщо ми намагаємося передбачити останнє слово в «Хмари у небі», нам не потрібен додатковий контекст — цілком очевидно, що наступним словом буде небо. У таких випадках, коли розрив між релевантною інформацією та місцем, де вона потрібна, невеликий, RNN можуть навчитися використовувати минулу інформацію.

Але є випадки, коли потрібне більше контексту. Наприклад, вгадати останнє слово у тексті: «Я виріс у Франції… Я вільно розмовляю французькою». Нещодавня інформація передбачає, що наступне слово, ймовірно, є назвою мови, але якщо ми хочемо звузити коло мов, нам потрібен ранній контекст Франції. Цілком можливо, що розрив між релевантною інформацією та точкою, де вона необхідна, може стати дуже великим.

На жаль, у міру того, як цей розрив збільшується, RNN стають нездатними навчитися зв'язувати інформацію.

1.2 LSTM

LSTM [13] (від англ. long short-term memory - довгостроково-короткочасна пам'ять) - це особливий вид RNN, здатний вивчати довгострокові залежності. Вони надзвичайно добре працюють при вирішенні різних завдань і в даний час широко використовуються.

LSTM розроблено спеціально для того, щоб уникнути проблеми довгострокової залежності. Запам'ятовування інформації протягом тривалого часу є їхньою поведінкою за умовчанням, а не тим, чого вони намагаються навчитися.

Всі рекурентні нейронні мережі мають вигляд ланцюжка модулів нейронної мережі, що повторюються (див. рис. 1.3). У стандартних RNN цей повторюваний модуль буде мати дуже просту структуру, наприклад, один шар *tanh.*

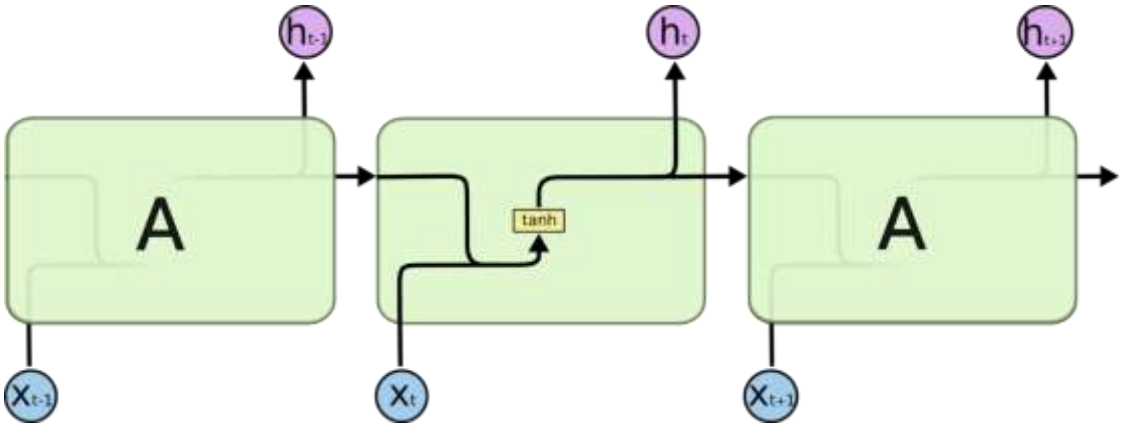


Рисунок 1.3 – Ланцюжок модулів нейронної мережі.

LSTM також мають структуру, подібну до ланцюжка, але повторюваний модуль має іншу структуру (рис 1.4). Замість одного шару нейронної мережі їх чотири, що взаємодіють особливим чином.

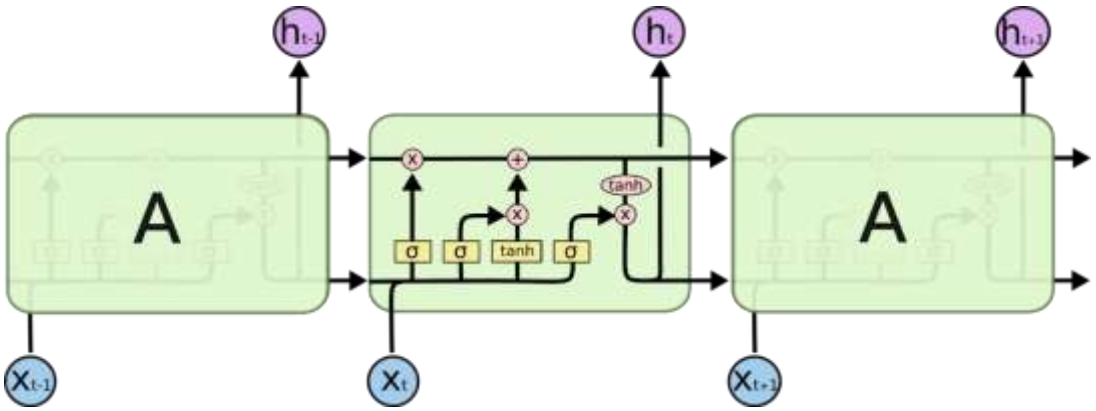


Рисунок 1.4 – Чотири взаємодіючі шари в LSTM, в модулі, що повторюється.

1.2.1 Основна ідея LSTM

Ключем до LSTM [14] є стан комірки, горизонтальна лінія, що проходить через верхню частину діаграми.

Стан комірки схожий на конвеєрну стрічку. Він проходить прямо по всьому ланцюжку, лише з невеликими лінійними взаємодіями (рис 1.5). Інформації дуже легко йти по ньому без змін.

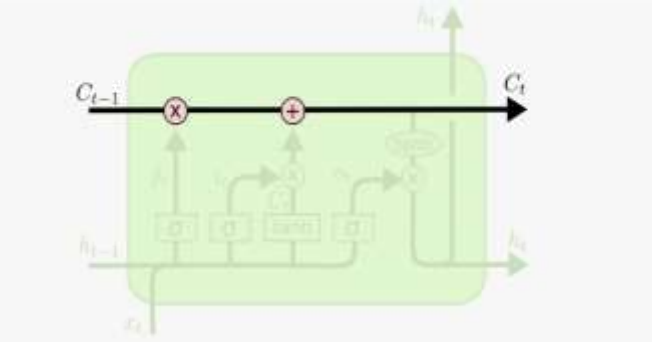


Рисунок 1.5 – Стан комірки.

LSTM має можливість видаляти або додавати інформацію у стан комірки, ретельно регульовану структурами, які називаються воротами.

Ворота - це спосіб опционального пропуску інформації. Вони складаються з шару сигмовидної нейронної мережі та операції поточкового множення (рис. 1.6).

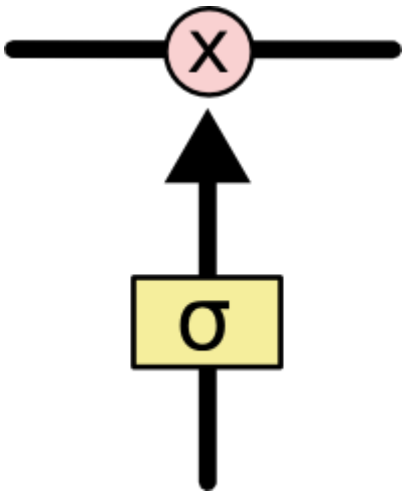


Рисунок 1.6 – Ворота LSTM.

Сигмовидний шар виводить числа від нуля до одиниці, що описують, скільки кожного компонента має бути пропущено. Нульове значення означає "нічого не пропускати", а значення "одиниця" означає "пропускати все".

LSTM має три таких воріт для захисту та керування станом комірки.

1.3 Що таке perplexity

Perplexity [12] - це показник для оцінки моделей у обробці природної мови (NLP).

Ми можемо використовувати два різні підходи для оцінки та порівняння мовних моделей:

а) зовнішня оцінка. Це включає оцінку моделей шляхом їх використання в реальному завданні (наприклад, машинний переклад) і розгляд їх остаточної втрати/точності. Це найкращий варіант, оскільки це єдиний спосіб наочно побачити, як різні моделі впливають на потрібне нам завдання. Однак це може бути дорого та повільно у обчислювальному відношенні, оскільки потребує підготовки повної системи;

б) внутрішня оцінка. Це включає пошук деякої метрики для оцінки самої мовної моделі, не беручи до уваги конкретні завдання, для яких вона буде використовуватися. Хоча внутрішня оцінка не така зручна, як зовнішня оцінка (як кінцевий показник), але це корисний спосіб швидкого порівняння моделей.

Треба, щоб потрібна модель надавала високі ймовірності пропозиціям, які є реальними та синтаксично правильними, а низькі ймовірності - помилковими та неправильними.

Припускаючи, що цей набір даних складається з пропозицій, які є реальними і правильними, це означає, що найкращою моделлю буде та, яка присвоює тестовому набору найбільшу ймовірність. Якщо модель надає тестовому набору високу ймовірність, це означає, що вона не здивована, побачивши це, що означає, що вона добре розуміє, як працює мова.

Однак варто відзначити, що набори даних можуть містити різну кількість слів та речень. Очевидно, що додавання більшої кількості пропозицій вносить більше невизначеності, тому більший набір тестів, ймовірно, матиме меншу ймовірність, ніж менший. В ідеалі - мати метрику, яка залежить від розміру набору даних. Це можливо отримати, нормалізуючі можливість набору тестів на загальну кількість слів, що дало б показник для кожного слова.

Як це зробити? Якщо ми хотіли б нормалізувати суму деяких термінів, ми могли б просто розділити її на кількість слів, щоб отримати показник для кожного слова. Але ймовірність послідовності слів задається добутком.

Наприклад, за допомогою моделі unigram:

$$P(W) = P(w_1, w_2, \dots, w_N) = P(w_1)P(w_2) \dots, P(w_N) = \prod_{i=1}^{N} \quad P(w_i), \tag{1.6}$$

де w_1, w_2, ..., w_N - послідовність слів,

w_i - це *i*-те слово,

P - навчена мовна модель.

Як нормалізувати цю ймовірність? Це простіше зробити, подивившись на логарифмічну ймовірність, яка перетворює добуток на суму:

$$ln \quad ln\,(P(W)) = ln\ ln\left(\prod_{i=1}^{N} \quad P(w_i)\right) = \sum_{i=1}^{N} \quad lnP(w_i)\ , \tag{1.7}$$

де w_i - це *i*-те слово,

P - навчена мовна модель.

Тепер можна нормалізувати це, розділивши на N, щоб отримати логарифмічну ймовірність для кожного слова:

$$\frac{ln \quad ln\,(P(W))}{N} = \frac{\sum_{i=1}^{N} \quad ln\ ln\,P(w_i)}{N}$$

$$e^{\frac{ln\,ln\,(P(W))}{N}} = e^{\frac{\sum_{i=1}^{N} \quad ln\,ln\,P(w_i)}{N}}$$

$$(e^{ln\,ln\,(P(W))})^{\frac{1}{N}} = (e^{\sum_{i=1}^{N} \quad lnP\,(W_i)})^{\frac{1}{N}}$$

$$(P(W))^{\frac{1}{N}} = \left(\prod_{i=1}^{N} \quad P(w_i)\right)^{\frac{1}{N}}, \tag{1.8}$$

де w_i - це *i*-те слово,

N - довжина,

P - навчена мовна модель.

Вийшла нормалізація, за допомогою корня N.

Тепер, повертаючись до початкового рівняння можна побачити, що його можна інтерпретувати як зворотню ймовірність тестового набору, нормалізовану на кількість слів у тестовому наборі:

$$PP(W) = \frac{1}{P(w_1, w_2, \dots, w_N)^{\frac{1}{N}}} = \sqrt[N]{\frac{1}{P(w_1, w_2, \dots, w_N)}}, \tag{1.9}$$

де w_1, w_2, ..., w_N - послідовність слів,

P - навчена мовна модель.

Оскільки ми приймаємо зворотну ймовірність, менший perplexity вказує на кращу модель.

Ще ми маємо таке поняття, як крос-ентропія мовної моделі. Ентропію можна інтерпретувати як середню кількість біт, необхідну для зберігання інформації в змінній, і вона задається:

$$E(p) = -\sum_i \quad p(x_i)p(x_i), \tag{1.10}$$

де p - фактичний розподіл.

А задається так:

$$H(p,q) = -\sum_i \quad p(x_i)q(x_i), \tag{1.11}$$

де p - фактичний розподіл,

q - очікуваний розподіл.

Це можна інтерпретувати як середню кількість бітів, необхідних для зберігання інформації у змінній, якщо замість реального розподілу ймовірностей p використати очікуваний розподіл q.

Очевидно, що ми не можемо знати реального значення p, але, враховуючи досить довгу послідовність слів, можна апроксимувати крос-ентропію для кожного слова, використовуючи теорему Шеннона-Макміллана-Бреймана:

$$H(p,q) \approx -\frac{1}{N}q(W), \tag{1.12}$$

де q - очікуваний розподіл,

N - довжина,

W - послідовність слів.

В наступному кроці перепишемо це, щоб воно відповідало позначенню, використаному в попередньому розділі. Враховуючи послідовність слів W довжину N та навчену мовну модель P, наближаємо крос-ентропію як:

$$H(W) = -\frac{1}{N}P(W) = -\frac{1}{N}P(w_{1,}w_2,\ldots,w_N)\ , \qquad (1.13)$$

де N - довжина,

w_1, w_2, ..., w_N - послідовність слів,

P - навчена мовна модель.

Ще раз подивимося на визначення perplexity:

$$PP(W) = 2^{-\frac{1}{N}P(w_1,w_2,\ldots,w_N)}\ , \qquad (1.14)$$

де N - довжина,

w_1, w_2, ..., w_N - послідовність слів,

P - навчена мовна модель,

P(W) - це середня кількість бітів, необхідна для кодування кожного слова. Це означає, що perplexity $2^{P(W)}$ - це середня кількість слів, які можуть бути закодовані з використанням P(W).

2. МЕТОД ВИЯВЛЕННЯ СЛІВ, ЩО НЕ Є ЧАСТИНОЮ МОВИ

Метод використовує припущення, що кожна мова може бути змодельована за допомогою статистичної моделі символьних n-грам. Отримуючи послідовність символів, така модель може обчислити її ймовірність. Обчислена ймовірність послідовності символів може бути мірою належності до мови.

Відповідний метод обчислення складається із наступних кроків:

а) створити мовну модель на основі корпусу текстів. В даній роботі використовується нейромережева модель типу LSTM, призначена для моделювання послідовностей. Під час тренування моделі на вхід подається послідовність символьних біграм або триграм;

б) Під час застосування:

1) модель отримує слово;

2) слово перетворюється на послідовність А символьних біграм;

3) на осноі вхідної послідовності біграммодель обчислює послідовність ймовірностей А та відповідне значення perplexityA;

4) на основі вхідної послідовності символьних біграм модель прогнозує ідеальну послідовність В біграм;

5) на основі вхідної послідовності символьних біграм модель прогнозує ідеальну послідовність В біграм;

6) на основі вхідної послідовності В обчислюється відповідна послідовність ймовірностей В та відповідне значення perplexityB;

7) на основі вхідної послідовності В обчислюється відповідна послідовність ймовірностей В та відповідне значення perplexityB;

8) обчислюється міра відмінності perplexityA та perplexityB;

9) в залежності від заданого заздалегідь порогового значення, приймається рішення про належність слова до мови.

Названі вище кроки використовують припущення, що вихідні значення останнього рівня мережі LSTM, який має тип softmax, можуть інтерпретуватися як ймовірність кожної біграми.

Замість біграм модель може використовувати уніграми, триграми і т. д. в залежності від наявних обчислювальних ресурсів.

Подробиці кожного кроку описуються нижче.

2.1 Розділення тексту на слова

Токенізація використовує NLTK. NLTK (англ. Natural Language ToolKit) — це бібліотека, написана на Python для символічної та статистичної обробки природної мови.

Встановити NLTK можна за допомогою наведеного нижче коду:

Лістинг 2.1 Встановлення NLTK.

```
pip install --user -U nltk;
```

NLTK містить модуль під назвою `word_tokenize`, який виконує токенізацію слів наступним чином**:**

Лістинг 2.2 - Процес токенізації.

```
from nltk.tokenize import word_tokenize.
text = """Місія SpaceX, заснована в 2002 році, полягає в тому, щоб дозволити людям стати багатопланетним видом, побудувавши самопідтримуване місто на Марсі. У 2008 році Falcon 1, компанії SpaceX, став першою приватно розробленою ракетою-носієм на рідкому паливі, яка вилетіла на орбіту Землі.""".

word_tokenize(text)
```

```
Output: ['Місія', 'SpaceX', ',', 'заснована', 'в',
'2002', 'році', ',', 'полягає', 'в', 'тому', ',', 'щоб',
'дозволити', 'людям', 'стати', 'багатопланетним', 'видом', ',',
'побудувавши', 'самопідтримуване', 'місто', 'на', 'Марсі', '.',
'У', '2008', 'році', 'Falcon', '1', ',', 'компанії',
'SpaceX', ',', 'став', 'першою', 'приватно', 'розробленою',
'ракетою', '-', 'носієм', 'на', 'рідкому', 'паливі', ',', 'яка',
'вилетіла', 'на', 'орбіту', 'Землі', '.']
```

Звернемо увагу на те, що NLTK розглядає пунктуацію як маркер.

2.2 Підготовка вхідних даних для символьної мовної моделі LSTM

Тепер виконуємо базову попередню обробку набору даних, таку, як опускання, тощо. Також витягуємо деякі статистичні дані про набір даних, такі як кількість унікальних токенів, які пізніше можна передати як параметр під час навчання нашої моделі.

Попередня обробка включає наступні кроки:

а) #завантажує дані, попередньо обробляє їх, та зберігає корпус у raw_text;

б) #видаляє нові рядки;

в) #видаляє всі спеціальні символи;

г) #прибирає парні періоди;

д) #конвертує корпус у нижній регістр;

е) #використовує попередньо оброблені дані та створює raw_text;

ж) #створює відображення унікальних символів у цілі числа;

з) #підготовує набір даних, де введенням є послідовність із 100 символів, а метою є наступний символ;

и) #швидко кодує вихідну змінну Y.

Лістинг 2.3 - Процес обробки набору даних.

```
    string.punctuation    =    string.punctuation    +'‘'+'’'+'-
'+'''+'`'+'—'
    string.punctuation = string.punctuation.replace('.', '')
    #завантажує дані та попередньо обробляє дані та зберігає
корпус у raw_text:
    raw_text = open('/content/sample_data/corpus.txt', encoding =
'utf8').read()

    file_nl_removed = ""
    for line in raw_text:
     line_nl_removed = line.replace("\n", " ")
     #видаляє нові рядки:
     file_nl_removed += line_nl_removed

    file_p = "".join([char for char in file_nl_removed if char
not in string.punctuation])
    #видаляє всі спеціальні символи:

    string.punctuation = string.punctuation + '.'
    file_q = "".join([char for char in file_p if char not in
string.punctuation])
    #прибирає парні періоди:
    words = nltk.word_tokenize(file_q)

    preprocessed_text = file_p.lower()
    #конвертує корпус у нижній регістр:

    #використовує попередньо оброблені дані та створює raw_text:
    raw_text = preprocessed_text

    #створює відображення унікальних символів у цілі числа:
    chars = sorted(list(set(raw_text)))
    char_to_int = dict((c, i) for i, c in enumerate(chars))
    int_to_char = dict((i, c) for i, c in enumerate(chars))
```

```
#підготовує набір даних, де введенням є послідовність із 100 символів, а метою є наступний символ:
seq_length = 100

dataX = []
dataY = []

for i in range(0, n_chars - seq_length, 1):
 seq_in = raw_text[i:i + seq_length]
 seq_out = raw_text[i + seq_length]

 dataX.append([char_to_int[char] for char in seq_in])
 dataY.append(char_to_int[seq_out])

n_patterns = len(dataX)
print ("Total Patterns: ", n_patterns) #змінює форму X to be [зразки, часові кроки, функції]
X = numpy.reshape(dataX, (n_patterns, seq_length, 1))

#швидко кодує вихідну змінну:
from keras.utils import np_utils
y = np_utils.to_categorical(dataY)
```

2.3 Структура символьної мовної моделі LSTM та вхідні параметри

Лістинг 2.4. - Код, який задає структуру моделі.

```
embedding_dim =100
max_length =100
model = Sequential()
model.add(Embedding(n_vocab,embedding_dim,input_length=max_length))
model.add(LSTM(256))
model.add(Dropout(0.2))
model.add(Dense(y.shape[1], activation='softmax'))
```

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

Embedding - спосіб відображення символів у N-вимірний простір дійсних чисел.

LSTM - це реалізація LSTM, описаної у попередньому розділі.

Dropout - метод вирішення проблеми перенавчання в нейронних мережах.

Dense - це найчастіше використовуваний шар нейронної мережі. Він робить нижню операцію на вході і повертає її на вихід.

2.3.1 Embedding Layer

Embedding Layer (шар вбудовування) є одним з доступних шарів в Keras. Він використовується в додатках, пов’язаних з обробкою природної мови, але його також можна використовувати в інших завданнях, які включають нейронні мережі.

В чому необхідність цього шару? Як відомо, при роботі з текстовими даними, потрібно конвертувати їх у числа, перш ніж передавати в будь-яку модель машинного навчання, включаючи нейронні мережі. Для полегшення можна порівняти з категоріальними змінними. Ми використовуємо одноразове кодування для перетворення категоріальних ознак у числа. Для цього створюються фіктивні функції для кожної категорії,які заповнюються 0 та 1.

Аналогічно, якщо використовувати однократне кодування для слів у текстових даних, матимемо фіктивну ознаку для кожного слова, що означає 10 000 ознак для словника з 10 000 слів. Такий підхід до вбудовування не є доцільним, оскільки вимагає великого простору для зберігання векторів слів і знижує ефективність моделі.

Embedding Layer дозволяє перетворити кожне слово у вектор фіксованої довжини визначеного розміру. Результуючий вектор є щільним і містить реальні значення, а не лише 0 та 1. Фіксована довжина векторів слів допомагає краще представляти слова, зменшуючи їхні розміри.

Таким чином, Embedding Layer працює як таблиця пошуку. Слова є ключами в цій таблиці, а щільні вектори слів - значеннями.

2.3.2 Dropout Layer

Dropout Layer (шар виключення) є одним із найпопулярніших методів регуляризації для зменшення перенавчання в моделях глибокого навчання. Перенавчання в моделі відбувається тоді, коли вона показує більшу точність даних навчання, але меншу точність даних тестування, або невидимих даних. У техніці Dropout деякі нейрони у прихованих або видимих шарах випадковим чином відкидаються або пропускаються. Експерименти показують, що ця техніка виключення впорядковує модель нейронної мережі, щоб створити надійну модель, яка не переповнюється. Розглянемо функцію виключення разом із параметрами для кращого розуміння використання:

Лістинг 2.5 - Функція виключення.

```
keras.layers.Dropout(rate, noise_shape = None, seed = None)
```

Параметри функції пояснюються наступним чином:

а) rate – це частка блоку введення, яку потрібно виключити. Це буде від 0 до 1;

б) noise_shape – представляє шару нейронної мережі, до якої буде застосовано виключення. Наприклад, вхідна форма «batch_size, timesteps,

features». Потім, щоб застосувати виключення в часових кроках, «batch_size, 1, features» потрібно вказати як noise_shape;

в) seed − ціле число Python використовується як випадкове початкове число.

Dropout Layer фактично застосовується для кожного шару в нейронних мережах, і може використовуватися з іншими шарами Keras для повноз´єднаних, згорткових, агрерувальних шарів тощо.

Його можна застосувати до вхідного, окремого, або всіх прихованих шарів, але не можна застосувати до вихідного шару. Діапазон значень для вилучення — від 0 до 1. Чим більше число, тим більше вхідних значень буде вилучено.

2.3.3 Dense softmax layer (щільний шар)

Keras softmax успадковується від Layer і визначається в модулі tensorflow. Елементи у вихідному векторі знаходяться в діапазоні від 0 до 1, і його сума буде дорівнювати 1. Кожен вектор обробляється незалежно. Аргумент осі буде встановлено, коли функція введення осі буде застосована. Він використовується для активації останнього рівня для мережі класифікації.

Результат keras softmax інтерпретується як розподіл ймовірності. Вектор softmax обчислюється як exp(x). Вхідні значення обчислені в кінцевому розподілі ймовірності.

Глибоке навчання є основною підсферою в рамках машинного навчання. Те саме підтримується багатьма бібліотеками, такими як tensor flow. Keras — найважливіша та проста у використанні бібліотека Python, яка була побудована на вершині бібліотек для навчання. Keras пропонує збір даних, який використовувався для навчання та тестування моделі.

2.4 Функція прогнозування послідовності символів

Тепер ми пишемо функцію, щоб отримати фактичний символ із прогнозу:

а) генерує послідовність, подібну до описаних вище методів;

б) отримує згенерований рядок за допомогою моделі.

Лістинг 2.6 - Код для визначення функції predict_next_n_chars.

```
def predict_next_n_chars(pattern, n):
     for i in range(n):
 x = numpy.reshape(pattern, (1, len(pattern), 1))
 prediction = model1.predict(x, verbose=0)
 print (int_to_char[numpy.argmax(prediction)], end = '')

 seq_in = [int_to_char[value] for value in pattern]
 pattern.append(numpy.argmax(prediction))
 pattern = pattern[1:len(pattern)]
```

Цей код приймає два аргументи: pattern і n. Аргумент pattern — це список цілих чисел, що представляють послідовність символів, а аргумент n — це ціле число, яке представляє кількість символів, які слід передбачити після вхідної послідовності.

Функція використовує модель нейронної мережі, яка називається model1, щоб передбачити наступні n символів за заданим шаблоном введення. Модель приймає послідовність символів і повертає розподіл ймовірностей за можливими наступними символами. Функція повторюється n разів, і в кожній ітерації виконуються наступні кроки:

а) список pattern переформовано в масив numpy із формою (1, len(pattern), 1), де перший вимір представляє кількість зразків, другий представляє довжину вхідної послідовності, а третій вимір представляє кількість

функції за часовий крок. Цей формат підходить для введення в модель нейронної мережі;

б) Model1 використовується для прогнозування розподілу ймовірностей за наступним символом за вхідним шаблоном;

в) Індекс символу з найвищою ймовірністю отримується за допомогою numpy.argmax(prediction);

г)Передбачуваний символ друкується за допомогою int_to_char[numpy.argmax(prediction)];

д) Індекс передбаченого символу додається до списку pattern;

е) Перший елемент списку pattern видаляється, для того, щоб він містив лише останні символи;

ж) Оновлений список pattern використовується як вхідні дані для наступної ітерації.

Наприкінці функції передбачається та друкується n символів, а функція повертає None. Реалізація передбачає, що numpy та int_to_char були визначені раніше в коді.

2.5 Функція перетворення послідовності символів на послідовність ймовірностей

Моделі довготривалої пам’яті або LSTM використовуються для вирішення проблеми короткочасної пам’яті за допомогою вентилів, що регулюють потік інформації. У цих моделях є механізми, які вирішують, чи зберігати інформацію чи ні, що дозволяє зберігати важливу інформацію протягом багато часу. Завдяки своїй здатності вивчати довгострокові залежності, вони широко використовуються в машинному перекладі, розпізнаванні мови, розпізнаванні та генерації рукописного тексту, мовному

моделюванні та перекладі, синтезі мови та багатьох інших завданнях глибокого навчання.

Класифікація послідовностей – це завдання прогнозного моделювання, коли існує деяка послідовність вхідних даних у просторі або часі, і ми хочемо передбачити категорію для цієї послідовності. Послідовності можуть відрізнятися за довжиною, складатися з дуже великого словника вхідних символів і можуть зажадати від моделі вивчення довгострокових залежностей між символами у вхідній послідовності, що робить цю проблему складною для RNN, але легко вирішується LSTM. Також ми отримаємо вектор передбачення ймовірностей кожного символу.

2.6 Функція обчислення perplexity

Перплексія PP дискретного розподілу ймовірності p визначається як:

$$PP\ (p) = 2^{H(p)} = 2^{\Sigma_x \quad p(x) log_2 p(x)} = \prod_x \quad p(x)^{-p(x)} \qquad (2.1),$$

де H(p) ентропія (у бітах) розподілу,

x – діапазон подій.

Підстава логарифму не обов’язково має дорівнювати 2: перплексія не залежить від підстави логарифму за умови, що ентропія та показова функція мають одну й ту саму основу. Перплексія – це показова функція від ентропії, яка є більш точною величиною. Ентропія – це міра очікуваної, або “середньої” кількості бітів, необхідних для кодування результату випадкової змінної, наприклад, використовуючи теоретично оптимальний код змінної довжини. Перплексія випадкової змінної X може бути визначена як перплексія розподілу за можливими значеннями x. У окремому випадку, коли p моделює k-сторонню гральну кістку (рівномірний розподіл по k дискретним подіям), її перплексія дорівнює k. Випадкова величина з перплексією k має таку ж невизначеність, як і k-стороння гральна кістка.

Модель з невідомим розподілом ймовірності величини p може створена на основі навчальної вибірки, взятої з p. Перплексія моделі q обчислюється як:

$$b^{-\frac{1}{N}\sum_{i=1}^{N} \log_b q(x_i)}, \qquad (2.2)$$

де b зазвичай дорівнює 2.

Показник експоненти також може розглядатися як перехресна ентропія

$$H(p,q) = -\sum_x p(x)\log_2 q(x) H(p,q) = -\sum_x p(x)\log_2 q(x). \qquad (2.3)$$

Таким чином, можна оцінювати перплексію через кросентропію.

3 СКЛАДАННЯ PYTON ТА KERAS В ОДНУ ПРОГРАМУ

Метод оцінки якості : як псуємо слова, берем і додаємо випадковий символ

які числа обчислюються якість. бінарної класифікції

переписати про методи оцінки якості бінарної класифікції

Робота складається з двох частин:

а) Програми, яка навчає моделі та зберігає результат на диск;

б) Програми, яка використовує заздалегідь навчену модель для аналізу тексту.

Для навчання моделі була використана перші 20 % тексту книги Charles Dickens «Oliver Twist or the Parish Boy's progress».

Лістинг 3.1 - код імпорту бібліотек та необхідних інструментів.

```
import numpy
import re
import pandas as pd
import numpy as np
import keras
import string
import nltk
import pickle
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import LSTM
from keras.layers import Embedding
from keras.callbacks import ModelCheckpoint
from keras.utils import np_utils
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
```

Лістинг 3.2 - Шлях до файлів моделі.

```
MODEL_LTSV1_NAME='../content/model/DickensLTSV1.model'
MODEL_LTSV2_NAME='../content/model/DickensLTSV2.model'
TEXT_FILE_NAME = '../content/sample_data/Dickens.txt'
```

Далі - читаємо текст та проводимо необхідні маніпуляції:

Лістинг 3.3 - Код видалення знаків пунктуації.

```
raw_text = open(TEXT_FILE_NAME, encoding = 'utf8').read()
file_nl_removed = ""
for line in raw_text:
   line_nl_removed = line.replace("\n", " ")
#removes newlines:
   file_nl_removed += line_nl_removed

file_p = "".join([char for char in file_nl_removed if char not in
string.punctuation])
#removes all special characters
preprocessed_text = file_p.lower()
```

Записуємо дані про навчання, скільки даних пішло на вхід:

Лістинг 3.4 - Заповнення даних для навчання.

```
raw_text = preprocessed_text

for i in range(0, n_chars - seq_length, 1):
   seq_in = raw_text[i:i + seq_length]
   seq_out = raw_text[i + seq_length]
   dataX.append([char_to_int[char] for char in seq_in])
   dataY.append(char_to_int[seq_out])
```

Збудуємо мережу яка складається з наступних шарів:

Лістинг 3.5 - Побудова мережі.

```
вembedding_dim =100
max_length =100
model = Sequential()
model.add(Embedding(n_vocab,embedding_dim,input_length=max_length)
)
model.add(LSTM(256))
model.add(Dropout(0.2))
model.add(Dense(y.shape[1], activation='sigmoid'))
model.compile(loss='categorical_crossentropy', optimizer='adam')
```

Виводимо статистику:

Лістинг 3.6 -

```
model.summary()
```

Запускаємо навчання:

Лістинг 3.7 - Запуск навчання.

```
model.fit(X, y, epochs = 5, batch_size=128)
```

Як видно, ми навчаємо нейромережу на вхідних даних і використовуємо п'ять епох з розміром блоку 128.

Далі - зберігаємо дані навчання з метою економії часу для експериментів з нейромережею щоб відірвати використання нейромережі від набору навчальних даних.

Лістинг 3.8 - Збереження результатів навчання.

```
model.save(MODEL_LTSV1_NAME,overwrite=True)
```

Модель LSTM1 на основі слів загалом дуже погано передбачає такі слова, незалежно від того, видимі дані, чи ні. Прогнози можуть бути покращені шляхом навчання моделі на більшу кількість епох.

Переробимо мережу тим за допомогою того, що додамо ще один шар LSTM, при цьому додаючи метод Dropout, який застосовується до прихованих нейронів у тілі мережевої моделі. Dropout застосовується між двома прихованими шарами та між останнім прихованим шаром та вихідним шаром. Використовується коефіцієнт відсіву 20% і обмеження за вагою для цих шарів.

Лістинг 3.9 - Побудова моделі.

```
model1 = Sequential()
model1.add(Embedding(n_vocab,embedding_dim,input_length=max_length
))
model1.add(LSTM(256,input_shape=(X.shape[1],embedding_dim),
return_sequences=True))
model1.add(Dropout(0.2))
model1.add(LSTM(256))
model1.add(Dropout(0.2))
model1.add(Dense(y.shape[1], activation='sigmoid'))
model1.compile(loss='categorical_crossentropy', optimizer='adam')
```

Аналогічно зробимо з моделлю LTSM2. Результати навчання зберігаються на диск.

Тепер переходимо до використання моделі

Порахуємо відсоток наявності англійських слів у тексті та відсоток слів не розпізнаних як англійські. Напишемо функцію за якою визначатимемо англійське чи не англійське слово. Створимо новий проект CheckTextLTSM, який завантажить модель з файлу та текст, що розпізнається.

Лістинг 3.10 - Функція обчислення perplexity.

```
def test_word(word):
    return word_perplexity(word) < TRESHOLD
```

Ця функція обчислює perplexity для слова і порівнює з константою TRESHOLD. Експериментально підібрано значення TRESHOLD = 0.35.

Для того щоб обчислити перплексію, сформуємо вектори p і q як модельне передбачення максимально очікуваної нормалізованої ймовірності символу і фактичної ймовірності відповідно:

Лістинг 3.11 - Формування векторів p і q.

```
  dataQ=[] #поточна ймовірність літери у слові
    dataP=[] #передбачена ймовірність літери у слові     for i in range(1, len(word)):
        dataQ.append(  probability_word(word, i))
        dataP.append(  predict_word(word, i))
```

де функції probability_word та predict_word визначені як:

```
def predict_word(word,nchar): #передбачає ймовірність наступної літери № nchar
    dataX = []
    n=len(word)
    if n<nchar:
        nchar=n
    if nchar<=0: nchar=1
    seq_in=word[0:nchar]
    dataX=[char_to_int[char] for char in seq_in]
    #dataX.append(lst)
    return predict_probabily_next_char(dataX)
```

```
def probability_word(word,nchar):
    dataX = []
    n=len(word)
    if n<nchar:
        nchar=n
    if nchar<=0: nchar=1
    seq_in=word[0:nchar]
    dataX=[char_to_int[char] for char in seq_in]
    #dataX.append(lst)
    probabilities=prediction_next_char(dataX)
    thecharindex=   char_to_int[word[nchar]]
    return probabilities[  thecharindex ]
```

Лістинг 3.12 - Програма перевірки тексту.

```
import numpy as np
…….....................................................
…….....................................................
…….....................................................

TRESHOLD = 0.35
MODEL_LTSV1_NAME = '../content/model/AliceLTSV1.model'
MODEL_LTSV2_NAME = '../content/model/AliceLTSV2.model'

string.punctuation    =    string.punctuation    +'‘'+'’'+'-
'+'''+'''+'—'
string.punctuation = string.punctuation.replace('.', '')
modelname = MODEL_LTSV2_NAME
tokensfilename = '../content/tokenizer.pickle'
#textname='../content/sample_data/Dickens.txt'
probe_textname = '../content/sample_data/Dickens.txt'

model = keras.models.load_model(modelname)
```

```
    chars = [' ', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i',
'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u',
             'v', 'w', 'x', 'y', 'z', 'ù', '—', ''', ''', '"',
'"', '.']
    char_to_int = dict((c, i) for i, c in enumerate(chars))
    int_to_char = dict((i, c) for i, c in enumerate(chars))
    ……..................................................
    ……..................................................
    ……..................................................
    ……..................................................

    def test_text(raw_text):
        file_nl_removed = ""
        for line in raw_text:
            line_nl_removed = line.replace("\n", " ")
            # removes newlines
            file_nl_removed += line_nl_removed

        file_p = "".join([char for char in file_nl_removed if
char not in string.punctuation])
        preprocessed_text = file_p.lower()
        words = preprocessed_text.split() # розбиваємо текст на
слова
        positive = 0
        negative = 0
        N=0
        nwords =round( len(words) *0.02)
        print()
        print()

        sumperp=0.0
        for i in range(nwords) :
            word=words[i]
            N+=1
            #if N%10==0:
```

```
            stdout.write("\r%5.2f" % (100.0*N/nwords))
            stdout.flush()
      #       sumperp+=word_perplexity(word)
            if    test_word(word):    #підраховуємо    кількість
розпізнаних і нерозпізнаних слів
             positive+=1
            else:
             negative+=1
        sumperp=sumperp/N
        print()
        print('всего слов ',N)
    print("розпізнано слів", positive * 100 / N, "% не розпізнано
слів", negative * 100 / N, "% ")
    #   print('<perplexity>= ',sumperp)
        return



    def test_corrupt_text(raw_text):
    #випадковим чином псуємо кожне слово
        file_nl_removed = ""
        for line in raw_text:
            line_nl_removed = line.replace("\n", " ")
            # removes newlines
            file_nl_removed += line_nl_removed
        file_p  =  "".join([char  for  char  in  file_nl_removed  if
char not in string.punctuation])
        preprocessed_text = file_p.lower()
        words = preprocessed_text.split()
        positive = 0
        negative = 0
        N=0
        nwords =round( len(words) *0.02)
        print()
        print()
        random.seed()
```

```
        for i in range(nwords) :
            word=words[i]
            for j in range(3):
             k = random.randrange(0,len(word))
             while True:
              c= int_to_char[random.randrange(1, n_vocab-1)]
              if c != word[k]: break
             word=word[:k]+c+word[k+1:]
            N+=1
            #if N%10==0:
            stdout.write("\r%5.2f" % (100.0*N/nwords))
           # print(100*N/nwords,' ',end='\r',flush=True)
            stdout.flush()
            if test_word(word):
             #print(word)
             positive+=1
            else:
             negative+=1
             #print(word)
        print()
        print('всего слів ',N)
        print("розпізнано слів ", positive * 100 / N,"% не
розпізнано слів ", negative * 100 / N, "% ")
        return
    # читаємо текст з файлу  з іменем probe_textname в форматі
'utf8'
    raw_text= open(probe_textname, encoding='utf8').read()
    print(probe_textname)
    test_text(raw_text) ) #перевіримо слова неспотвореного тексту
та надрукуємо статистику

    # спотворимо текст
    print()
    print('test corrupted words')
    test_corrupt_text(raw_text)
```

```
#перевіримо слова спотвореного тексту та надрукуємо статистику.
```

Коротко результати експериментів:

Використовувана модель навчена за уривком тексту “Alice in Wounderlan” на зразку з “Oliver Twist Or The Parish Boy’s Progress”(Charles Dickens).

Лістинг 3.14 - Екперимент 1, TRESHOLD=0.2.

```
2023-05-02 14:58:44.761727: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations:  AVX AVX2

To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

2023-05-02 14:58:45.317591: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1510] Created device /job:localhost/replica:0/task:0/device:GPU:0 with 2136 MB memory:  -> device: 0, name: NVIDIA GeForce GTX 1650, pci bus id: 0000:09:00.0, compute capability: 7.5

../content/sample_data/Dickens.txt

 0.032023-05-02 14:58:47.874499: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR Optimization Passes are enabled (registered 2)

WARNING:tensorflow:Model was constructed with shape (None, 100) for input KerasTensor(type_spec=TensorSpec(shape=(None, 100), dtype=tf.float32, name=’embedding_1_input’), name=’embedding_1_input’, description="created by layer ’embedding_1_input’"), but it was called on an input with incompatible shape (None, 1).
```

```
2023-05-02 14:58:48.835735: I tensorflow/stream_executor/cuda/cuda_dnn.cc:369] Loaded cuDNN version 8600

WARNING:tensorflow:Model was constructed with shape (None, 100) for input KerasTensor(type_spec=TensorSpec(shape=(None, 100), dtype=tf.float32, name='embedding_1_input'), name='embedding_1_input', description="created by layer 'embedding_1_input'"), but it was called on an input with incompatible shape (None, 2).

100.00

усього слів 3142

розпізнано слів 74.92043284532146 % не розпізнано слів 25.07956715467855 %

test corruped+ words


100.00

усього слів  3142

розпізнано слів  40.706556333545514 % не розпізнано слів 59.293443666454486 %

Process finished with exit code 0
```

Лістинг 3.15 - Екперимент 2, TRESHOLD=0.35.

```
2023-05-02 15:29:32.849153: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations:  AVX AVX2

To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
```

2023-05-02 15:29:33.422133: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1510] Created device /job:localhost/replica:0/task:0/device:GPU:0 with 2136 MB memory: -> device: 0, name: NVIDIA GeForce GTX 1650, pci bus id: 0000:09:00.0, compute capability: 7.5

../content/sample_data/Dickens.txt

0.032023-05-02 15:29:35.989077: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR Optimization Passes are enabled (registered 2)

WARNING:tensorflow:Model was constructed with shape (None, 100) for input KerasTensor(type_spec=TensorSpec(shape=(None, 100), dtype=tf.float32, name='embedding_1_input'), name='embedding_1_input', description="created by layer 'embedding_1_input'"), but it was called on an input with incompatible shape (None, 1).

2023-05-02 15:29:36.936132: I tensorflow/stream_executor/cuda/cuda_dnn.cc:369] Loaded cuDNN version 8600

WARNING:tensorflow:Model was constructed with shape (None, 100) for input KerasTensor(type_spec=TensorSpec(shape=(None, 100), dtype=tf.float32, name='embedding_1_input'), name='embedding_1_input', description="created by layer 'embedding_1_input'"), but it was called on an input with incompatible shape (None, 2).

100.00

усього слів 3142

розпізнано слів 89.40165499681731 % не розпізнано слів 10.598345003182686 %

test corrupted words

```
100.00

усього слів 3142
розпізнано слів 34.75493316359007 % не розпізнано слів
65.24506683640993 %

Process finished with exit code 0
```

Отже, видно що з TRESHOLD=0.35 класифікація слів краще. 65% неанглійських слів у разі спотвореного тексту пов’язано з тим, що з певною ймовірністю не вдається зіпсувати слово (наприклад, *to* може перетворитися на *do* або *ne*w на *mew*).

ВИСНОВКИ

За результатами виконання дипломної кваліфікаційної роботи було створено нейромережеву мовну модель на рівні символів, яка може служити доповненням до простих бібліотек видобування текстів із PDF, та покращить якість видобутого із PDF тексту, одночасно забезпечуючи можливість швидкої обробки великої кількості документів.

Під час виконання дипломної кваліфікаційної роботи було виконано основні задачі, які було сформовано на початку. Вони включають у себе: пошук визначень основних термінів; огляд технологій, які мають змогу створити мовну модель символьного рівня; застосування LSTM для створення мовної моделі символьного рівня; реалізація обчислення perplexity для виявлення незвичайних послідовностей; перевірка якості отриманого рішення.

Один з найважливіших задач будь-якої моделі машинного навчання - це оцінка її якості. У нашому випадку був використаний такий метод: ми по декілька разів змінювали випадковий символ на випадкову позицію. Оцінка якості моделі дорівнює 74-89%.

Розроблений продукт повністю виконує поставлені в роботі задачі. Завдання дипломної кваліфікаційної роботи в результаті повністю виконано.

ПЕРЕЛІК ПОСИЛАНЬ

1.Thoma M. Add table extraction benchmark. URL: https://github.com/py-pdf/benchmarks

2. Сайт “Deepset” URL: https://www.deepset.ai/blog/what-is-a-language-model

3. Bird S., Klein E., and Loper E. “Natural Language Processing with Python”. Printing History. June 2009. 479 pp.

4. Beardon C. and Lumsden D. Geoffrey Holmes. “Natural Language and Computational Linguistics An Introduction”. University of California. Ellis Horwood 1991. 232 pp.

5. Brownlee J. “Deep Learning for Natural Language Processing”. Machine Learning Mastery. 2017. 414 pp.

6. Russell S. and Norvig P. “Artificial Intelligence: A Modern Approach”. Fourth edition. Hoboken: Pearson, 2021. 1069 pp.

7. Christopher D. Schütze M. and Schütze H. “Foundations of Statistical Natural Language Processing”. The MIT Press; Fourth edition. June 18, 1999. 620 pp.

8. Indurkhya N. and Damerau F.J. “Machine Learning for Natural Language Processing”. Chapman & Hall. February 23, 2010. 702 pp.

9. Jurafsky D. and. Martin J.H. “Speech and Language Processing” Third Edition draft. January 7, 2023. 628 pp.

10. Rao D. and McMahan B. “Natural Language Processing with PyTorch”. O’Reilly Media, 1st edition. February 19, 2019. 254 pp.

11. Sarkar D. “Text Analytics with Python”. Bangalore, Karnataka, India. 2016. 385 pp.

12. Перплексія розподілення ймовірностей. URL: https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%BF%D0%BB%D0%B5%D0%BA%D1%81%D0%B8%D1%8F

13. Hochreiter S. and Schmidhuber Jü. Long short-term memory (англ.). Neural Computation (англ.): journal. 1997. Vol. 9, no. 8. URL: https://en.wikipedia.org/wiki/Long_short-term_memory

14. Huang Z., Wei Xu, Kai Yu. LSTM-CRF Models for Sequence Tagging. 10 pp.URL: https://arxiv.org/pdf/1508.01991.pdf

ДОДАТОК А

Програма навчання моделі

Лістинг А.1 - Програма навчання моделі.

```
# навчання моделей
import numpy
import re
import pandas as pd
import numpy as np
import keras
import string
import nltk
import pickle
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import LSTM
from keras.layers import Embedding
from keras.callbacks import ModelCheckpoint
from keras.utils import np_utils
#from sklearn.model_selection import train_test_split
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
string.punctuation = string.punctuation +'‘'+'’'+'-
'+'''+'''+'—'
string.punctuation = string.punctuation.replace('.', '')
#
MODEL_BIDIRECT_NAME='../content/model/DickensBidirectional.model'
# MODEL_LTSV1_NAME='../content/model/DickensLTSV1.model'
# MODEL_LTSV2_NAME='../content/model/DickensLTSV2.model'
# TEXT_FILE_NAME = '../content/sample_data/Dickens.txt'
# TOKEN_FILE_NAME='../content/Dickenstokenizer.pickle'

MODEL_BIDIRECT_NAME='../content/model/AliceBidirectional.mode
l'
```

```
    MODEL_LTSV1_NAME='../content/model/AliceLTSV1.model'
    MODEL_LTSV2_NAME='../content/model/ALiceLTSV2.model'
    TEXT_FILE_NAME = '../content/sample_data/corpus.txt'
    TOKEN_FILE_NAME='../content/AliceTokens.pickle'
    #Loads the data and preprocesses data and stores corpus in
raw_text

    # Читаємо текст та видаляємо знаки пунктуації
    raw_text = open(TEXT_FILE_NAME, encoding = 'utf8').read()

    file_nl_removed = ""
    for line in raw_text:
      line_nl_removed = line.replace("\n", " ")
    #removes newlines
      file_nl_removed += line_nl_removed

    file_p = "".join([char for char in file_nl_removed if char
not in string.punctuation])
    #removes all special characters
    sents = nltk.sent_tokenize(file_p)
    print("The number of sentences is", len(sents))
    #prints the number of sentences

    string.punctuation = string.punctuation + '.'
    file_q = "".join([char for char in file_p if char not in
string.punctuation])   #removes even periods.
    words = nltk.word_tokenize(file_q)
    print("The number of tokens is", len(words))
    #prints the number of tokens

    average_tokens = round(len(words)/len(sents))
    print("The average number of tokens per sentence is",
average_tokens)
    #prints the average number of tokens per sentence

    unique_tokens = set(words)
```

```
    print("The number of unique tokens are", len(unique_tokens))
    #prints the number of unique tokens

    preprocessed_text = file_p.lower()
    #converts corpus into lowercase

    # Hyperparameters of the model
    vocab_size = 2750  #chosen based on statistics of the model
    oov_tok = '<OOV>'
    embedding_dim = 100
    padding_type='post'
    trunc_type='post'

    # tokenizes sentences
    tokenizer = Tokenizer(num_words = vocab_size,
oov_token=oov_tok)
    tokenizer.fit_on_texts([preprocessed_text])
    # savingtokenizer
    with open(TOKEN_FILE_NAME, 'wb') as handle:
        pickle.dump(tokenizer, handle,
protocol=pickle.HIGHEST_PROTOCOL)

    word_index = tokenizer.word_index
    seq_length = 50
    tokens = tokenizer.texts_to_sequences([preprocessed_text])[0]

    #fill model data


    dataX = []
    dataY = []

    for i in range(0, len(tokens) - seq_length - 1, 1):
      seq_in = tokens[i:i + seq_length]
      seq_out = tokens[i + seq_length]
```

```
        if seq_out == 1:  # Skip samples where target word is OOV
          continue

        dataX.append(seq_in)
        dataY.append(seq_out)

    N = len(dataX)
    print("Total training data size is -", N)
    X = numpy.array(dataX)

    # one hot encodes the output variable
    #y = numpy.array(dataY)
    y = np_utils.to_categorical(dataY)
    # Bidirectional LSTM
    # with embedding
    model = keras.Sequential([
        keras.layers.Embedding(vocab_size,          embedding_dim,
input_length=seq_length),
        keras.layers.Bidirectional(keras.layers.LSTM(64)),
        keras.layers.Dense(vocab_size, activation='sigmoid')
    ])

    # compiles model
    model.compile(loss='categorical_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])

    # model summary
    model.summary()
    num_epochs = 5
    history = model.fit(X, y, epochs=num_epochs, batch_size =
128, verbose=1, validation_split=0.2)
    model.save(MODEL_BIDIRECT_NAME,overwrite=True)

    # LTSV1
```

```
    raw_text = preprocessed_text   #periods have not been removed
for better results
    print("\n Model: LTSV1")
    # creates mapping of unique characters to integers
    # chars - набор символов который содержит текст
    chars = [' ', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i',
'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v',
'w', 'x', 'y', 'z', 'ù', '—', ''', ''', '"', '"','.']
    char_to_int = dict((c, i) for i, c in enumerate(chars))
    int_to_char = dict((i, c) for i, c in enumerate(chars))
    # Prints the total characters and character vocab size
    n_chars = len(raw_text)
    n_vocab = len(chars)

    print("The number of total characters are", n_chars)
    print("\nThe character vocab size is", n_vocab)
    #Prepares dataset where the input is sequence of 100
characters and target is next character.
    seq_length = 100

    dataX = []
    dataY = []
    #Заповнюємо дані для навчання
    for i in range(0, n_chars - seq_length, 1):
      seq_in = raw_text[i:i + seq_length]
      seq_out = raw_text[i + seq_length]
      dataX.append([char_to_int[char] for char in seq_in])
      dataY.append(char_to_int[seq_out])

    n_patterns = len(dataX)
    print ("Total Patterns: ", n_patterns)
    # reshapes X to be [samples, time steps, features]
    X = numpy.reshape(dataX, (n_patterns, seq_length, 1))

    # one hot encodes the output variable
    y = np_utils.to_categorical(dataY)
```

```
    embedding_dim =100
    max_length =100
    model = Sequential()
    model.add(Embedding(n_vocab,                    embedding_dim,
input_length=max_length))
    model.add(LSTM(256))
    model.add(Dropout(0.2))
    model.add(Dense(y.shape[1], activation='sigmoid'))
    model.compile(loss='categorical_crossentropy',
optimizer='adam')
    model.summary()

    model.fit(X, y, epochs = 5, batch_size=128)
    model.save(MODEL_LTSV1_NAME,overwrite=True)

    print("\n Model: LTSV2")
    model1 = Sequential()
    model1.add(Embedding(n_vocab,                   embedding_dim,
input_length=max_length))
    model1.add(LSTM(256, input_shape=(X.shape[1], embedding_dim),
return_sequences=True))
    model1.add(Dropout(0.2))
    model1.add(LSTM(256))
    model1.add(Dropout(0.2))
    model1.add(Dense(y.shape[1], activation='sigmoid'))
    model1.compile(loss='categorical_crossentropy',
optimizer='adam')
    model1.summary()

    model1.fit(X, y, epochs=20, batch_size=64)
    model.save(MODEL_LTSV2_NAME,overwrite=True)
```

ДОДАТОК Б

Програма аналізу текстового файла

Лістинг Б.1 - програма аналізу текстового файла.

```
import random
from sys import stdout

import numpy as np
import keras
import string
#import nltk
from clang.cindex import xrange
import pickle

TRESHOLD = 0.35  # 0.1663747994450379
# MODEL_LTSV1_NAME = '../content/model/DickensLTSV1.model'
# MODEL_LTSV2_NAME = '../content/model/DickensLTSV2.model'


MODEL_LTSV1_NAME = '../content/model/AliceLTSV1.model'
MODEL_LTSV2_NAME = '../content/model/AliceLTSV2.model'

string.punctuation    =    string.punctuation    +'‘'+'’'+'-
'+'''+'''+'—'
string.punctuation = string.punctuation.replace('.', '')
#textname  = '../content/sample_data/corpus.txt'
#modelname = '../content/model/LTSV2.model'
modelname = MODEL_LTSV2_NAME
tokensfilename = '../content/tokenizer.pickle'
#textname='../content/sample_data/Dickens.txt'
#probe_textname='../content/sample_data/corpus.txt'
probe_textname = '../content/sample_data/Dickens.txt'

model = keras.models.load_model(modelname)
tokenizer=''
```

```
    chars = [' ', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i',
'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v',
'w', 'x', 'y', 'z', 'ù', '—', ''', ''', '‟', '"', '.']
    char_to_int = dict((c, i) for i, c in enumerate(chars))
    int_to_char = dict((i, c) for i, c in enumerate(chars))
    # Prints the total characters and character vocab size
    #n_chars = len(raw_text)
    n_vocab = len(chars)


    def LoadTokenizer(tokens_file):
     with open(tokens_file, 'rb') as handle:
        tokenizer = pickle.load(handle)


    #Predict probability of next char
    def predict_probabily_next_char(pattern):
          x = np.reshape(pattern, (1, len(pattern), 1))
          prediction = model.predict(x, verbose=0)
    #        print (int_to_char[np.argmax(prediction)], end = '')
#get next char index.
          return  prediction[0][np.argmax(prediction)]

    #Get probability of the real char
    def prediction_next_char(pattern):
        x = np.reshape(pattern, (1, len(pattern), 1))
        prediction = model.predict(x, verbose=0)
        #        print (int_to_char[np.argmax(prediction)], end =
'')   #get next char index.
        return prediction[0]

    def   predict_word(word,nchar):  #предсказывает  вероятность
следующей буквы № nchar
        dataX = []
        n=len(word)
```

```
    if n<nchar:
        nchar=n
    if nchar<=0: nchar=1
    seq_in=word[0:nchar]
    dataX=[char_to_int[char] for char in seq_in]
    #dataX.append(lst)
    return predict_probabily_next_char(dataX)

def probability_word(word,nchar):
    dataX = []
    n=len(word)
    if n<nchar:
        nchar=n
    if nchar<=0: nchar=1
    seq_in=word[0:nchar]
    dataX=[char_to_int[char] for char in seq_in]
    #dataX.append(lst)
    probabilities=prediction_next_char(dataX)
    thecharindex=   char_to_int[word[nchar]]
    return probabilities[  thecharindex ]

def DKL(p,q): # расходження Кульбака — Лейблера
    s=0.0
    numlen=len(p)
    for i in range(numlen):
        s = s + p[i]*(np.log(p[i])-np.log(q[i]) )
    return s

def compute_perplexity(dataX,dataY):

    return abs(  DKL(dataX,dataY))

def word_perplexity(word):
    word=word.lower()
  #  dataX = [char_to_int[char] for char in word]
    dataQ=[] #поточна ймовірність букви у слові
```

```
        dataP=[] #передбачена ймовірність букви у слові
       # print(word)
        for i in range(1, len(word)):
            dataQ.append(  probability_word(word, i))
            dataP.append(  predict_word(word, i))
        sumQ=0.0
        sumP=0.0
        N=0
        for i in range(1, len(dataQ)):
           sumP+=dataP[i]
           sumQ+=dataQ[i]
           N=N+1
        for i in range(1, len(dataQ)):
            dataP[i]=dataP[i]/sumP
            dataQ[i]=dataQ[i]/sumQ

    #    for i in range(1, len(word)):
    #                print(probability_word(word,  i),     ’  ’,
predict_word(word, i))
        return compute_perplexity(dataP,dataQ)

    #повертає true, якщо слово распізнано як англійське
    def test_word(word):
        return word_perplexity(word) < TRESHOLD

    def test_text(raw_text):
        file_nl_removed = ""
        for line in raw_text:
            line_nl_removed = line.replace("\n", " ")
            # removes newlines
            file_nl_removed += line_nl_removed

        file_p = "".join([char for char in file_nl_removed if
char not in string.punctuation])
        preprocessed_text = file_p.lower()
```

```
        words = preprocessed_text.split() #розбиваємо текст на
слова
        positive = 0
        negative = 0
        N=0
        nwords =round( len(words) *0.02)
        print()
        print()

        for i in range(nwords) :
            word=words[i]
            N+=1
            stdout.write("\r%5.2f" % (100.0*N/nwords))
            stdout.flush()
            if test_word(word): #рахуємо кількість розпізнаних и
нерозпізнаних слів
             positive+=1
            else:
             negative+=1
        print()
        print('усього слів ',N)
        print("розпізнано слів ", positive * 100 / N,"% не
розпізнано слів ", negative * 100 / N, "% ")
        return


    def test_corrupt_text(raw_text):
        #випадковим чином псуємо кожне слово
        file_nl_removed = ""
        for line in raw_text:
            line_nl_removed = line.replace("\n", " ")
            # removes newlines
            file_nl_removed += line_nl_removed

        file_p = "".join([char for char in file_nl_removed if
char not in string.punctuation])
```

```
        preprocessed_text = file_p.lower()
        words = preprocessed_text.split()
        positive = 0
        negative = 0
        N=0
        nwords =round( len(words) *0.02)
        print()
        print()
        random.seed()
        for i in range(nwords) :
            word=words[i]
            for j in range(3):
             k = random.randrange(0,len(word))
             while True:
              c= int_to_char[random.randrange(1, n_vocab-1)]
              if c != word[k]: break
             word=word[:k]+c+word[k+1:]
            N+=1
            stdout.write("\r%5.2f" % (100.0*N/nwords))
            stdout.flush()
            if test_word(word):
             positive+=1
            else:
             negative+=1
        print()
        print('усього слів ',N)
        print("розпізнано слів ", positive * 100 / N,"% не
розпізнано слів ", negative * 100 / N, "% ")
        return


    raw_text= open(probe_textname, encoding='utf8').read()
    print(probe_textname)
    test_text(raw_text)

    # спотворимо текст
```

```
print()
print('test corrupted words')
test_corrupt_text(raw_text)
```