Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки

«Затверджую»
в.о. завідуючого кафедри
комп'ютерних систем та робототехніки
_____ к. ф.-м. н., доцент Максим Хруслов
«___» грудня 2024 р.

Пояснювальна записка

до кваліфікаційної роботи
магістра

на тему: «**МОДЕЛЬ АВТОМАТИЗОВАНОГО ПЕРЕКЛАДУ ТЕКСТІВ НА ОСНОВІ АЛГОРИТМІВ МАШИННОГО НАВЧАННЯ**»

Спеціальність 174 – Автоматизація, комп'ютерно-інтегровані технології та робототехніка
Галузь знань 17 – Електроніка, автоматизація та електронні комунікації.
Освітня програма «Комп'ютеризовані системи управління та автоматика»

Захищено на засіданні
Екзаменаційної комісії № 44
протокол № __ від __.12.2024 р.
Оцінка _____ / _____
Голова Екзаменаційної комісії
_____ **СКОБ Ю. О.**

Виконав:
Студент групи КУ– 61
ШАРАПОВ Ігор Миколайович ____

Керівник: доцент кафедри
комп'ютерних систем та робототехніки,
канд. техн. наук
СТРІЛЕЦЬ Вікторія Євгенівна ____

Рецензент: канд. техн. наук, доцент, в.о.
зав. кафедри теоретичної та прикладної
інформатики
МЕНЯЙЛОВ Євген Сергійович _____

Харків – 2024

АНОТАЦІЯ

Пояснювальна записка до магістерської атестаційної роботи складається зі вступу, трьох розділів, висновків, списку використаних джерел і трьох додатків. Загальний обсяг роботи складає 85 сторінок, із яких 66 сторінок основної частини з 18 рисунками, 0 таблиць, списку використаних джерел із 20 найменувань та чотирма додатками.

Метою кваліфікаційної роботи є забезпечення високої якості перекладу текстових даних через створення моделі автоматизованого перекладу на основі алгоритмів машинного навчання.

Об'єкт дослідження – процес автоматизованого перекладу текстів з однієї мови на іншу.

Предмет дослідження – моделі і методи машинного перекладу.

Проблема, яка вирішується в кваліфікаційній роботі полягає в тому, щоб розробити модель автоматизованого перекладу текстів на основі алгоритмів машинного навчання, яка забезпечить високу точність, адаптивність та ефективність перекладу. Дослідження спрямоване на аналіз існуючих методів, розробку моделі обробки природної мови та створення системи, що здатна здійснювати переклад з англійської на українську мову.

Область застосування: розроблена модель автоматизованого перекладу текстів на основі алгоритмів машинного навчання може бути застосована в різних сферах, таких як переклад офіційних та технічних документів, підтримка міжнародної комунікації в бізнесі, автоматизація перекладу навчальних матеріалів у освіті.

Ключові слова: машинний переклад, нейронні мережі, статистичний переклад, автоматизований переклад, обробка природної мови.

ABSTRACT

An explanatory note to the master's thesis consists of an introduction, three sections, conclusions, a list of sources used and three appendices. The total volume of the work is 85 pages, of which 66 pages are the main part with 18 figures, 0 tables, a list of sources used with 20 names and three appendices.

The purpose of the qualification work is to ensure high quality of translation of text data by creating an automated translation model based on machine learning algorithms.

The object of the research is the process of automated translation of texts from one language to another.

The subject of the research is models and methods of machine translation.

The problem solved in the qualification work is to develop a model of automated translation of texts based on machine learning algorithms, which will ensure high accuracy, adaptability and efficiency of translation. The research is aimed at analyzing existing methods, developing a natural language processing model and creating a system capable of translating from English into Ukrainian.

Scope: the developed model of automated text translation based on machine learning algorithms can be applied in various areas, such as translation of official and technical documents, support of international communication in business, automation of translation of educational materials in education, ensuring accessibility of multimedia content through subtitle translation, integration into social networks and communication applications for instant translation, as well as in content management systems for automating translation of websites and scientific research, which makes this model an important tool in a globalized world.

Keywords: machine translation, neural networks, statistical translation, automated translation, natural language processing.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ І УМОВНИХ ПОЗНАЧЕНЬ

МП – Машинний переклад

RBMT – Rule-based Machine Translation

SMT – Statistical Machine Translation

NMT – Neural Machine Translation

TBMT– Transfer-based machine translation

MT – machine translation

EBMT – Example-Based Machine Translation

CNN – Convolutional Neural Network

RNN – Recurrent Neural Network

BPTT – Backpropagation Through Time

LSTM – Long Short-Term Memory

GRU – Gated Recurrent Unit

NLP – Natural Language Processing

BLEU – Bilingual Evaluation Understudy

METEOR – Metric for Evaluation of Translation with Explicit ORdering

GPU – Graphics Processing Unit

TPU – Tensor Processing Unit

GPT – Generative Pre-trained Transformer

BERT – Bidirectional Encoder Representations from Transformers

BPE – Byte-Pair Encoding

BBU – bert-base-uncased

BPE – Byte-Pair Encoding

CBOW –: Continuous Bag of Words

BoW – Bag-of-Words

GloVe – Global Vectors for Word Representation

ELMo – Embeddings from Language Models

SGD – Stochastic Gradient Descent

Adam – Adaptive Moment Estimation

RMSprop – Root Mean Square Propagation

Adagrad – Adaptive Gradient Algorithm

Nadam – Nesterov-accelerated Adaptive Moment Estimation

MSE – Mean Squared Error Loss

NLTK – Natural Language Toolkit

ВСТУП

З розвитком технологій та глобалізацією, потреба в ефективному та точному перекладі текстів зростає з кожним днем. Автоматизований переклад, що базується на алгоритмах машинного навчання та нейронних мережах, відкриває нові можливості для подолання мовних бар'єрів та покращення міжнародної комунікації.

Актуальність даного дослідження зумовлена зростаючими потребами в автоматизованому перекладі текстів у глобалізованому світі, де ефективна комунікація між різними мовами стає дедалі важливішою. Використання алгоритмів машинного навчання та нейронних мереж відкриває нові можливості для підвищення якості перекладу, що критично важливо для бізнесу, освіти, науки та міжнародного співробітництва. Таким чином, дослідження є необхідним для задоволення сучасних потреб суспільства..

Мета дослідження – забезпечити високу якість перекладу текстових даних через створення моделі автоматизованого перекладу на основі алгоритмів машинного навчання. Дослідження спрямоване на аналіз існуючих методів обробки природної мови, розробку моделі автоматизованого перекладу на основі нейронних мереж, яка здатна з високою точністю здійснювати переклад текстів.

Задачі дослідження включають аналіз існуючих методів машинного навчання для автоматизованого перекладу текстів, вивчення специфіки перекладу з урахуванням контексту, розробку моделі автоматизованого перекладу на основі нейронних мереж, програмну реалізацію цієї моделі, а також формулювання рекомендацій щодо практичного використання розробленої моделі.

Об'єкт дослідження – процес автоматизованого перекладу текстів, що здійснюється за допомогою алгоритмів машинного навчання.

Предмет дослідження – моделі, алгоритми та методи машинного навчання, які використовуються для автоматизації процесу перекладу текстів, а також методи підготовки даних і навчання моделей.

Задачі дослідження:

- аналіз існуючих підходів до автоматизованого перекладу;
- розробка архітектури нейронної мережі для перекладу;
- навчання моделі на великому корпусі паралельних текстів;
- оцінка якості перекладу за допомогою метрик.

Очікувані результати цього дослідження можуть знайти широке застосування в різних сферах, таких як бізнес, наука, освіта та туризм. Розроблена модель може бути інтегрована в різноманітні програмні продукти та сервіси, що значно спростить процес перекладу текстів і підвищить його ефективність.

РОЗДІЛ 1.
МАШИННИЙ ПЕРЕКЛАД ТЕКСТОВИХ ДАНИХ

1.1 Задача машинного (автоматизованого) перекладу

Машинний переклад (МП), також відомий як автоматизований переклад, - це процес використання комп'ютерних програм для перекладу тексту або мовлення з однієї природної мови на іншу. Суть машинного перекладу полягає в автоматизації процесу перекладу, який традиційно виконується людиною-перекладачем.

Основні цілі машинного перекладу:

- зменшення витрат часу та зусиль на переклад, особливо для великих обсягів тексту;
- подолання мовних бар'єрів та забезпечення доступу до інформації для ширшої аудиторії;
- сприяння міжкультурній комунікації та співпраці.

Важливо розуміти, що машинний переклад не є простою заміною слів. Ефективний машинний переклад потребує розуміння граматики, синтаксису, семантики та контексту обох мов. Сучасні системи машинного перекладу використовують складні алгоритми та великі обсяги даних для досягнення високої точності та природності перекладу [6].

Історія машинного перекладу налічує вже понад півстоліття і може бути поділена на кілька основних етапів:

- 1950-ті - 1960-ті роки: Ранні дослідження та "правильний" підхід. Перші спроби машинного перекладу базувалися на правилах та словниках. Цей підхід, відомий як правильний машинний переклад (RBMT), мав обмежений успіх через складність формалізації всіх мовних правил [1].
- 1970-ті - 1990-ті роки: Статистичний підхід. З розвитком обчислювальної техніки та накопиченням великих мовних корпусів став

популярним статистичний машинний переклад (SMT). Цей підхід базується на аналізі статистичних закономірностей у паралельних текстах [8].

- 2010-ті роки – дотепер: Нейронний підхід. З появою глибинного навчання нейронний машинний переклад (NMT) став домінуючим підходом. Нейронні мережі здатні навчатися складним мовним закономірностям та генерувати більш природні та точні переклади [11].

Ключові віхи в історії машинного перекладу:

- 1954 р.: Перша публічна демонстрація системи машинного перекладу (англійська-російська).
- 1990-ті: Розробка перших комерційних систем SMT.
- 2014 р.: Поява перших систем NMT, які перевершили SMT за якістю перекладу.
- Сьогодні: Швидкий розвиток NMT, вдосконалення архітектур нейронних мереж та використання величезних обсягів даних.[3]

Машинний переклад продовжує активно розвиватися, і в майбутньому ми можемо очікувати ще більш точних, швидких та універсальних систем перекладу.

Хоча машинний переклад значно просунувся, створення ідеальної системи все ще стикається з низкою труднощів. Однією з них є багатозначність, де слова мають різні значення залежно від контексту. Наприклад, "ключ" може бути і фізичним предметом, і розгадкою, і музичним символом. Машинам важко розрізняти ці нюанси, що призводить до неточностей.

Іншою проблемою є розбіжності в граматиці та синтаксисі між мовами. Порядок слів, способи узгодження, часи – все це може суттєво відрізнятися. Наприклад, фіксований порядок слів в англійській мові протиставляється гнучкішому в українській, що створює труднощі для машинного перекладу.

Культурні та контекстуальні особливості також ускладнюють завдання. Гумор, сарказм, ідіоми – все це потребує глибокого розуміння культури, що

важко запрограмувати. Навіть просте звертання "пан" чи "пані" не має прямого аналога в англійській мові, що демонструє глибину проблеми [7].

Крім того, існує проблема недостатніх даних для навчання, особливо для малопоширених мов. Обмежена кількість паралельних текстів ускладнює навчання систем машинного перекладу. Також виникають труднощі з перекладом спеціалізованих текстів, де потрібне знання термінології та специфіки галузі.

Нарешті, існують етичні аспекти, пов'язані з авторським правом, конфіденційністю та відповідальністю за помилки перекладу. Постійна еволюція мови також ставить виклики, адже системи потребують постійного оновлення, щоб враховувати нові слова та вирази.

Подолання цих перешкод є пріоритетом для розробників. Сучасні дослідження фокусуються на глибинному навчанні, механізмах уваги та інших підходах, спрямованих на створення більш точних, природних та адаптивних систем машинного перекладу.

1.2. Підходи до автоматизованого перекладу текстових даних

Серед основних методів машинного перекладу виділяють:

- заснований на правилах (Rule-based machine translation, RBMT);
- заснований на словниках (Dictionary-based);
- заснований на трансформації (Transfer-based machine translation, TBMT);
- заснований на міжмовному підході (Interlingual MT);
- заснований на статистичному підході;
- заснований на прикладах (Example-based);
- заснований на гібридному підході (Hybrid MT);
- заснований на нейронних мережах (Neural MT).

Ці підходи можуть застосовуватись поодинці (Single approach) або в комбінаціях (це і називається гібридний підходом).

В загальному еволюція різних підходів до машинного перекладу зображена на рис. 1.1 [3].

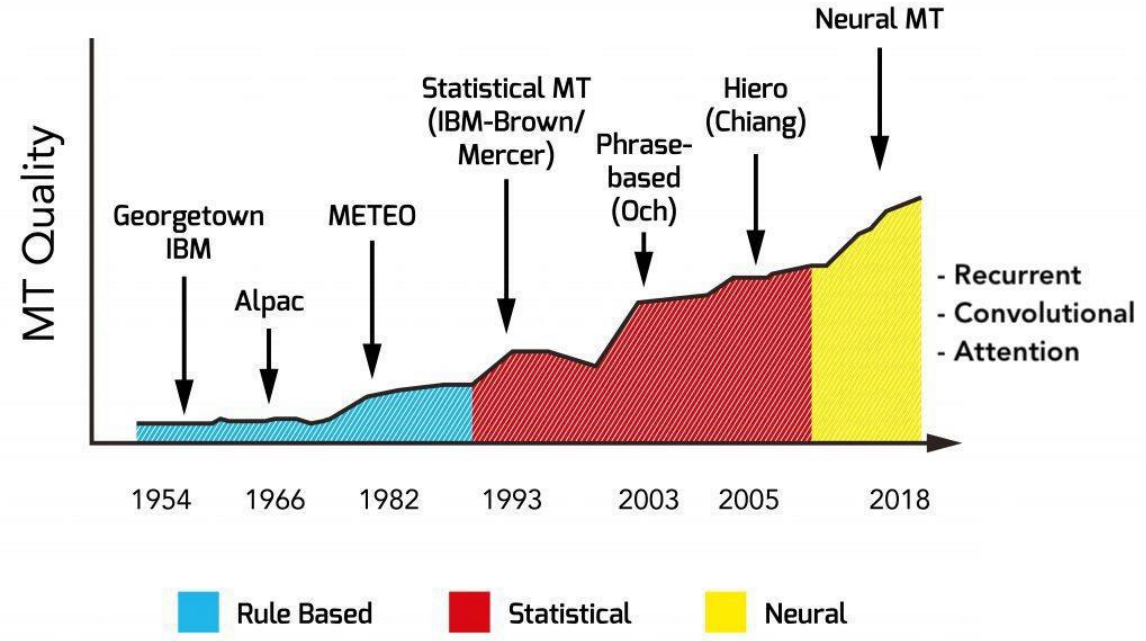

Рисунок 1.1 – Еволюція підходів до машинного перекладу

1.2.1. Машинний переклад, заснований на правилах

Цей метод включає в себе машинний переклад заснований на трансформації, міжмовному підході та словниках. Він заснований на зборі даних про структуру початкової та цільової мови, словників та правил, аналізу їх, та генерування перекладу на основі морфологічного, синтаксичного та семантичного аналізу як початкової, так і цільової мов (чи мов), що беруть участь у конкретному завданні перекладу.

Переваги:

1. Для роботи системи не потрібен корпус двомовних текстів.

2. Може бути перенесений на будь-яку систему, так як правило не залежить від її налаштувань.

3. При достатньо точному та детальному наборі правил (пристосованих до вимог часу) можна було б отримати ідеальну систему.

4. Може бути підкоректована.

5. Може бути перевикористана для інших цілей.

Недоліки:

1. Створення настільки достовірних словників, як необхідно для ідеальної системи є вартісним процесом (і з сторони фінансування і часових ресурсів).

2. Деякі аспекти системи все ж задаватимуться вручну.

3. Створення правила для фразеологізмів, двохзначних та інших проблемних з точки зору перекладу структур.

4. Внесення змін в систему може призвести до непередбачуваних наслідків

1.2.2. Машинний переклад, заснований на словниках

Цей метод працює за принципом прямої заміни слів з вихідної мови на слова з цільової мови, використовуючи лише словник. Він не враховує граматику, контекст, багатозначність слів, фразеологізми та інші важливі аспекти мови [12]

Переваги:

- Не потребує паралельних текстів для навчання.
- Універсальний та не залежить від налаштувань системи.
- Може бути ефективним для споріднених мов зі схожою структурою (наприклад, українська та російська) [4].

Недоліки:

- Створення якісних словників потребує значних ресурсів.
- Потребує ручного налаштування.
- Не враховує багатозначність, фразеологізми та інші нюанси мови, що призводить до неточних та неприродних перекладів.

Простими словами, цей метод перекладає кожне слово окремо, не розуміючи зв'язків між ними та контексту, в якому вони використовуються. Це

як шукати переклад кожного слова в паперовому словнику, не звертаючи уваги на речення загалом.

1.2.3. Машинний переклад, заснований на трансформації

Трансформаційний машинний переклад працює, частково спираючись на міжмовний підхід. Це означає, що система спочатку аналізує вхідний текст і створює його проміжне представлення, яке потім трансформується у цільову мову. При цьому враховуються особливості обох мов, але не настільки жорстко, як у правильному підході.

Сильні сторони:

- Не потребує великої кількості паралельних текстів.
- Більш універсальний, ніж правильний підхід, і може бути легше адаптований до нових мов.
- Не вимагає створення окремого міжмовного представлення для кожної пари мов.

Слабкі сторони:

- Все ще може допускати помилки при перекладі фразеологізмів, багатозначних слів та інших складних мовних конструкцій.
- Потребує певного доналаштування для кожної пари мов, хоча і менше, ніж правильний підхід.

Простими словами, цей метод намагається знайти баланс між універсальністю та точністю. Він не настільки жорсткий, як правильний підхід, але і не настільки гнучкий, як статистичний чи нейронний переклад.

1.2.4. Машинний переклад, заснований на міжмовному підході

Замість прямого перекладу з однієї мови на іншу, цей метод використовує проміжну, абстрактну мову, яку можна назвати "мовою-посередником".

1. Аналіз: Спочатку вхідний текст аналізується, і його зміст, структура та граматичні особливості перетворюються на цю універсальну "міжмовну" репрезентацію. Це як розкласти текст на універсальні будівельні блоки сенсу.

2. Генерація: На основі цієї абстрактної репрезентації система генерує переклад на цільову мову, використовуючи правила та словники для цієї мови.

Переваги:

- Універсальність: Система може перекладати між будь-якими мовами, для яких є правила перетворення в "міжмовну" репрезентацію. Це наче мати один універсальний ключ, який підходить до будь-яких дверей.
- Економія ресурсів: Замість створення окремих систем для кожної пари мов (українська-англійська, українська-німецька, англійська-німецька тощо), потрібно лише розробити правила для кожної мови та "міжмовної" репрезентації. Це значно спрощує розробку та підтримку системи.
- Гнучкість: Міжмовний підхід дозволяє враховувати різні способи вираження однієї й тієї ж думки (парафрази), що робить переклад більш природним та точним.
- Обробка різних мов: Цей метод особливо корисний для перекладу між мовами з різними структурами (наприклад, англійська та арабська), оскільки "міжмовна" репрезентація дозволяє абстрагуватися від конкретних граматичних особливостей кожної мови.

Недоліки:

- Складність створення "міжмовної" репрезентації: Розробка універсальної "мови-посередника", яка б адекватно відображала всі нюанси всіх мов світу, є надзвичайно складним завданням.
- Обмежена адаптивність: Системи, засновані на міжмовному підході, можуть бути менш ефективними при перекладі текстів з вузькоспеціалізованих областей (наприклад, медичних чи юридичних), оскільки "міжмовна" репрезентація може не враховувати специфічну термінологію та стилістику.

1.2.5. Машинний переклад, заснований на статистичному підході

Цей метод використовує величезні колекції текстів (корпуси) на двох мовах, які вже перекладені людьми. Аналізуючи ці тексти, система виявляє закономірності та статистичні зв'язки між словами та фразами в різних мовах. На основі цих даних створюються статистичні моделі, які потім використовуються для перекладу нових текстів.

Простіше кажучи: комп'ютер "вчиться" перекладати, аналізуючи велику кількість прикладів. Чим більше прикладів, тим точніший переклад.

Переваги:

– Ефективність: Цей підхід дозволяє ефективно використовувати великі обсяги даних та людські ресурси, які вже вкладені в переклад текстів.

– Доступність даних: Існує багато корпусів паралельних текстів для різних мов, що дозволяє створювати системи перекладу для багатьох мовних пар.

– Уникнення проблем: Статистичний підхід уникає багатьох проблем, характерних для методів, заснованих на правилах, таких як складність створення та підтримки правил, обмежена гнучкість та нездатність враховувати різноманітність мовних конструкцій.

Недоліки:

– Потреба в даних: Для кожної пари мов потрібні великі корпуси паралельних текстів, що може бути проблемою для рідкісних мов.

– Складність виправлення помилок: Конкретні помилки, допущені системою, важко передбачити та виправити, оскільки вони залежать від статистичних моделей, а не від чітких правил.

– Штучна "вільність": Іноді переклад може здаватися "вільним" та природним, але насправді це може маскувати проблеми з точністю та адекватністю передачі сенсу.

– Проблеми з різними мовами: Статистичний підхід може бути менш ефективним для мов з різним порядком слів, оскільки статистичні моделі можуть не враховувати ці відмінності.

1.2.6. Машинний переклад, заснований на прикладах

Метод, який працює як мозаїка: він шукає схожі фрагменти тексту в величезній базі даних перекладів та складає з них готовий переклад. Це і є ***машинний переклад, заснований на прикладах (Example-Based Machine Translation, EBMT).***

Як працює:

1. Пошук: Система аналізує вхідний текст і розбиває його на окремі фрагменти (слова, фрази, речення). Потім вона шукає схожі фрагменти в корпусі паралельних текстів, тобто в базі даних, де кожен фрагмент однією мовою має відповідник іншою мовою.[8]

2. Комбінування: Знайдені фрагменти-переклади комбінуються, адаптуються та з'єднуються разом, формуючи готовий переклад вхідного тексту.

Простіше кажучи: система "згадує" схожі фрази з перекладів, які вона вже бачила, і використовує їх для перекладу нового тексту.

Переваги:

• Точність у складних випадках: EBMT чудово справляється з перекладом фразеологізмів, фразових дієслів та інших мовних явищ, які важко перекласти за допомогою правил чи статистики.

• Ефективність для вузьких областей: Система добре працює з текстами з вузькоспеціалізованих областей (наприклад, медицина, юриспруденція), оскільки може використовувати корпус, який містить саме таку лексику.

• Переклад між різними мовами: EBMT особливо корисний для перекладу між мовами з різними структурами (наприклад, англійська та японська),

оскільки він не покладається на граматичні правила, а шукає прямі відповідності в корпусі.

Недоліки:

- Потреба у великому корпусі: Для ефективної роботи системи потрібен дуже великий корпус паралельних текстів, що може бути проблемою для деяких мовних пар.
- Проблеми з узагальненням: Якщо система була навчена на широкоспеціалізованому корпусі, вона може допускати помилки при перекладі вузькоспеціалізованих текстів, оскільки не зможе знайти достатньо схожих прикладів.

1.2.7. Машинний переклад заснований на гібридному підході

Об'єднання найкращих рис різних методів машинного перекладу в одній системі є ***гібридним підходом***.

Він поєднує в собі елементи різних методів, таких як:

- Статистичний: використовує статистичні моделі, побудовані на аналізі великих корпусів текстів.
- Заснований на правилах: застосовує лінгвістичні правила для аналізу та генерації тексту.
- Нейронний: використовує нейронні мережі для "навчання" перекладу на основі великих обсягів даних.

Різні типи гібридних систем:

- Модель на основі слова: поєднує статистичні дані про частоту появи слів з правилами їхнього поєднання.
- Модель на основі фрази: аналізує та перекладає цілі фрази, а не окремі слова.
- Модель на основі синтаксису: враховує граматичну структуру речень для більш точного перекладу.

- Модель на основі лісу: використовує "ліс" синтаксичних дерев для аналізу та генерації тексту.

З появою нейронних мереж гібридні системи стали ще потужнішими, поєднуючи переваги статистичного, заснованого на правилах та нейронного підходів.

Переваги:

- Синергія: Гібридні системи поєднують в собі сильні сторони різних методів, що дозволяє досягти кращих результатів, ніж при використанні кожного методу окремо. Наприклад, статистичний підхід може забезпечити плавність перекладу, а правила - граматичну правильність.
- Гнучкість: Гібридні системи можуть бути адаптовані до різних мовних пар та типів текстів, що робить їх універсальним інструментом перекладу.

Недоліки:

- Складність: Розробка та налаштування гібридних систем є складним завданням, оскільки потрібно враховувати особливості різних методів та їхню взаємодію.
- Ресурсоємність: Гібридні системи можуть потребувати значних обчислювальних ресурсів для обробки великих обсягів даних та виконання складних алгоритмів.
- Потенційні конфлікти: Поєднання різних методів може призвести до конфліктів між ними, що може негативно вплинути на якість перекладу. Наприклад, правила можуть суперечити статистичним даним, що призведе до неграматичних або неприродних конструкцій.

1.2.8. Машинний переклад, заснований на нейронних мережах

Система перекладу, яка не просто дотримується правил чи статистики, а "розуміє" текст, подібно до людського мозку. Це машинний переклад, заснований на нейронних мережах (Neural Machine Translation, NMT).

NMT використовує штучні нейронні мережі, які імітують роботу людського мозку. Ці мережі складаються з безлічі взаємопов'язаних вузлів (нейронів), які обробляють інформацію.

Система "навчається" перекладати, аналізуючи величезну кількість паралельних текстів. Під час навчання мережа самостійно виявляє складні закономірності та зв'язки між мовами, включаючи морфологічні, синтаксичні та семантичні аспекти.

Простіше кажучи: система не просто перекладає слова, а намагається "зрозуміти" їхнє значення та контекст, що дозволяє їй створювати більш природні та точні переклади.

Переваги:

- Ефективність: NMT потребує менше пам'яті порівняно з традиційними статистичними методами, що робить його більш ефективним для обробки великих обсягів даних.
- Узагальнення: Система може застосовувати отримані знання до нових текстів, навіть якщо вони відрізняються від тих, на яких вона навчалася.
- Цілісність: Усі частини моделі навчаються разом, що забезпечує узгодженість та підвищує ефективність перекладу.

Недоліки:

- Потреба у великих даних: Для досягнення високої точності NMT потребує величезних корпусів паралельних текстів, що може бути проблемою для деяких мовних пар.
- Тривале навчання: Процес навчання нейронної мережі може займати багато часу та потребувати значних обчислювальних ресурсів.
- Обмежена переносність: Систему потрібно навчати заново для кожної нової пари мов, що може бути трудомістким процесом.
- "Чорний ящик": Важко зрозуміти, як саме система приймає рішення, оскільки процеси, що відбуваються всередині нейронної мережі, є складними та не до кінця зрозумілими.

1.3 Підходи до нейронного машинного перекладу

Протягом останніх років спостерігається експоненційне зростання кількості, точності та якості моделей нейронного машинного перекладу. Це зростання зумовлене не лише розвитком академічних досліджень, але й більшою доступністю даних та потужних комп'ютерів. Найважливішими відкриттями останнього часу стали рекурентні нейронні мережі, довга-короткочасна пам'ять (підтип рекурентних мереж), механізм уваги та моделі трансформери.

1.3.1. Згорткові нейронні мережі

Основна концепція, яка лежить в основі CNN, полягає в тому, що ми обчислюємо вектор для кожної можливої комбінації слів. Наприклад, у рядку "my favorite neural network". Спочатку обчислюється векторне подання для: «my favorite», «favorite neural», «neural network», «my favorite neural», і так далі. Отримавши це, обчислюються всі вектори біграми, поки не дійдемо до верхнього вектора.

Обчислення цих біграм (або n-грамів у більш складних, але зручних архітектурах CNN) можна обчислити паралельно. Цей механізм успішно використовується для класифікування зображень. Наприклад, у зображенні кота: спочатку обробляються скупчення пікселів, потім розпізнаються фігури, потім частини зображення (вуха, ноги, хвіст тощо) і нарешті, чи на знімку є кішка.

1.3.2. Рекурентні нейронні мережі

Рекурентні нейронні мережі (RNN) були вперше представлені в 1980-х роках, але їх справжній потенціал розкрився лише в останні роки. RNN розширюють можливості традиційних (feed-forward) нейронних мереж, стаючи першим алгоритмом, який здатен «запам'ятовувати» свої попередні

стани. Це робить їх ідеальними для вирішення задач машинного навчання, що включають послідовні дані, такі як часові ряди, розмови, текст (включаючи переклад), фінансові дані, аудіо, відео, прогнози погоди та багато іншого [9].

На відміну від звичайних нейронних мереж, які приймають на вхід вектор фіксованого розміру, що обмежує їх використання в ситуаціях з вхідними послідовностями без попередньо визначеного розміру, RNN спеціально розроблені для обробки послідовностей даних без таких обмежень.

У нейронних мережах з прямим поширенням (feed-forward) інформація рухається лише в одному напрямку — від вхідного шару через приховані шари до вихідного шару. При цьому дані, представлені на поточному етапі, не враховують інформацію з попередніх моментів часу, що обмежує їх здатність до аналізу часових залежностей.

У RNN, натомість, інформація проходить через цикл, що дозволяє їй враховувати як поточні вхідні дані, так і дані, отримані раніше. Це означає, що RNN може обробляти вхідні дані в контексті їхнього розташування у часі, що є ключовим для багатьох завдань.

Таким чином, RNN отримує на вхід не лише «сьогодення» (поточні дані), а й інформацію з недавнього минулого. Це робить їх надзвичайно потужними для розв'язання задач, які недоступні іншим алгоритмам.

Як і всі інші алгоритми глибокого навчання, рекурентні нейронні мережі призначають вагові коефіцієнти, але в RNN ці ваги застосовуються як до поточного, так і до попереднього вводу. Крім того, RNN налаштовують ваги за допомогою методів градієнтного спуску та зворотного поширення в часі (Backpropagation Through Time, BPTT), що дозволяє їм ефективно навчатися на послідовних даних.

1.3.3. Довга-короткочасна пам'ять

Довга-короткочасна пам'ять (Long Short-Term Memory, LSTM) є вдосконаленням рекурентних нейронних мереж, яке значно покращує їх

здатність до зберігання інформації. Ця модель особливо ефективна для задач, що вимагають врахування досвіду, який може бути далеко в минулому. Елементи LSTM використовуються як основні компоненти для шарів RNN, які часто називають мережею LSTM.

У LSTM є три типи воріт (gates): ворота вводу (input gate), ворота забування (forget gate) та ворота виводу (output gate). Ці ворота виконують важливі функції: вони контролюють, чи впускати нові дані (input gate), видаляти непотрібну інформацію (forget gate) або дозволяти інформації впливати на результати на поточному етапі (output gate). Завдяки цій структурі LSTM може ефективно зберігати та маніпулювати даними з різними часовими залежностями.

Таким чином, LSTM не лише покращує пам'ять у порівнянні з традиційними RNN, але й забезпечує більшу гнучкість у роботі з послідовними даними. Це робить їх особливо корисними в таких сферах, як обробка природної мови, розпізнавання мови, прогнозування часових рядів та багато інших задач, де важливо враховувати контекст минулого. На рис. 1.2. зображено RNN з трьома воротами LSTM, що демонструє їхню архітектурну структуру.

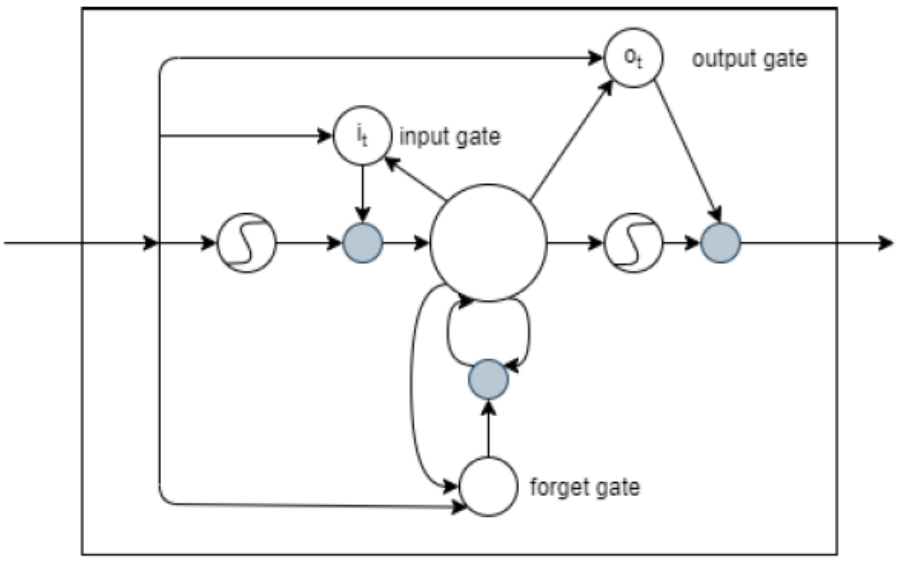

Рисунок 1.2 – Схема Long short-term memory

1.3.4. Моделі трансформер

Стаття «Увага — це все, що вам потрібно» (2017) [5] представляє нову архітектуру, відому як Трансформер. Як і LSTM, Трансформер призначений для перетворення однієї послідовності в іншу, використовуючи дві основні складові: енкодер і декодер. Проте, на відміну від раніше існуючих моделей seq2seq, Трансформер не містить жодних рекурентних мереж, таких як GRU чи LSTM.

На рис. 1.3 можна побачити архітектуру Трансформера: енкодер розташований зліва, а декодер — справа. Обидва компоненти складаються з модулів, які можна комбінувати в різних кількостях (позначено як Nx на рис. 1.3). Основними елементами цих модулів є шари Multi-Head Attention та Feed Forward. Вхідні та вихідні послідовності (цільові речення) спершу перетворюються у n-мірний простір, оскільки текстові рядки не можуть бути використані безпосередньо.

Ключовим аспектом моделі є позиційне кодування слів. Оскільки у Трансформера немає рекурентних мереж, які б дозволяли запам'ятовувати порядок введення, необхідно надати кожному слову або леммі у вхідній послідовності інформацію про їхнє відносне положення. Це важливо, оскільки значення слів залежить від їхнього місця у контексті. Позиційна інформація додається до векторних представлень (embedded representation) кожного слова, що дозволяє моделі враховувати порядок слів при обробці даних.

Таким чином, архітектура Трансформера забезпечує ефективну обробку послідовностей, зберігаючи інформацію про позиції елементів, що робить її надзвичайно потужною для різноманітних завдань у машинному навчанні, зокрема в обробці природної мови.

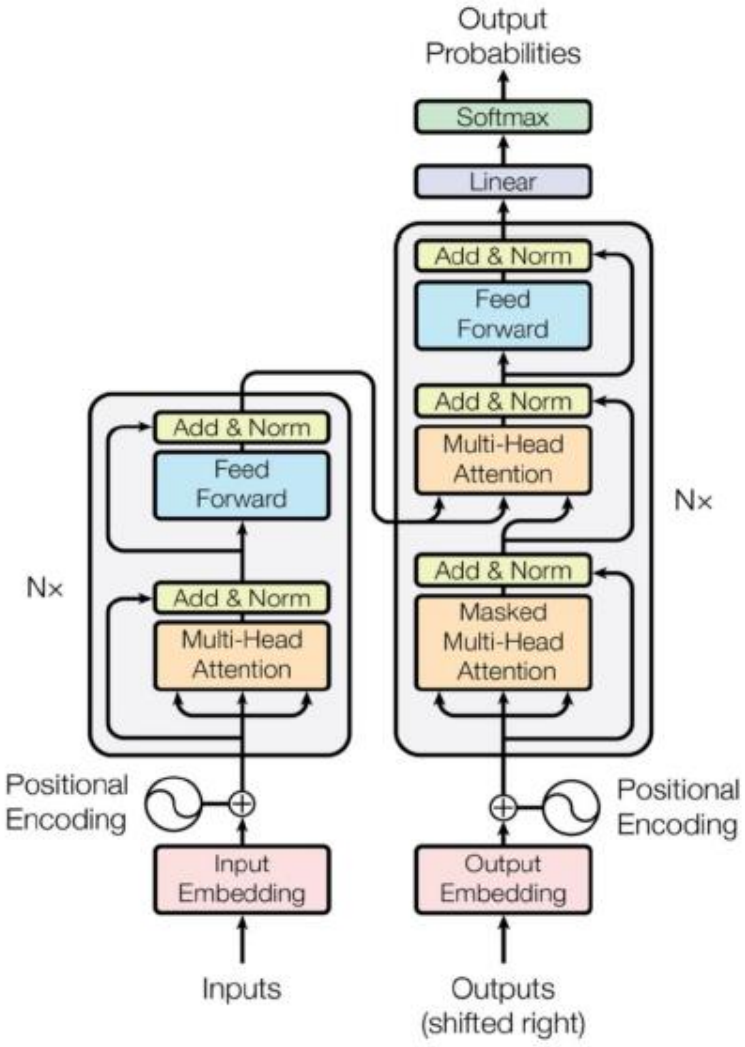

Рисунок 1.3 – Архітектура моделі трансформера.

1.4 Оцінка якості машинного перекладу

Оцінка якості машинного перекладу є важливим етапом у розвитку та вдосконаленні систем автоматичного перекладу. Адже саме від якості залежить, наскільки корисним та ефективним буде машинний переклад для користувачів. Оцінка якості дозволяє визначити сильні та слабкі сторони системи, порівняти різні системи між собою та визначити напрямки подальшого розвитку.

Існує два основних підходи до оцінки якості машинного перекладу: автоматична та ручна. Автоматичні методи використовують математичні алгоритми для порівняння машинного перекладу з еталонним перекладом,

виконаним людиною. Найпоширенішими метриками є BLEU (Bilingual Evaluation Understudy) та METEOR (Metric for Evaluation of Translation with Explicit ORdering). BLEU оцінює точність перекладу шляхом підрахунку збігів слів та фраз, тоді як METEOR враховує також синоніми та порядок слів.

Хоча автоматичні метрики є швидкими та зручними, вони не завжди точно відображають якість перекладу з точки зору людини. Тому важливим є також ручна оцінка, яка проводиться професійними перекладачами або носіями мови. Вони оцінюють такі аспекти, як точність, граматична правильність, стилістична відповідність та природність мови. Ручна оцінка є більш суб'єктивною, але вона дозволяє врахувати нюанси мови та контексту, які важко формалізувати в автоматичних метриках.

Окрім зазначених методів, існують також комбіновані підходи, які поєднують автоматичну та ручну оцінку. Наприклад, система може автоматично виділяти фрагменти тексту, які потребують уваги перекладача, або ж пропонувати альтернативні варіанти перекладу для ручного вибору.

Вибір методу оцінки залежить від конкретних завдань та вимог до якості перекладу. Для попередньої оцінки та порівняння систем можна використовувати автоматичні метрики. Однак для остаточної оцінки та забезпечення високої якості перекладу необхідна ручна оцінка експертів.

Висновки за розділом 1

Машинний переклад пройшов довгий шлях розвитку від перших спроб у 50-х роках до сучасних нейронних мереж. Кожен етап цього розвитку приносив нові підходи та методи, спрямовані на покращення якості та точності перекладу. Від правил та словників до статистичних моделей та нейронних мереж, машинний переклад постійно вдосконалюється, наближаючись до здатності "розуміти" текст та враховувати контекст. Сучасні системи, засновані на нейронних мережах, здатні навчатися на величезних обсягах даних та генерувати природні та точні переклади, хоча все ще стикаються з

певними труднощами, такими як обмеженість даних для малопоширених мов та складність перекладу спеціалізованих текстів.

Незважаючи на значний прогрес, машинний переклад все ще не є ідеальним. Існують проблеми з багатозначністю, розбіжностями в граматиці між мовами, культурними особливостями та етичними аспектами. Подолання цих перешкод є пріоритетом для розробників, які фокусуються на глибинному навчанні, механізмах уваги та інших підходах, спрямованих на створення більш точних, природних та адаптивних систем. Важливою складовою розвитку машинного перекладу є оцінка його якості, яка може бути як автоматичною, так і ручною.

В цілому, машинний переклад є потужним інструментом, який значно спрощує комунікацію та доступ до інформації для людей з різних країн. З розвитком технологій та збільшенням обсягів даних можна очікувати подальшого вдосконалення систем машинного перекладу, що зробить їх ще більш точними, швидкими та універсальними.

РОЗДІЛ 2.
НЕЙРОМЕРЕЖЕВА МОДЕЛЬ ТРАНСФОРМЕР

2.1 Загальні відомості про модель Трансформер

Трансформери — це сучасна архітектура нейронних мереж, яка суттєво підвищила ефективність моделей машинного навчання, особливо в галузі обробки природної мови (NLP) та комп'ютерного зору. Цю архітектуру вперше представили дослідники компанії Google у 2017 році в статті під назвою «Attention is All You Need».[5] Один із найяскравіших прикладів використання трансформерів — це речення, яке складається з упорядкованого набору слів.

Трансформери генерують цифрове представлення кожного елемента послідовності, захоплюючи важливу інформацію про нього та його контекст. Ці представлення можуть передаватися іншим нейронним мережам, що дозволяє їм використовувати отриману інформацію для розв'язання конкретних задач, таких як синтез та класифікація. Представлення, створені за допомогою трансформерів, допомагають виявляти та розуміти приховані шаблони і взаємозв'язки у вхідних даних, що робить ці нейромережеві моделі ефективними для завдань синтезу послідовностей та взаємопов'язаних результатів.

Основна перевага трансформерів полягає в їхній здатності обробляти послідовності даних паралельно завдяки механізму уваги (attention mechanism), зокрема мультиголовій самоувазі (multi-head self-attention). Це забезпечує трансформерам значні переваги в порівнянні з попередніми архітектурами, такими як рекурентні нейронні мережі (RNN) та довго-короткочасна пам'ять (LSTM).

Ключові переваги трансформерів:

а) *паралелізація:*

- Ефективність обчислень. Трансформери можуть обробляти всі елементи вхідної послідовності одночасно, що значно прискорює навчання та зменшує час обробки даних.
- Використання апаратного прискорення. Завдяки можливості паралелізації, трансформери ефективно використовують сучасні графічні (GPU) та тензорні процесори (TPU), що сприяє швидшому навчання великих моделей.

б) *контекст та довгострокові залежності:*

- Глобальний контекст. Механізм уваги дозволяє моделі враховувати всі елементи послідовності при обчисленні кожного виходу, що покращує розуміння контексту.
- Довгострокові залежності. Трансформери ефективно обробляють залежності між віддаленими елементами в послідовності, у той час як RNN та LSTM мають обмежену здатність зберігати довгострокову інформацію.

в) *гнучкість та узагальнюваність*:

- Універсальність. Трансформери застосовуються для різноманітних завдань, включаючи обробку природної мови, машинний переклад, генерацію тексту, розпізнавання зображень тощо.
- Масштабованість. Архітектура трансформерів добре масштабується, що дозволяє створювати моделі з мільярдами параметрів, такі як GPT-3 від OpenAI або BERT від Google.

Приклади застосування трансформерів:

- BERT (Bidirectional Encoder Representations from Transformers): використовується для розуміння природної мови, включаючи аналіз настроїв, відповіді на запитання та розпізнавання сутностей.
- GPT (Generative Pre-trained Transformer): застосовується для генерації тексту, машинного перекладу, створення діалогових систем та інших задач.

2.2. Підготовка даних до обробки

2.2.1. Токенізація

Перед подачею даних до трансформерів, їх необхідно перетворити на послідовні токени. Токени – це основні елементи, на які розбивається текст під час обробки природної мови (NLP) [2].

Токенізація – це процес перетворення тексту на список токенів. Кожен токен може представляти собою слово, частину слова, символ або групу символів, залежно від завдання та застосовуваних методів токенізації.

Види токенів:

- Словесні токени (Word Tokens). Кожне слово є окремим токеном. Наприклад, у реченні «The cat sat on the mat» кожне слово розглядається як окремий токен: [«The», «cat», «sat», «on», «the», «mat»].
- Підсловні токени або морфеми (Subword Tokens). Слова можуть бути розділені на менші частини або морфеми, що особливо актуально для мов з великою кількістю словоформ. Наприклад, слово «unhappiness» може бути розділене на [«un», «happiness»].
- Символьні токени (Character Tokens). Кожен символ тексту є окремим токеном. Це корисно для мов з великим алфавітом або при обробці текстів з помилками чи нестандартним написанням. Наприклад, «hello» може бути токенізоване як [«h», «e», «l», «l», «o»].

Використання токенів у машинному навчанні:

- Моделі обробки природної мови. Токени слугують основними вхідними даними для моделей, таких як трансформери, які навчаються на послідовностях токенів.
- Векторизація тексту. Токени перетворюються на числові вектори (ембеддінги), які використовуються як вхідні дані для нейронних мереж.
- Аналіз тексту. Токени застосовуються для виконання різних видів текстового аналізу, таких як частотний аналіз, пошук ключових слів, машинний переклад тощо.

Методи токенізації **.Серед *методів токенізації* відомі такі:**

- Пробіл-розділена токенізація (Whitespace Tokenization). Текст розбивається на слова на основі пробілів. Це простий метод, але він не враховує пунктуацію та інші символи.
- Токенізація за регулярними виразами (Regex Tokenization). Використовуються регулярні вирази для точнішого розділення тексту на токени, що дозволяє враховувати пунктуацію.
- Byte-Pair Encoding (BPE). Цей метод об'єднує часті пари символів у нові токени, що дозволяє ефективно працювати з рідкісними словами. Часто використовується в трансформерах.
- WordPiece. Схожий на BPE, але використовує інший алгоритм для формування токенів. Застосовується в моделях, таких як BERT.
- SentencePiece. Метод, що не потребує попередньої токенізації тексту, роблячи його більш універсальним для різних мов і завдань.

Приклад токенізації. Розглянемо простий приклад роботи з трансформерами та токенами на задачі перетворення певної послідовності токенів, використовуючи словник як таблицю для пошуку. Необхідно зіставити слова з числами, адже будь-яке слово можна співвіднести з будь-яким числом (рис. 2.1).

```
Raw text: A black cat
Vocab: {"A": 0, "black": 1, "cat": 2, "cats": 3}
Tokenized text: [0, 1, 2]
```

Рисунок 2.1 – Приклад кодування тексту

Можна побачити найпростіший приклад кодування тексту та помітити, що слова cats та cat було закодовано різними токенами, незважаючи на те, що це одне і те слово, але у різній формі множини.

Також існують більш продвинуті способи токенізаці,ї при якій перед тим, як присвоїти індекси словам, їх розбивають на фрагменти. В ході

експериментів також була помічена зручність у використання окремих токенів, які б позначали кінець та початок речень, адже це надає більше контексту.

Розглянемо приклад токенізації на представленому реченні: «Hello there, isn't the weather nice today in Drosval?»

Використовуючи токенізатор bert-base-uncased (BBU) із бібліотеки «transformers library» можна перетворити речення на послідовність токенів (рис. 2.2).

```
[101, 7592, 2045, 1010, 3475, 1521, 1056, 1996, 4633, 3835, 2651, 1999, 2852, 2891, 10175, 1029, 102]
```

Рисунок 2.2 – Перетворення речення на послідовність токенів

Числа, які було присвоєно словам, будуть змінюватися в залежності від обраного способу токенізації та від методу навчання моделі.

Після декодування надаються слова, які було отримано з токенів (рис. 2.3).

```
[CLS] hello there , isn ’ t the weather nice today in dr ##os ##val ? [SEP]
```

Рисунок 2.3 – Слова, представлені токенами

Як можна побачити, результат не такий, який очікувався, але кінцевий результат не схожий на вхідне речення. Такий результат вийшов через те, що було додано певні додаткові токени та вигадану нами назву модель представила декількома фрагментами. Великі літери також було втрачено з причини використання такого виду моделі, яка їх не підтримує.

Треба враховувати, що трансформери також можуть обробляти не лише текст, а й виконувати певні задачі з візуалізації.

2.2.2 Векторизація тексту

Моделювання мови передбачає обробку великих обсягів вхідних даних, тому ефективне представлення тексту є критично важливим для розробки алгоритмів його аналізу. Існує кілька методів *векторизації* тексту:

• **One-hot encoding**. Цей метод полягає в розбивці тексту на токени (слова) і підрахунку їх кількості в корпусі. Вибирається фіксований розмір словника NNN, і кожному з NNN найуживаніших токенів присвоюється цілочисельний індекс від 0 до N−1. Слова, що не потрапили до словника, замінюються на спеціальний токен OOV (out of vocabulary) з індексом N. Кожному з N+1 токенів присвоюється вектора $o \in R^{n+1}$ де $o_i = 1(i = k)$, k – індекс токена, а 1 — функція-індикатор. Це дозволяє уникнути надання ваги індексам слів, оскільки всі вектори мають однакову величину. Перетворення послідовності з M слів призводить до створення матриці $M \times N$, де кожен j-ий рядок відповідає one-hot вектору j-го слова.

• **Bag-of-words**. Цей метод перетворює послідовність з M токенів на вектор розмірності N+1, де N — розмір словника. Основна відмінність від one-hot encoding полягає в підрахунку кількості кожного токена в словнику, включаючи OOV. Цей підхід дозволяє працювати з текстами різної довжини. Одним з розширень є метод tf-idf [12], що дозволяє надавати ваги токенам з додатковою інформаційною цінністю та зменшувати ваги малоінформативних токенів (наприклад, прийменників і сполучників).

• **Bag of character n-grams**.[13] Цей метод використовує n-грам на символьному рівні для створення векторної репрезентації кожного слова як суми векторних репрезентацій символьних n-грам, що робить його простим в обчисленні та дозволяє використовувати інформацію про структуру слова.

• **Byte-Pair Encoding (BPE)**.[14] Цей метод передбачає рекурсивну заміну найчастішої пари символів на символ заміни, поки всі пари не стануть унікальними. Це дозволяє алгоритмам обробки даних працювати з стисненою репрезентацією, яка зберігає інформацію про субсловну структуру. Для мов з

розширеним алфавітом або текстів, що використовують кодування UTF-8 або UTF-16, можна застосовувати BPE байтового рівня (BBPE). [15]

- **Word2vec** [16]. Цей алгоритм, що не використовує глибоке навчання, перетворює слова на векторні репрезентації, зберігаючи семантику. Він ґрунтується на контексті (n слів перед і після) для покращення векторів слів. Існує два підходи: Continuous Bag of Words (CBOW) і "skip-gram", що поліпшує вектори контексту на основі поточного слова. На рис. 2.4 показано схему роботи цих підходів [17].

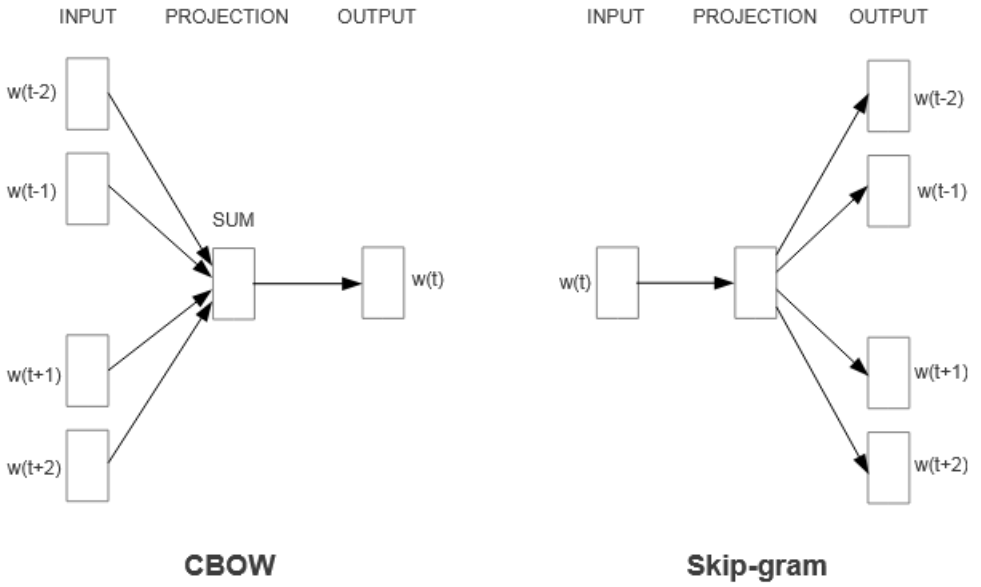

Рисунок 2.4 – Різниця між CBOW та skip-gram підходами в алгоритмі word2vec

Дослідники наводять приклади складних семантичних зв'язків, які навіть проста модель може вловити. Завдяки векторним репрезентаціям слів можна продемонструвати, що різниця між цими репрезентаціями відображає змістовні відносини між словами. Наприклад, різниця між векторними представленнями слів "король" і "королева" дуже близька до різниці між "чоловік" і "жінка". Це дозволяє використовувати своєрідну семантичну арифметику у векторному просторі — наприклад, (Athens - Greece) + Oslo =

Norway. Тут відношення між векторами "Афіни" та "Греція" є відношенням між столицею та країною, а додавання вектора "Осло" призводить до вектора, близького до слова "Норвегія".

Існують також підходи, які спрямовані на зменшення розмірності даних шляхом скорочення граматичної чи семантичної інформації через приведення споріднених слів до спільної форми (канонікалізація). Основними методами є стемінг і лематизація.

• **Стемінг** — це процес, що дозволяє зводити слова до їх основної форми, позбавленої допоміжних частин, таких як суфікси чи закінчення[18]. Наприклад, слова "швидко", "швидкий" і "швидкі" перетворюються на "швидк", а "бігом", "бігаю" та "бігати" — на корінь "біг". Існує кілька способів реалізації цього алгоритму, зокрема, через таблиці, відсікання закінчень та стохастичні методи.

• **Лематизація** — це підхід у обчислювальній лінгвістиці, що дозволяє звести слово до форми, яка зберігає його семантику, хоча граматика може не зберігатися [19]. Наприклад, слова "ходить", "ходитимуть", "ходити" і "ходила" зводяться до однієї лексеми — "ходити". Головна відмінність лематизації від стемінгу полягає в урахуванні контексту та семантичних зв'язків між словами, що робить її більш складною. Проте правильне визначення леми залежить від якості маркування частин мови (POS tagging).

2.3 Ембеддінг токенів

Ембедінг — це метод представлення об'єктів (зазвичай слів або фраз) у вигляді векторів у багатовимірному просторі. Вони використовуються в машинному навчанні, зокрема в завданнях обробки природної мови (NLP) та комп'ютерного зору, для перетворення нечислових даних у числовий формат, що зручно для нейронних мереж та інших алгоритмів.

Основні концепції ембедінгів:

- Контекстуальність. Ембедінги враховують контекст слова, тобто його значення може змінюватися в залежності від оточення в тексті. Це забезпечує кращу семантичну інтерпретацію порівняно з традиційними підходами, такими як Bag-of-Words (BoW) або TF-IDF.
- Високовимірний простір. Слова або інші об'єкти представляються у вигляді векторів у просторі з великою кількістю вимірів. Вектори, що представляють схожі за значенням слова, розташовані ближче один до одного.

Методи **створення ембедінгів:**

- Word2Vec. Алгоритм, розроблений у 2013 році командою Google, що створює векторні представлення слів. Він має два основних варіанти: Skip-gram та CBOW (Continuous Bag of Words), які використовують нейронні мережі для передбачення контексту слова або слова за його контекстом.
- GloVe (Global Vectors for Word Representation) - метод, розроблений командою Stanford, що використовує статистичні дані про частоти спільного використання слів у великому корпусі текстів для створення векторів.
- FastText. Розроблений Facebook, цей метод розширює Word2Vec, включаючи підслова, що дозволяє моделі краще працювати з морфологічно складними мовами та рідкісними словами.
- Контекстуальні ембедінги (BERT, GPT, ELMo) враховують контекст слова в реченні, що дозволяє створювати різні представлення для одного і того ж слова залежно від його оточення. Наприклад, BERT (Bidirectional Encoder Representations from Transformers) генерує контекстуальні ембедінги, які враховують всю послідовність слів у обох напрямках.

Ембедінги використовують для:

- класифікації тексту. Ембедінги слугують вхідними даними для нейронних мереж, що виконують класифікацію текстів на різні категорії.
- аналізу настроїв. Вектори слів допомагають визначити позитивний або негативний настрій тексту.

- машинного перекладу. Ембедінги використовуються для представлення слів у різних мовах, що полегшує процес перекладу.
- пошуку та рекомендацій. Схожість між ембедінгами може бути використана для пошуку схожих документів або рекомендацій товарів.

Якщо є послідовність цілих чисел, яка представляє вхідні дані, її можна перетворити на ембедінги, які передають скорочений сенс кожного токена у вигляді числової послідовності. Спочатку виконується синтез послідовних чисел як випадкової послідовності, а під час навчання формується значиме уявлення.

Основним обмеженням ембедінгів є те, що вони не враховують контекст речення, в якому відбувається синтез токенів. Цей недолік має спадковий характер і виникає через два основні фактори.

По-перше, вхідна задача визначає подальші дії, адже може бути необхідно зберегти порядок токенів під час їх перетворення на ембедінги, що є особливо важливим в обробці природних мов. Основна ідея полягає в тому, що ми використовуємо додатковий набір ембедінгів, який вказує позиції кожного токена в певній послідовності. Цей набір зазвичай комбінується з ембедінгами токенів.

Ще однією проблемою є те, що токени можуть мати різне значення в залежності від сусідніх токенів. Наприклад, у реченнях «It’s dark, who turned off the light?» та «Wow, this parcel is really light!» слово "light" використовується в різних контекстах, що призводить до абсолютно різних значень. Однак, залежно від методу токенізації, ембедінги можуть залишатися однаковими. У трансформерах це вирішується за допомогою механізму уваги.

2.4 Механізм уваги

Механізм уваги є основним компонентом архітектури трансформерів, який дозволяє моделям ефективно обробляти послідовності даних, зокрема в

завданнях обробки природної мови. Він забезпечує можливість моделі фокусуватися на різних частинах вхідних даних залежно від контексту.

Механізм уваги дозволяє моделі навчатися, на які слова або символи слід звертати увагу при обробці кожного елемента послідовності. Це здійснюється шляхом обчислення "ваг уваги", які відображають важливість кожного слова в контексті інших слів.

Основні компоненти механізму уваги:

– Запити (Queries): Вектори, які представляють елементи, для яких ми хочемо знайти відповідні значення. Наприклад, у задачі перекладу запитом може бути слово, для якого ми намагаємося знайти переклад.

– Ключі (Keys): Вектори, що представляють елементи, з яких ми хочемо отримати інформацію. Вони можуть бути аналогічні запитам, але вони використовуються для обчислення уваги.

– Значення (Values): Вектори, які містять саму інформацію, яку ми хочемо отримати. Значення можуть бути векторизованими формами слів або фраз.

Обчислення механізму уваги здійснюється в декілька етапів.

1. Обчислення подібності. Для кожного запиту обчислюються подібності з ключами, зазвичай за допомогою скалярного добутку.

$$\text{Attention\_Score} = \text{Q} \cdot \text{K}^{\text{T}}. \tag{2.1}$$

2. Нормалізація. Отримані оцінки уваги нормалізуються за допомогою softmax, щоб отримати ймовірності, які вказують на важливість кожного елемента.

$$\text{Attention\_Weights} = \text{softmax}\left(\frac{\text{Q}\cdot\text{K}^{T}}{\sqrt{d_k}}\right), \tag{2.2}$$

де d_k — це розмірність ключів, що використовується для стабілізації градієнтів.

3. Зважене додавання. Зважені значення обчислюються шляхом множення ваг на відповідні значення.

$$\text{Output} = \text{Attention Weights} \cdot V. \quad (2.3)$$

Приклад роботи механізму уваги

Розглянемо простий приклад. Припустимо, задане речення: “*The cat sat on the mat.*”

Крок 1. Векторизація.

Перш за все, необхідно перетворити слова у вектори (ембедінги):

- "The" → [0.2, 0.1];
- "cat" → [0.5, 0.6];
- "sat" → [0.4, 0.3];
- "on" → [0.1, 0.2];
- "the" → [0.2, 0.1];
- "mat" → [0.6, 0.5].

Крок 2. Створення запитів, ключів і значень

Нехай, використовується слово "sat" як запит. Тоді:

- Запит (Q) для "sat" = [0.4, 0.3];
- Ключі (K) = вектори всіх слів у реченні;
- Значення (V) = ті ж самі вектори.

Крок 3. Обчислення схожості.

Обчислюється подібність “sat” з усіма словами:

"The": 0.4·0.2 + 0.3·0.1 = 0.080.4 \cdot 0.2 + 0.3 \cdot 0.1 = 0.080.4·0.2 + 0.3·0.1 = 0.08;

"cat": 0.4·0.5 + 0.3·0.6 = 0.390.4 \cdot 0.5 + 0.3 \cdot 0.6 = 0.390.4·0.5 + 0.3·0.6 = 0.39;

"sat": 0.4·0.4 + 0.3·0.3 = 0.250.4 \cdot 0.4 + 0.3 \cdot 0.3 = 0.250.4·0.4 + 0.3·0.3 = 0.25;

"on": 0.4·0.1 + 0.3·0.2 =0.10.4 \cdot 0.1 + 0.3 \cdot 0.2 = 0.10.4·0.1 + 0.3·0.2 = 0.1;

"the": 0.4·0.2 + 0.3·0.1 = 0.080.4 \cdot 0.2 + 0.3 \cdot 0.1 = 0.080.4·0.2 + 0.3·0.1 = 0.08;

"mat": 0.4·0.6 + 0.3·0.5 = 0.470.4 \cdot 0.6 + 0.3 \cdot 0.5 = 0.470.4·0.6 + 0.3·0.5 = 0.47.

Крок 4. Нормалізація.

Нормалізуються отримані значення за допомогою функції softmax.

Крок 5. Зважене додавання.

Отримані ваги множаться на відповідні значення, щоб отримати вихід для "sat".

Переваги механізму уваги:

- Гнучкість. Механізм уваги може адаптуватися до різних контекстів, що робить його ефективним для багатьох завдань.
- Паралелізація. На відміну від RNN, де обробка послідовностей відбувається послідовно, трансформери можуть обробляти всю послідовність одночасно, що значно пришвидшує тренування.

2.5 Особливості навчання Трансформерів

При навчанні трансформерів застосовуються різні методи, що охоплюють як базові принципи, так і вдосконалені підходи для оптимізації процесу.

2.5.1 Ініціалізація вагових коефіцієнтів

Ініціалізація вагових коефіцієнтів є фундаментальним етапом у процесі навчання нейронних мереж, особливо таких складних, як трансформери. Вона впливає на швидкість збіжності, стабільність навчання та якість отриманої моделі. Даний розділ надає детальний огляд основних методів ініціалізації ваг, їх математичного обґрунтування та специфіки застосування в контексті трансформерів.

Неправильна ініціалізація може призвести до таких проблем, як:

– затухання градієнтів: ваги, ініціалізовані занадто малими значеннями, можуть призвести до того, що градієнти стануть дуже малими, ускладнюючи навчання;

– проблема з експлозією градієнтів: ваги, ініціалізовані занадто великими значеннями, можуть викликати випадання градієнтів, що ускладнить стабільність навчання4

– симетрія: однакова ініціалізація ваг у нейронах одного шару може призвести до того, що всі нейрони будуть навчатися однаково, що знижує їхню ефективність.

Ініціалізація вагів може відбуватися з використанням статистичних законів розподілу або за спеціальними методами.

Нормальний розподіл. Ваги W ініціалізуються з нормального розподілу зі середнім 0 і стандартним відхиленням σ:

$$W \sim N(0, \sigma^2).$$

Дисперсія σ^2 часто вибирається емпірично або обчислюється за формулами, запропонованими в методах He або Xavier.

Рівномірний розподіл. Ваги W ініціалізуються з рівномірного розподілу в інтервалі [$-a$, a]:

$$W \sim U(-a, a).$$

Значення a часто вибирається емпірично або обчислюється за формулами, запропонованими в методах He або Xavier.

Ініціалізація He [1]. Для шарів з активацією ReLU дисперсія ваг обчислюється за формулою:

$$\sigma = \sqrt{\frac{2}{fan\_in}},$$

де fan_in – кількість входів нейрона.

He-ініціалізація намагається зберегти дисперсію активацій постійною під час проходження вперед по мережі.

Ініціалізація Xavier (Glorot)[10]. Для шарів з симетричними активаціями (tanh, sigmoid) дисперсія ваг обчислюється за формулою:

$$\sigma = \sqrt{\frac{2}{(fan\_in + fan\_out)}},$$

де fan_in – кількість входів нейрона, fan_out – кількість виходів нейрона.

Xavier ініціалізація намагається зберегти дисперсію градієнтів постійною під час зворотного поширення помилки.

Специфіка ініціалізації в трансформерах:

- Multi-Head Attention: кожна голова уваги має свої ваги, які ініціалізуються окремо.
- Позиційне кодування: ваги, що відповідають за позиційне кодування, зазвичай ініціалізуються випадковими значеннями з певного розподілу або навчаються спільно з моделлю.
- Нормалізація шарів: нормалізація шарів (Layer Normalization) допомагає стабілізувати навчання і може впливати на вибір методу ініціалізації.

Додаткові методи та нюанси ініціалізації ваг у трансформерах включають ортогональну ініціалізацію, яка використовує ортогональні матриці для покращення умови обумовленості, що може сприяти стабільнішому навчанні. Важливим аспектом є також ініціалізація за допомогою попередньо навчених моделей, де ваги з моделей, що навчалися на великих датасетах, слугують початковою точкою. Адаптивна ініціалізація передбачає динамічну зміну масштабних коефіцієнтів під час навчання, що дозволяє більш гнучко реагувати на умови навчання. Крім того, різні активаційні функції вимагають різних стратегій ініціалізації, оскільки вони впливають на розподіл виходів. Вибір оптимізатора також може суттєво впливати на ефективність різних методів ініціалізації, оскільки різні оптимізатори мають свої специфічні вимоги до початкових значень ваг.

Вибір методу ініціалізації вагових коефіцієнтів у нейронних мережах, трансформерах, залежить від кількох ключових факторів:

1. Тип архітектури: різні архітектури можуть вимагати специфічних методів ініціалізації. Наприклад, трансформери, які використовують механізм уваги, можуть виграти від ортогональної або Не ініціалізації.

2. Активаційні функції: вибір активаційних функцій (ReLU, tanh, sigmoid тощо) впливає на оптимальний метод ініціалізації. Наприклад, ReLU зазвичай використовує Не ініціалізацію, тоді як для tanh більше підходить Xavier ініціалізація.

3. Глибина мережі: у глибоких мережах важливо уникати затухання або експлозії градієнтів, що може спричинити погана ініціалізація. Тому методи, які враховують розподіл ваг, стають критичними.

4. Обсяг даних: якщо модель навчається на малих датасетах, використання попередньо навчених ваг може бути корисним для покращення загальної продуктивності.

5. Оптимізатор: вибір оптимізатора (SGD, Adam, RMSprop тощо) також впливає на ефективність ініціалізації, оскільки різні оптимізатори можуть по-різному реагувати на стартові значення ваг.

6. Складність завдання: для складних задач, які вимагають високої точності, може бути доцільно використовувати адаптивні методи ініціалізації, які враховують динаміку навчання.

7. Експерименти та емпіричні дослідження: часто вибір методу базується на практичних спостереженнях та результатах попередніх експериментів, що можуть вказати на найбільш ефективні підходи для конкретних умов.

Ці фактори слід враховувати під час вибору методу ініціалізації, оскільки вони можуть суттєво вплинути на продуктивність і стабільність навчання нейронної мережі.

2.5.2 Оптимізатори нейронних мереж трансформерів

Трансформери, революційна архітектура нейронних мереж, досягла значних успіхів у таких областях, як обробка природної мови, машинний переклад та генерація тексту. Однак, ефективне навчання таких складних моделей вимагає застосування потужних методів оптимізації. В умовах

складності архітектури яка використовуються в різноманітних завданнях обробки природної мови та комп'ютерного зору, вибір оптимізатора може суттєво вплинути на результативність моделі. Розглянимо основні методи оптимізації, що застосовуються для трансформерів, їх переваги та недоліки, а також специфіку їх використання в навчанні.

Stochastic Gradient Descent (SGD) є класичним методом оптимізації, який оновлює ваги моделі на основі градієнта функції втрат для випадково обраної підмножини даних. Основна перевага SGD полягає в простоті та надійності. Однак цей метод може бути повільним у зближенні до оптимуму, особливо при навчанні великих моделей, таких як трансформери.

Adam (Adaptive Moment Estimation) є одним з найпопулярніших оптимізаторів у сучасному глибокому навчанні. Він комбінує переваги адаптивних методів (таких як AdaGrad) і методів з моментами. Adam використовує експоненційно зважене середнє градієнтів і квадратів градієнтів для адаптивного регулювання швидкості навчання. Це робить його особливо ефективним для трансформерів, які мають велику кількість параметрів, оскільки Adam забезпечує швидше зближення до оптимуму.

RMSprop (Root Mean Square Propagation) є модифікацією SGD, яка використовує експоненційно зважене середнє квадрата градієнтів для регулювання швидкості навчання. Цей метод демонструє хороші результати при навчанні трансформерів, оскільки дозволяє уникати проблеми затухання градієнтів і забезпечує стабільне та швидке зближення до оптимуму.

Adagrad (Adaptive Gradient Algorithm) адаптивно змінює швидкість навчання для кожного параметра на основі історії градієнтів. Цей метод є особливо корисним для рідкісних параметрів, але може призводити до занадто малих швидкостей навчання в процесі, що негативно впливає на результати.

AdamW є варіантом методу Adam, який включає вагове розкладання (weight decay) для регуляризації. Цей метод допомагає покращити

узагальненість моделей на великих наборах даних, що є критично важливим для трансформерів, які часто навчаються на великих корпусах тексту.

Nadam (Nesterov-accelerated Adaptive Moment Estimation) об'єднує Adam і Nesterov's accelerated gradient, пропонуючи поліпшену продуктивність порівняно з Adam, особливо в складних завданнях, що вимагають високої точності.

Вибір оптимізатора для трансформерів залежить від кількох факторів, таких як обсяг даних, складність задачі, ресурси обчислення та специфіка моделі. Наприклад, для великих наборів даних можуть бути доцільні адаптивні методи, такі як Adam або RMSprop, тоді як для менших наборів даних класичні методи, такі як SGD, можуть бути більш ефективними.

2.5.3. Регуляризація в Трансформерах

Перенавчання виникає, коли модель занадто добре вивчає навчальний набір даних, включаючи шум і випадкові коливання, що призводить до погіршення її продуктивності на нових, невідомих даних. Трансформери, завдяки своїй складній архітектурі, є особливо вразливими до цієї проблеми, тому регуляризація стає необхідною.

1. L1 і L2 регуляризація: L1 регуляризація (Lasso) додає до функції втрат суму абсолютних значень ваг. Сприяє розрідженості моделі, тобто багатьом вагам присвоюється нульове значення; L2 регуляризація (Ridge) додає до функції втрат суму квадратів ваг. Зменшує величину ваг і зменшує ризик перенавчання.

2. Dropout є ефективним методом регуляризації, що передбачає випадкове "вимкнення" частини нейронів під час навчання. Це зменшує залежність між нейронами, що, в свою чергу, покращує узагальненість моделі. У трансформерах dropout зазвичай застосовується в шарах самої уваги та в feed-forward мережах.

3. Раннє зупинення (Early Stopping) – це стратегія, яка полягає в моніторингу продуктивності моделі на валідаційному наборі і зупинці навчання, коли продуктивність починає погіршуватися. Це дозволяє уникнути перенавчання, зберігаючи модель на найкращій ітерації.

4. Аугментація даних включає в себе генерацію нових навчальних зразків шляхом модифікації існуючих (наприклад, шляхом додавання шуму, зміни масштабу або обертання). Це допомагає моделі навчатися на більш різноманітних даних, що зменшує ризик перенавчання.

5. Регуляризація за допомогою контрастивного навчання полягає в навчанні моделі розрізняти позитивні та негативні приклади в контексті даних. У трансформерах це може бути реалізовано через контрастивні втрати, що спонукають модель навчатися більш загальним представленням даних.

Осиобливості регуляризації в трансформерах:

- Механізм уваги. Через складність механізму уваги в трансформерах можуть виникати проблеми з перенавчанням. Регуляризація допомагає уникнути цих проблем.
- Велика кількість параметрів. Трансформери мають велику кількість параметрів, що збільшує ризик перенавчання. Тому застосування регуляризації є особливо важливим.
- Позиційне кодування. Регуляризація може допомогти стабілізувати навчання позиційного кодування.

Регуляризація є невід'ємною частиною навчання трансформерів. Вона допомагає запобігти перенавчанню, покращує узагальнювальну здатність моделі та підвищує її стійкість до шуму в даних. Вибір оптимальних методів регуляризації вимагає глибокого розуміння як самих трансформерів, так і принципів регуляризації в нейронних мережах.

2.5.4 Функції втрат в Трансформерах

Функції втрат визначають, наскільки добре модель відповідає навчальним даним і напрямок, в якому модель повинна бути оптимізована. У цьому розділі ми розглянемо різноманітні функції втрат, які широко використовуються для навчання трансформерів, та їх специфіку.

Крос-ентропійна втрата (Cross-Entropy Loss) є однією з найпоширеніших функцій втрат для задач класифікації, включаючи багато прикладів використання в трансформерах. Вона вимірює відстань між розподілом ймовірностей, передбаченим моделлю, і справжніми мітками класів.

Формально, для двох класів крос-ентропійна втрата визначається як:

$$L = -\sum_{i=1}^{N} y_i \log(\hat{y}_i),$$

де y_i — справжня мітка (1 для позитивного класу і 0 для негативного), а $\hat{y}_i$ — передбачене значення ймовірності. Крос-ентропійна втрата особливо ефективна в задачах, таких як машинний переклад та генерація тексту, де модель повинна генерувати ймовірності для кожного слова в словнику.

Втрата категориальної крос-ентропії (Categorical Cross-Entropy Loss) використовується для багатокласової класифікації. Вона є розширенням основної крос-ентропійної втрати і враховує множину класів. Формула для категориальної крос-ентропійної втрати виглядає так:

$$L = -\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij} \log(\hat{y}_{ij})$$

де C — кількість класів, y_{ij}— справжня мітка для класу j (1, якщо спостереження належить класу, і 0 в іншому випадку), а $\hat{y}_{ij}$ — передбачена ймовірність для класу j. Ця функція втрат широко використовується в задачах, що вимагають класифікації тексту.

Середньоквадратична помилки (Mean Squared Error Loss, MSE) використовується в задачах регресії, де метою є прогнозування числових

значень. Вона вимірює середню квадратичну різницю між передбаченими і справжніми значеннями:

$$L = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2.$$

Хоча трансформери в основному застосовуються для класифікації, MSE може бути корисною в задачах, пов'язаних із регресією, наприклад, в прогнозуванні числових значень у фінансових або наукових даних.

Втрата для послідовного навчання (Sequence Loss). Трансформери часто використовуються для генерації послідовностей, наприклад, у задачах машинного перекладу. У таких випадках функція втрат може бути адаптована для врахування контексту та порядку слів у згенерованій послідовності. Втрата для послідовного навчання може включати використання механізмів, які враховують як правильність передбачення, так і структуру виходу [20].

Втрата контрастивного навчання (Contrastive Loss) використовується для навчання моделей, які повинні розрізняти позитивні та негативні пари прикладів. У трансформерах цей метод може бути ефективним для задач, пов’язаних із семантичним розумінням. Формально, контрастивна втрата може бути описана як:

$$L = \frac{1}{2N}\sum_{i=1}^{N}(y_i d^2 + (1 - y_i)\ \ max(0, m - d)^2),$$

де d – відстань між векторами, y_i – мітка пари (1 для позитивних, 0 для негативних), а m – маржа. Ця функція втрат дозволяє моделі вчитися на відмінностях між схожими і різними прикладами, що є важливим для задач, що вимагають глибокого семантичного аналізу.

Спеціальні функції втрат відіграють важливу роль у навчанні трансформерів, забезпечуючи необхідну гнучкість і адаптивність для вирішення різноманітних задач. Вибір правильної функції втрат може суттєво вплинути на продуктивність моделі. Розуміння різних функцій втрат і їх специфіки є важливим для досягнення кращих результатів у задачах обробки природної мови, комп'ютерного зору та інших сферах, де використовуються трансформери. Дослідження в цій області можуть допомогти в розробці нових

функцій втрат, що підвищать ефективність трансформерів у складних задачах обробки даних.

2.5.5. Розподілене навчання

Трансформери, завдяки своїй потужності в обробці послідовностей, стали основою багатьох сучасних моделей штучного інтелекту. Однак, їхня складна архітектура та велика кількість параметрів вимагають значних обчислювальних ресурсів. Розподілене навчання є ключовим підходом для подолання цього обмеження, дозволяючи розподілити обчислення по декількох пристроях або вузлах кластера. Це дозволяє значно зменшити час навчання і підвищити ефективність обробки великих обсягів даних. Основні підходи до розподіленого навчання включають паралелізм даних і паралелізм моделі.

Паралелізм даних. Метод паралелізму даних передбачає копіювання моделі на кілька пристроїв, кожен з яких працює з підмножиною навчальних даних. Після кожної ітерації градієнти з усіх пристроїв об'єднуються, що дозволяє оновлювати ваги моделі. Цей підхід забезпечує ефективне використання обчислювальних ресурсів і значно скорочує час навчання.

Паралелізм моделі. У методі паралелізму моделі різні частини (шари) трансформера розподіляються між кількома пристроями. Це корисно, коли модель занадто велика для одного пристрою. Наприклад, один GPU може обробляти шари самої уваги, в той час як інший — шари feed-forward. Це дозволяє зменшити вимоги до пам’яті та підвищити загальну продуктивність.

Для реалізації розподіленого навчання трансформерів використовуються різні *технології та фреймворки*:

1) *TensorFlow* надає можливості для реалізації розподіленого навчання через TensorFlow Distributed. Цей фреймворк дозволяє легко реалізувати як паралелізм даних, так і моделі, пропонуючи прості API для синхронізації градієнтів.

2) *PyTorch* має модулі для розподіленого навчання, такі як torch.distributed, які забезпечують гнучкість у реалізації паралелізму. PyTorch також підтримує динамічний граф обчислень, що робить його популярним вибором серед дослідників.

3) *Horovod* – це бібліотека, яка спрощує реалізацію розподіленого навчання для TensorFlow і PyTorch. Вона використовує алгоритм Ring-AllReduce для швидкої синхронізації градієнтів, що зменшує затримки при навчанні.

Розподілене навчання має кілька ключових *переваг*:

- Швидкість**:** значне скорочення часу, необхідного для навчання великих моделей, завдяки паралельній обробці.
- Масштабованість**:** здатність обробляти великі набори даних, що не вміщуються в пам’ять одного пристрою.
- Ефективність**:** зменшення витрат на обчислення та енергоспоживання за рахунок оптимального використання ресурсів.

Попри численні переваги, розподілене навчання також стикається з певними *викликами*:

- Синхронізація**:** необхідність ефективної синхронізації ваг і градієнтів між пристроями може призводити до затримок, особливо при збільшенні кількості пристроїв.
- Баланс навантаження**:** оптимальний розподіл даних та моделі є критично важливим для уникнення перевантаження окремих пристроїв.
- Управління пам’яттю**:** збільшення кількості пристроїв також підвищує вимоги до пам’яті, що може бути проблемою при навчанні великих моделей.

Висновки за розділом 2

Трансформери: це революційна архітектура нейронних мереж, яка докорінно змінила підхід до обробки послідовностей даних, особливо в галузі обробки природної мови. Їхньою ключовою особливістю є механізм уваги,

який дозволяє моделі фокусуватися на найбільш релевантних частинах вхідних даних, що значно покращує якість результатів.

Ключові переваги трансформерів:

Паралельність обчислень: завдяки механізму уваги, трансформери можуть обробляти всі елементи послідовності одночасно, що значно прискорює навчання і дозволяє ефективно використовувати сучасні обчислювальні ресурси.

Здатність захоплювати довгострокові залежності: трансформери можуть ефективно обробляти залежності між віддаленими елементами в послідовності, що є важливим для багатьох завдань обробки природної мови.

Гнучкість та універсальність: трансформери можуть бути застосовані до широкого спектра завдань, включаючи машинний переклад, генерацію тексту, аналіз настроїв, розпізнавання мови та багато інших.

РОЗДІЛ 3.
РОЗРОБКА НЕЙРОМЕРЕЖЕВОЇ МОДЕЛІ МАШИННОГО ПЕРЕКЛАДУ

Цей розділ присвячений практичній реалізації системи машинного перекладу на основі нейронних мереж Transformers. Буде описаний процес підготовки даних, тренування моделі та її оцінки. Для реалізації системи перекладу був обраний хмарний сервіс Google Colaboratory. Ця платформа забезпечує безкоштовний доступ до потужних GPU, що дозволило ефективно тренувати нашу модель. Крім того, зручне середовище розробки та інтеграція з Google Drive значно спростили роботу. Можливість візуалізації даних допомогла краще зрозуміти процес навчання і оцінити якість отриманої моделі."

3.1 Дані для навчання нейромережі

Моделювання мови полягає у вивченні розподілу послідовностей токенів у великих корпусах тексту. Чим більше репрезентативних прикладів використовується, тим вищою буде якість моделі.

Більшість досліджень у сфері обробки природної мови зосереджувалися на англійській мові, що призвело до розвитку численних датасетів і появи багатьох відкритих англомовних джерел даних. Серед них — вручну анотовані семантичні структури, такі як Penn Tree Bank, а також нерозмічений півекзабайтний дамп Інтернету, відомий як Common Crawl.

Сучасні методи, разом із суттєвим зростанням обчислювальних потужностей за останнє десятиліття, продемонстрували високу ефективність використання нерозмічених даних для навчання великих мовних моделей. Це дозволяє зменшити потребу у значних ресурсах для ручної анотації великих обсягів тексту [13].

Доступні такі корпуси української мови:

- **Дамп Вікіпедії** — містить близько 2 ГБ нерозміченого тексту з україномовних матеріалів проекту. Більшість текстів написана в науковому стилі і охоплює різноманітні теми, такі як природничі науки, історія, мистецтво тощо.
- **Корпус законів та правових актів** — складається з понад 560 МБ юридичного тексту, доступного у токенізованих і лематизованих версіях.
- **Браунський корпус української мови** — створений як аналог Brown Corpus, має обсяг 1 мільйон слововживань. Включає тексти, які репрезентують сучасне використання української мови в електронному середовищі.
- **Корпус UberText** — містить 6 ГБ тексту з 11 онлайн-українських періодичних джерел і Вікіпедії. Для уникнення юридичних обмежень тексти розбиті на речення та перемішані.
- **OSCAR Corpus UK** — найбільший набір текстів українською мовою, зібраний за допомогою Common Crawl та автоматичного визначення мови. Обсяг корпусу складає 28 ГБ, що відповідає понад 2 мільярдам слововживань.

У даній роботі був розглянутий датасет англійсько-українських пар речень з https://www.manythings.org/anki/.

Датасет містить двомовні пари речень, в яких англійські фрази перекладені українською мовою. Ці дані були взяті з проекту Tatoeba і підготовлені для використання в додатках, таких як Anki, що дозволяє ефективно вивчати мови за допомогою флеш-карток.

Кожен запис у датасеті представлений у табульованому форматі, що складається з трьох колонок:

1. *Англійська фраза* — оригінальний текст англійською мовою.
2. *Український переклад* — відповідний переклад цієї фрази українською.
3. *Атрибуція* — інформація про авторство речення, що включає ліцензію (CC-BY 2.0) та ідентифікатори речень на сайті Tatoeba.

Приклад рядка з датасету:

Go. Йди. CC-BY 2.0 (France) Attribution: tatoeba.org #2877272 (CM) & #6584257 (deniko)

Тобто за структурою:

- Go. – англійське речення;
- Йди. – український переклад;
- CC-BY 2.0 (France) Attribution: tatoeba.org #2877272 (CM) & #6584257 (deniko) – метадані, що вказують на ліцензію, джерело та ідентифікатори авторів перекладу (CM та deniko).

3.2 Попередня обробка даних

Попередня обробка даних є критично важливим етапом у створенні моделей машинного навчання, особливо в задачах обробки природної мови (NLP). У даній роботі для навчання моделі перекладу з англійської на українську було виконано кілька ключових етапів попередньої обробки даних.

Першим кроком до виконання завдання було визначення необхідних бібліотек та моделей. На рис 3.1 наведено список ключових інструментів, які були використані.

```
[ ]  import os
     import pathlib
     import random
     import string
     import re
     import numpy as np
     import tensorflow as tf
     from tensorflow import keras
     from tensorflow.keras import layers
     from tensorflow.keras.layers import TextVectorization
     import nltk
     from nltk.stem import WordNetLemmatizer
     import tensorflow.strings as tf_strings

     # Set Keras backend
     os.environ["KERAS_BACKEND"] = "tensorflow"
```

Рисунок 3.1 – Імпортування необхідних програмних засобів

Завантаження даних

Дані були завантажені з відкритого джерела. Набір даних містить пари англійських та українських речень, що дозволяє навчити модель на основі реальних прикладів. Завантаження даних реалізовано за допомогою команд wget та unzip, які автоматично отримують архів з даними та розпаковують його, як показано на рис. 3.2.

Рисунок 3.2 – Завантаження даних

Після розпакування даних, відбувається їх читання з файлу ukr.txt. Кожен рядок файлу містить англійське речення та відповідне йому українське речення, розділені символом табуляції. Дані зчитуються та зберігаються у вигляді списку кортежів (рис. 3.3).

```
text_file = pathlib.Path(".") / "ukr.txt"
# Читання даних з файлу
with open(text_file) as f:
    lines = f.read().split("\n")[:-1]

text_pairs = []
for line in lines:
    parts = line.split("\t")
    eng = parts[0]
    ukr = parts[1]
    ukr = ukr
    text_pairs.append((eng, ukr))
```

Рисунок 3.3 – Читання даних

Розбиття даних

Для оцінки якості моделі важливо створити набір даних, на якому буде перевірятися її ефективність. Ключовим аспектом вибору такого набору є його репрезентативність щодо загального розподілу даних у відповідній області. Крім того, критично важливо, щоб тестовий набір не перетинався з навчальним, оскільки це може призвести до того, що модель «запам'ятає» навчальні дані і, як наслідок, покаже неадекватні результати на тестовій вибірці.

Існує кілька способів розбиття даних.

- Випадкове розбиття (random train/test split) — цей метод базується на припущенні, що всі дані належать до одного розподілу, тому випадкові вибірки також матимуть схожий розподіл.
- Розбиття на тренувальну, валідаційну та тестові вибірки (train/validation/test split) — цей підхід аналогічний попередньому, але передбачає наявність додаткової валідаційної вибірки для підбору гіперпараметрів. Фінальна оцінка якості моделі проводиться після навчання з оптимальними параметрами, підібраними на валідаційній вибірці, на всіх даних, крім тестових. Це дозволяє достовірніше стверджувати, що модель

справді краща, а не просто обирати моделі, яким пощастило показати кращі результати на тестовій вибірці.

• Крос-валідація (cross-validation) — цей підхід є узагальненням розбиття на тренувальну та валідаційну вибірки, що передбачає багаторазове використання розбиття (наприклад, через k-fold split). При достатній кількості ітерацій це дозволяє провести статистичний аналіз якості гіперпараметрів.

Для оцінки якості моделі, дані були випадково розбиті на три набори: навчальний, валідаційний та тестовий (рис. 3.4). Таке розбиття дозволяє запобігти перенавчанню моделі та перевірити її здатність узагальнювати нову інформацію.

Оскільки великі розміри датасету значно сповільнюють процес навчання, Вибір розміру цих вибірок залежить від конкретної задачі та обсягу наявних даних.

```
random.shuffle(text_pairs)
num_val_samples = int(0.1 * len(text_pairs))
num_train_samples = len(text_pairs) - 2 * num_val_samples
train_pairs = text_pairs[:num_train_samples]
val_pairs = text_pairs[num_train_samples : num_train_samples + num_val_samples]
test_pairs = text_pairs[num_train_samples + num_val_samples :]

print(f"{len(text_pairs)} total pairs")
print(f"{len(train_pairs)} training pairs")
print(f"{len(val_pairs)} validation pairs")
print(f"{len(test_pairs)} test pairs")
```

```
159432 total pairs
127546 training pairs
15943 validation pairs
15943 test pairs
```

Рисунок 3.4 – Розбиття даних на три набори

Препроцесинг даних

Препроцесинг даних — це важливий етап в обробці текстової інформації, який полягає у підготовці сирих даних до подальшого аналізу або навчання моделей машинного навчання. На цьому етапі дані очищуються,

нормалізуються та формуються у зручний для використання формат. В даній роботі препроцесинг текстових даних, включає в себе такі кроки:

Попередня обробка тексту – це набір методів, спрямованих на приведення текстових даних до формату, придатного для подальшого аналізу комп'ютерними системами. Для задач машинного перекладу, попередня обробка включає такі етапи:

- токенізацію – розбиття тексту на окремі слова або підслова (токени).
- нормалізацію - приведення слів до стандартної форми (наприклад, приведення до нижнього регістру, видалення пунктуації).
- лематизацію – зведення слів до їхньої леми – базової форми слова.
- векторизацію – перетворення текстових даних у числові вектори, які можуть бути оброблені моделями машинного навчання.

Розглянемо детальніше, як реалізована попередня обробка тексту в даній роботі.

Завантаження ресурсів NLTK. Спочатку була завантажена база даних WordNet, яка використовується для лематизації. Лематизація переводить слова до їхньої базової форми (наприклад, "бігати" стає "бігти").

Створення лематизатора WordNet. потім створений об'єкт класу WordNetLemmatizer, який буде використовуватися для лематизації тексту пізніше.

Функції попередньої обробки:

- custom_standardization(text): Ця функція переводить текст до нижнього регістру (text.lower()). Це поширений крок попередньої обробки, який покращує узгодженість даних та усуває проблеми з чутливістю до регістру.
- lemmatize_text(text): Ця функція розділяє текст на окремі слова (text.split()), потім використовує об'єкт lemmatizer для лематизації кожного слова. Нарешті, вона знову об'єднує лематизовані слова в речення (' '.join(lemmas)).

Функція preprocess_text(text). Ця функція приймає текст як вхідні дані та виконує такі дії:

- перевіряє, чи є вхідні дані тензором TensorFlow (спеціальна структура даних). Якщо так, він перетворює його на звичайний рядок і декодує його в кодування UTF-8;
- викликає custom_standardization для переведення тексту до нижнього регістру;
- викликає lemmatize_text для лематизації тексту;
- повертає попередньо оброблений текст.

Підготовка текстів для векторізації: ця частина коду витягує англійську та українську частини тексту з пар тренувальних даних (train_pairs). Припускається, що кожна пара містить англійське речення та його відповідний український переклад.

Параметри векторізації:

- strip_chars: Ця змінна визначає символи, які потрібно видалити з тексту під час стандартизації. Вона включає знаки пунктуації та деякі додаткові символи, наприклад "¿".
- vocab_size: Ця змінна визначає максимальну кількість слів, які будуть враховуватися у словнику. Тільки vocab_size найчастіших слів будуть використовуватися для векторізації.
- sequence_length: Ця змінна встановлює максимальну довжину (кількість слів) для кожної послідовності після векторізації. Речення, довші за цю довжину, будуть обрізані, а коротші - доповнені нулями.
- batch_size: Ця змінна визначає кількість пар текстів, які будуть оброблятися разом під час навчання.

Функція custom_standardization_tf(input_string): ця функція є спеціалізованою версією custom_standardization для TensorFlow. Вона виконує ту саму задачу (переведення до нижнього регістру), але використовує спеціальні операції TensorFlow.

Після того як виконали попередню обробку тексту та векторизацію далі ми підготуємо наші набори даних.

На кожному етапі навчання модель намагатиметься передбачити цільові слова N+1 та наступні, використовуючи вихідне речення і цільові слова від 0 до N.

Таким чином, навчальний набір даних представляє собою кортеж (inputs, targets), де:

- **inputs** – це словник з ключами encoder_inputs та decoder_inputs. encoder_inputs містить векторизоване вихідне речення, а decoder_inputs є цільовим реченням на даний момент, тобто словами від 0 до N, які використовуються для прогнозування слова N+1 та наступних у цільовому реченні.
- **target** – це цільове речення, зміщене на один крок: воно надає наступні слова в цільовому реченні, які модель спробує передбачити.

3.3 Побудова моделі

У роботі реалізовано архітектуру трансформера, яка складається з трьох основних компонентів: кодувальника (TransformerEncoder), декодувальника (TransformerDecoder) та шару позиційного векторизування (PositionalEmbedding). Ця архітектура використовується для задач обробки природної мови, таких як машинний переклад та генерація тексту.

1) TransformerEncoder

Клас TransformerEncoder реалізує кодувальник, який відповідає за обробку вхідних даних та генерацію їх контекстуального представлення.

Параметри:

- embed_dim: розмір векторів, що представляють токени;
- dense_dim: розмір проміжного шару нейронної мережі;
- num_heads: кількість голів у механізмі багатоголового уваги;
- kernel_regularizer: регуляризатор для шарів Dense (опціонально).

Методи:

- call: основний метод, що виконує обробку вхідних даних. Він спочатку перевіряє наявність маски, потім використовує механізм уваги, після чого нормалізує результати та пропускає їх через два Dense-шари;
- get_config: повертає конфігурацію класу для подальшого збереження та відновлення.

2) PositionalEmbedding

Клас PositionalEmbedding відповідає за додавання позиційної інформації до векторизованих токенів, що дозволяє моделі враховувати порядок слів у реченні.

Параметри:

- sequence_length: максимальна довжина вхідної послідовності;
- vocab_size: розмір словника, що використовується для векторизації токенів;
- embed_dim: розмір векторів векторизації.

Методи:

- call: обчислює векторизовані токени та позиції, а потім об'єднує їх, створюючи фінальне представлення.
- compute_mask: повертає маску, що вказує на ненульові токени. Якщо маски немає, повертає None.
- get_config: повертає конфігурацію класу.

3)TransformerDecoder

Клас TransformerDecoder реалізує декодувальник, який генерує вихідні дані на основі кодування.

Параметри:

- embed_dim: розмір векторів векторизації;
- latent_dim: розмір проміжного шару в декодувальнику;
- num_heads: кількість голів у механізмі багатоголового уваги.

Методи:

- call: виконує обробку вхідних даних, використовуючи два механізми уваги: перший — для самостійної уваги, другий — для уваги до виходу кодувальника. Результати нормалізуються та проходять через Dense-шар.
- get_causal_attention_mask: генерує маску, що забезпечує причинність у механізмі уваги, дозволяючи декодувальнику бачити лише попередні токени.
- get_config: пвертає конфігурацію класу.

Наступним кроком є *створення наскрізної моделі*, яка об'єднує кодер та декодер в єдину архітектуру. Для цього спочатку визначаються гіперпараметри моделі: розмірність вбудовувань, розмірність прихованого простору та кількість голів уваги.

Далі створюються вхідні тензори для кодера та декодера. Кодер приймає на вхід послідовність токенів, які перетворюються на векторні представлення з використанням позиційного вбудовування. Отримані векторні представлення подаються на вхід трансформерного енкодера, який обробляє їх за допомогою механізму уваги та багатошарової перцептронної мережі.

Декодер також приймає на вхід послідовність токенів, які перетворюються на векторні представлення. Крім того, він отримує на вхід векторне представлення, отримане з кодера. Декодер використовує механізм маскованої самоуваги для обробки власної послідовності та механізм кодувально-декодувальної уваги для отримання інформації з кодера. Після цього, декодер генерує вихідну послідовність токенів, використовуючи розподіл ймовірностей, отриманий з лінійного шару з функцією активації softmax.

Наскрізна модель об'єднує кодер та декодер, приймаючи на вхід два речення (оригінальне та початок перекладу) і генеруючи повний переклад. Така архітектура дозволяє моделі ефективно навчатися на великих корпусах даних і досягати високої якості перекладу.

3.4. Тренування моделі

У роботі реалізується етап навчання моделі трансформера, що включає в себе налаштування параметрів, компіляцію моделі та виконання процесу навчання. Спочатку встановлюється кількість епох навчання – 40, що дозволяє моделі досягти кращої збіжності. Для перегляду структури моделі викликається метод summary(), що надає інформацію про кількість шарів, параметрів та архітектуру трансформера. Модель компілюється з використанням оптимізатора RMSprop, налаштованого на малу швидкість навчання (0.0001), а також функції втрат sparse_categorical_crossentropy, яка підходить для задачі класифікації з кількома класами. Також відслідковується значення показника точності ("accuracy") під час навчання.

Для запобігання перенавчанню моделі впроваджується механізм раннього зупинення (Early Stopping). Цей механізм моніторить значення втрат на валідаційних даних, зупиняючи навчання, якщо не спостерігається покращення протягом 5 епох, та відновлює найкращі ваги моделі. Процес навчання виконується за допомогою методу fit(), де передаються навчальний і валідаційний набори даних, а також вказується використовуваний відклик для раннього зупинення. Таким чином, код забезпечує ефективний та контрольований процес навчання моделі трансформера.

3.5 Тестування моделей

У дослідженні було проведено тестування двох моделей машинного перекладу з англійської на українську мову, реалізованих на основі архітектури Transformer.

Перша модель:

- використовувалася традиційна токенізація, яка розбиває текст на слова;
- для покращення якості обробки тексту застосовувалась лематизація, що дозволяє привести слова до їх базової (лематичної) форми;

• для реалізації лематизації у Python була використана бібліотека NLTK (Natural Language Toolkit), а саме клас WordNetLemmatizer.

Друга модель:

• для токенізації було обрано більш сучасний підхід — WordPiece токенізація, яка дозволяє розбивати слова на менші частини, відомі як підслова;

• цей метод також включає використання біграмів, що дозволяє моделі краще розуміти контекст слів у реченні.

• для реалізації WordPiece токенізації у Python використовувалась бібліотека TensorFlow, зокрема клас TextVectorization.

Перша модель була навчена протягом 40 епох, в результаті чого були отримані доволі високі показники якості на тренувальному і валідаційному наборі (рис. 3.5).

```
Epoch 37/40
1993/1993 ━━━━━━━━━━━━━━━━━━━━ 103s 52ms/step - accuracy: 0.9866 - loss: 0.1543 - val_accuracy: 0.9886 - val_loss: 0.1836
Epoch 38/40
1993/1993 ━━━━━━━━━━━━━━━━━━━━ 103s 52ms/step - accuracy: 0.9870 - loss: 0.1498 - val_accuracy: 0.9890 - val_loss: 0.1797
Epoch 39/40
1993/1993 ━━━━━━━━━━━━━━━━━━━━ 102s 51ms/step - accuracy: 0.9875 - loss: 0.1453 - val_accuracy: 0.9893 - val_loss: 0.1761
Epoch 40/40
1993/1993 ━━━━━━━━━━━━━━━━━━━━ 103s 52ms/step - accuracy: 0.9879 - loss: 0.1409 - val_accuracy: 0.9897 - val_loss: 0.1720
```

Рисунок 3.5 – Результат навчання першої моделі

У результаті модель №1 демонструє хороші результати на тестовому наборі, генеруючи адекватні переклади.

Друга модель була навчена протягом 70 епох, та були отримані високі показники якості на тренувальному і валідаційному наборі (рис. 3.6).

```
Epoch 67/70
997/997 ━━━━━━━━━━━━━━━━━━━━ 75s 75ms/step - accuracy: 0.9961 - loss: 0.0331 - val_accuracy: 0.9941 - val_loss: 0.0786
Epoch 68/70
997/997 ━━━━━━━━━━━━━━━━━━━━ 76s 76ms/step - accuracy: 0.9963 - loss: 0.0321 - val_accuracy: 0.9943 - val_loss: 0.0781
Epoch 69/70
997/997 ━━━━━━━━━━━━━━━━━━━━ 75s 75ms/step - accuracy: 0.9965 - loss: 0.0314 - val_accuracy: 0.9944 - val_loss: 0.0776
Epoch 70/70
997/997 ━━━━━━━━━━━━━━━━━━━━ 75s 75ms/step - accuracy: 0.9967 - loss: 0.0306 - val_accuracy: 0.9945 - val_loss: 0.0768
```

Рисунок 3.6 – Результат навчання другої моделі

У результаті модель також демонструє хороші результати на тестовому наборі, але якість перекладів трохи нижча, ніж у першої моделі.

Аналіз графіків навчання другої моделі показує, що модель не перенавчена, оскільки втрати та точність на тренувальному та валідаційному наборах змінюються подібним чином (рис. 3.8).

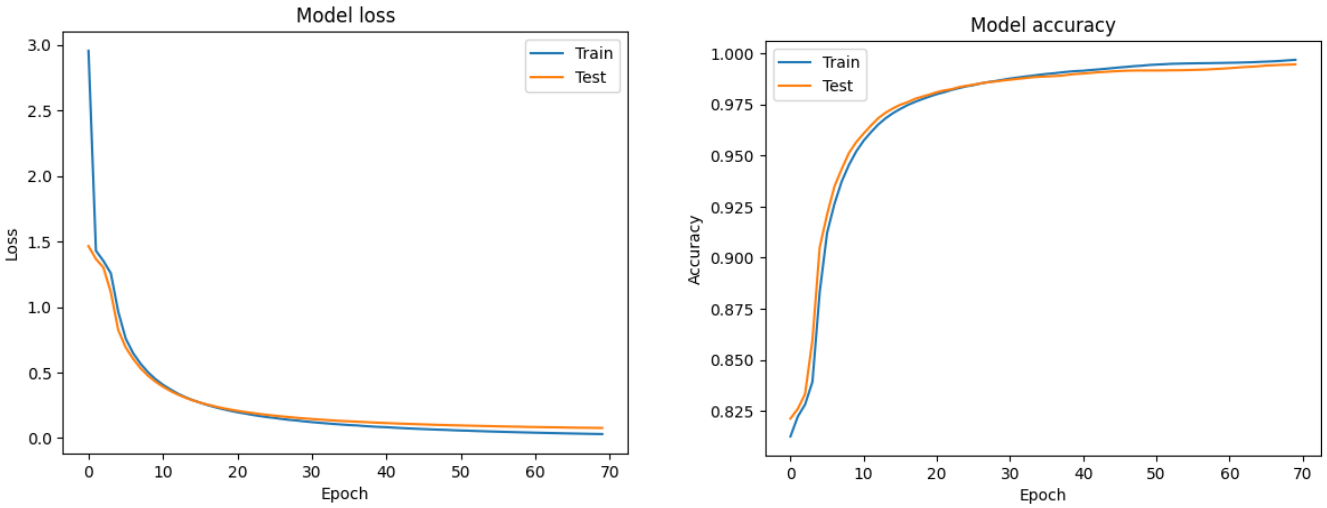

Рисунок 3.8 – Криві втрат та точності під час навчання для другої моделі

Обидві моделі показали добрі результати, але перша модель з словниковим методом токенізації продемонструвала трохи кращу якість перекладів. Це може бути пов'язано з тим, що словниковий метод краще зберігає семантичну інформацію слів, що важливо для адекватного перекладу.

Також реалізовано метод, який приймає на вхід речення англійською мовою, оброблює його за допомогою навчених моделей та надає його переклад українською мовою (рис. 3.9). В цілому, результати дослідження підтверджують, що архітектура Transformer є ефективним інструментом для машинного перекладу.

```
def decode_sequence(input_sentence):
    tokenized_input_sentence = eng_vectorization([input_sentence])
    decoded_sentence = "[start]"
    for i in range(max_decoded_sentence_length):
        tokenized_target_sentence = ukr_vectorization([decoded_sentence])[:, :-1]
        predictions = transformer([tokenized_input_sentence, tokenized_target_sentence])

        sampled_token_index = ops.convert_to_numpy(
            ops.argmax(predictions[0, i, :])
        ).item(0)
        sampled_token = ukr_index_lookup[sampled_token_index]
        decoded_sentence += " " + sampled_token

        if sampled_token == "[end]":
            break
    return decoded_sentence


while True:
    input_sentence = input("Введіть речення англійською мовою (або 'q' для виходу): ")
    if input_sentence == 'q':
        break
    translated = decode_sequence(input_sentence)
    print("Переклад:", translated)
```

Рисунок 3.9 – Метод для перекладу речень за допомогою навчених моделей

Результат перекладу речень за допомогою першої моделі показаний на рис. 3.10.

```
Введіть текст на англійській мові (або 'q' для завершення): I visited Australia
Переклад: Я відвідав Австралію.
Введіть текст на англійській мові (або 'q' для завершення): I think that Tom didn't tell us the truth.
Переклад: Я думаю, Том сказав нам неправду.
Введіть текст на англійській мові (або 'q' для завершення): If you see a mistake, then please correct it.
Переклад: Якщо ти бачиш помилку, виправ її, будь ласка.
Введіть текст на англійській мові (або 'q' для завершення): The decision has not yet been
Переклад: Рішення ще не прийнято.
Введіть текст на англійській мові (або 'q' для завершення): turned on my computer
Переклад: Я ввімкнула свій комп'ютер.
Введіть текст на англійській мові (або 'q' для завершення): don't kill me
Переклад: Не цькуйте мене.
Введіть текст на англійській мові (або 'q' для завершення): q
```

Рисунок 3.10 – Приклад перекладу речень на основі першої моделі

Результат перекладу речень за допомогою другої моделі показаний на рис. 3.11.

```
Введіть текст на англійській мові (або 'q' для завершення): I usually go home soon after five o'clock.
Переклад: Я, як правило, йду додому відразу по п'ятій.
Введіть текст на англійській мові (або 'q' для завершення): Would you like me to do that for you
Переклад: Ти б хотів, щоб я це зробив для тебе?
Введіть текст на англійській мові (або 'q' для завершення): She was educated
Переклад: Я був переможений.
Введіть текст на англійській мові (або 'q' для завершення): I really appreciate
Переклад: Я дуже ціную, що ти прийшла.
Введіть текст на англійській мові (або 'q' для завершення): Don't stand up
Переклад: Не вставайте.
Введіть текст на англійській мові (або 'q' для завершення): q
```

Рисунок 3.11 – Приклад перекладу речень на основі другої моделі

Висновки за розділом 3

У розділі було розглянуто практичну реалізацію моделей машинного перекладу на основі нейронних мереж Transformers. Було описано процес підготовки даних, тренування моделі та її оцінки. Для реалізації моделей перекладу було обрано хмарний сервіс Google Colaboratory, який забезпечує безкоштовний доступ до потужних GPU та зручне середовище розробки.

Було детально розглянуто процес розробки нейромережевої моделі машинного перекладу. Використання великого корпусу паралельних текстів забезпечило основу для ефективного навчання нейромережі. Процес попередньої обробки, включаючи токенізацію та векторизацію тексту, суттєво вплинув на продуктивність моделі.

Розроблена архітектура нейронної мережі на основі моделі трансформер продемонструвала високу ефективність у навчанні. Процес тренування з параметрами оптимізації, такими як Adam та регуляризація, забезпечив стабільне зростання точності перекладу. Результати тестування показали, що модель здатна досягати високих показників якості перекладу. Розроблена модель може бути інтегрована в різні програмні рішення для автоматизації перекладу, що підкреслює її значення в умовах глобалізації.

ВИСНОВКИ

Завдання, яке розглядалося у кваліфікаційній роботі, полягало у розробці моделі автоматизованого перекладу текстів на основі алгоритмів машинного навчання. Робота передбачала аналіз підходів до автоматизованого перекладу, розробку алгоритмів, а також їхню імплементацію в програмному забезпеченні.

Основною метою було створення ефективної моделі, яка б могла перекладати текстові дані, враховуючи різні аспекти машинного навчання. Для цього був проведений детальний літературний огляд, вивчені існуючі методи машинного перекладу. І для реалізації були обрані підходи із використанням штучних нейронних мереж, а саме мереж Трансформерів.

У роботі були досягнуті такі результати:

– визначено основні аспекти машинного перекладу, включаючи його цілі та завдання;

– досліджено різні підходи до автоматизованого перекладу текстових даних, зокрема, методи, засновані на правилах, статистичних даних, прикладах та нейронних мережах;

– виявлено переваги та недоліки кожного з підходів, що дозволяє оцінити їхню ефективність у сучасних умовах;

– обговорено актуальні проблеми та виклики, з якими стикається машинний переклад, зокрема, багатозначність, культурні особливості та потребу в великих обсягах даних для навчання систем;

– побудовані і протестовані моделі машинного перекладу на основі Трансформерів, які показали високу ефективність у перекладі коротких речень;

– висвітлено майбутні напрямки розвитку технологій машинного перекладу, акцентуючи увагу на можливостях використання гібридних та нейронних систем.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. History of Machine Translation. [Електронний ресурс] – режим доступу: URL: https://translatorstudio.co.uk/machine-translation-history// (дата звернення: 10.09.2024)
2. Keras, Natural Language Processing [Електронний ресурс] – режим доступу: URL: https://keras.io/examples/nlp/ (дата звернення: 10.09.2024)
3. Thang Luong, Kyunghyun Cho, Christopher Manning. Neural Machine Translation. [Електронний ресурс] – режим доступу: URL: https://aclanthology.org/P16-5005/ (дата звернення: 12.09.2024)
4. Ilyas Cicekli, Kemal AltintasA Machine Translation System Between a Pair of Closely Related Languages. [Електронний ресурс] – режим доступу: URL: https://www.researchgate.net/publication/238160330_A_Machine_Translation_System_Between_a_Pair_of_Closely_Related_Languages (дата звернення: 12.09.2024)
5. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin. Attention Is All You Need. arXiv: [Електронний ресурс] – режим доступу: URL: https://arxiv.org/abs/1706.03762 (дата звернення: 12.09.2024).
6. Що таке нейронний машинний переклад? [Електронний ресурс] – режим доступу: URL: https://www.linguise.com/uk/%D0%B1%D0%BB%D0%BE%D0%B3/%D0%BA%D0%B5%D1%80%D1%96%D0%B2%D0%BD%D0%B8%D1%86%D1%82%D0%B2%D0%BE/%D1%89%D0%BE-%D1%82%D0%B0%D0%BA%D0%B5-%D0%BD%D0%B5%D0%B9%D1%80%D0%BE%D0%BD%D0%BD%D0%B8%D0%B9-%D0%BC%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D0%B8%D0%B9-%D0%BF%D0%B5%D1%80%D0%B5%D0%BA%D0%BB%D0%B0%D0%B4/ (дата звернення: 12.09.2024).

7. Arabic-english translator of specific noun phrase with transfer-based approach. Namiq Sultan Abdullah [Електронний ресурс] – режим доступу: URL: https://www.researchgate.net/publication/330161752_ARABIC-ENGLISH_TRANSLATOR_OF_SPECIFIC_NOUN_PHRASE_WITH_TRANSFER-BASED_APPROACH (дата звернення: 13.09.2024).
8. Methods for Leveraging Lexical Information in SMT. Karan Singla [Електронний ресурс] – режим доступу: URL https://www.researchgate.net/publication/279181014_Methods_for_Leveraging_Lexical_Information_in_SMT (дата звернення: 13.09.2024).
9. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification, автор Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. [Електронний ресурс] – режим доступу: URL: https://arxiv.org/abs/1502.01852 (дата звернення: 13.09.2024).
10. Understanding the difficulty of training deep feedforward neural networks. Автор Xavier Glorot, Yoshua Bengio. [Електронний ресурс] – режим доступу: URL: https://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf (дата звернення: 15.09.2024).
11. "Deep Learning" – Ian Goodfellow, Yoshua Bengio, і Aaron Courville [Електронний ресурс] – режим доступу: URL: https://www.deeplearningbook.org/ (дата звернення: 15.09.2024).
12. Juan Ramos, Using TF-IDF to Determine Word Relevance in Document Queries. [Електронний ресурс] – режим доступу: URL: https://www.researchgate.net/publication/228818851_Using_TF-IDF_to_determine_word_relevance_in_document_queries (дата звернення: 15.09.2024).
13. P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, “Enriching Word Vectors with Subword Information,” [Електронний ресурс] – режим доступу: URL: https://arxiv.org/abs/1607.04606 (дата звернення: 20.09.2024).

14. P. Gage, “A New Algorithm for Data Compression.” [Електронний ресурс] – режим доступу: URL: http://www.pennelynn.com/Documents/CUJ/HTML/94HTML/19940045.HTM (дата звернення: 20.09.2024).
15. C. Wang, K. Cho, and J. Gu, Neural Machine Translation with Byte-Level Subwords. [Електронний ресурс] – режим доступу: URL: https://doi.org/10.48550/arXiv.1909.03341 (дата звернення: 1.10.2024).
16. T. Mikolov, K. Chen, G. Corrado, and J. Dean, Efficient Estimation of Word Representations in Vector Space. [Електронний ресурс] – режим доступу: URL: https://doi.org/10.48550/arXiv.1301.3781(дата звернення: 1.10.2024).
17. R. Kulshrestha, NLP 101: Word2vec - skip-gram and CBOW. Towards Data Science, 2020. [Електронний ресурс] – режим доступу: URL:https://towardsdatascience.com/nlp-101-word2vec-skip-gram-and-cbow-93512ee24314 (дата звернення: 5.10.2024).
18. J. B. Lovins, “Development of a stemming algorithm,” Mech. Transl. Comput. Linguistics, vol. 11, pp. 22–31, 1968. [Електронний ресурс] – режим доступу: URL: https://aclanthology.org/www.mt-archive.info/MT-1968-Lovins.pdf (дата звернення: 5.10.2024).
19. N. A. Smith, D. A. Smith, and R. W. Tromble, “Context-Based Morphological Disambiguation with Random Fields,” in Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing, [Електронний ресурс]. Режим доступу: https://aclanthology.org/H05-1060 (дата звернення: 10.10.2024).
20. «English Machine translation System Based on Neural Network Algorithm», 2023. Author: Suxia Lei, You Li [Електронний ресурс]. Режим доступу: https://doi.org/10.1016/j.procs.2023.11.047 (дата звернення: 10.10.2024).

ДОДАТКИ

Додаток А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В. Н. Каразіна

Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки
Рівень вищої освіти (освітньо-кваліфікаційний рівень) **Магістр**
Галузь знань: 17 – Електроніка, автоматизація та електронні комунікації
Спеціальність: 174 – Автоматизація, комп'ютерно-інтегровані технології та робототехніка
Освітня програма «Комп'ютеризовані системи управління та автоматика»

ЗАТВЕРДЖУЮ
в.о. завідувач комп'ютерних
систем та робототехніки
к. ф.-м. н., доц. ХРУСЛОВ М. М.
«12» вересня 2024 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ МАГІСТРА

Шарапова Ігоря Миколайовича
(прізвище, ім'я, по батькові студента)

1. Тема роботи **Модель автоматизованого перекладу текстів на основі алгоритмів машинного навчання**

керівник роботи Стрілець Вікторія Євгенівна, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету ***№ 4101-5/3657 від 12 листопада 2024 року***

2. Строк подання студентом роботи ***30 листопада 2024 року***

3. Перелік питань, які потрібно розробити
1. Аналіз підходів і методів машиного навчання для автоматизованого перекладу текстових даних.
2. Аналіз особливостей задачі перекладу текстової інформації на зображеннях.
3. Розробка моделі автоматизованого перекладу тексту на основі нейронних мереж.
3. Програмна реалізація моделі автоматизованого перекладу тексту.

4. План роботи

№ з/п	Назви етапів роботи	Термін виконання етапів роботи
1	Проведення літературного огляду з методів машиного навчання для автоматизованого перекладу	1.09.2024 — 05.09.2024
2	Аналіз методів машиного навчання для автоматизованого перекладу тексту, зокрема на зображеннях	06.09.2024 — 15.09.2024
3	Пошук даних для навчання моделі автоматизованого перекладу текстів на основі алгоритмів машинного навчання	16.09.2024 — 25.09.2024
4	Розробка моделі для автоматизованого перекладу на основі методів машиного навчання, зокрема штучних нейронних мереж	26.09.2024 — 10.10.2024
5	Тестування моделі для автоматизованого перекладу та аналіз отриманих результатів	11.10.2024 — 20.10.2024
6	Розробка рекомендацій щодо застосування моделі для автоматизованого перекладу	21.10.2024 — 30.10.2024
7	Оформлення пояснювальної записки	31.10.2024 — 15.11.2024
8	Представлення кваліфікаційної роботи керівнику та рецензенту	15.11.2024 - 30.11.2024

5. Дата видачі завдання ***05 вересня 2024року.***

Студент _____ І.М. Шарапов _____
ініціали, прізвище
підпис

Керівник роботи _____ В.Є. Стрілець _____
ініціали, прізвище
підпис

Додаток Б

Затверджую

_____

«_____»_____2024 р.

Технічне завдання

на розробку програмного виробу «*Модель автоматизованого перекладу текстів на основі алгоритмів машинного навчання*»

1.	Введення	1.1. Назва: **Модель автоматизованого перекладу текстів на основі алгоритмів машинного навчання**. 1.2. Галузь застосування: інформаційні технології, переклад, комп'ютерні технології, машинне навчання, лінгвістика.
2.	Підстава для розробки	2.1. Навчальний план за спеціальністю 174 – Автоматизація та комп'ютерно-інтегровані технології 2.2. Завдання на кваліфікаційну роботу магістра № ***4101-5/3657*** від ***12 листопада 2024 року*** (представити як Додаток А до пояснювальної записки до кваліфікаційної роботи).
3.	Призначення розробки	3.1. Мета розробки: створення моделі автоматизованого перекладу текстів на основі алгоритмів машинного навчання 3.2. Призначення розробки надає можливість автоматизувати процес перекладу текстів з однієї мови на іншу, підвищуючи його швидкість та ефективність. 3.3. Вихідні дані розробки: текстові дані для навчання та тестування моделі, експериментальні данні.
4.	Технічні вимоги до програмного виробу	4.1. Вимоги до функціональних характеристик: здатність до перекладу текстів з однієї мови на іншу мову, можливість налаштування параметрів моделі для покращення якості перекладу. 4.2. Вимоги до надійності: забезпечення певного рівня точності та адекватності перекладу, забезпечувати стабільну роботу та відсутність критичних помилок. 4.3.Вимоги до умов експлуатації: немає 4.4. Вимоги до складу і параметрів технічних засобів: звичайне обчислювальне обладнання, ПК, тощо

<table>
<tr><td></td><td></td><td colspan="2">4.5. Вимоги до інформаційної та програмної сумісності: немає
4.6. Вимоги до маркування та упаковки: немає
4.7. Вимоги до транспортування і зберігання: на звичайних носіях інформації
4.8. Спеціальні вимоги: немає.</td></tr>
<tr><td>5.</td><td>Вимоги до програмної документації</td><td colspan="2">Програмною документацією до виробу Модель автоматизованого перекладу текстів на основі алгоритмів машинного навчання» вважати:
1) Справжнє Технічне завдання на розробку виробу (представити у вигляді Додатку Б до пояснювальної записки до кваліфікаційної роботи).
2) Методику розрахунку інформативності змінних стану (у вигляді глав 3.3 та 3.4 пояснювальної записки до кваліфікаційної роботи).
3) Опис виробу (представити в розділі 3 пояснювальної записки до кваліфікаційної роботи)</td></tr>
<tr><td>6.</td><td>Вимоги до техніко-економічних показників</td><td colspan="2">Програмною документацією до виробу «Модель автоматизованого перекладу текстів на основі алгоритмів машинного навчання» вважати:
1) Справжнє Технічне завдання на розробку виробу (представити у вигляді Додатку Б до пояснювальної записки до кваліфікаційної роботи).
2) Методику розрахунку інформативності змінних стану (у вигляді глав 3.3 та 3.4 пояснювальної записки до кваліфікаційної роботи).
3) Опис виробу (представити в розділі 3 пояснювальної записки до кваліфікаційної роботи)</td></tr>
<tr><td rowspan="4">7.</td><td rowspan="4">Стадії і етапи розробки</td><td>Дата</td><td>Назва етапу</td></tr>
<tr><td>Від 1 вересня 2024
до 5 вересня 2024</td><td>Проведення літературного огляду з методів машинного навчання для автоматизованого перекладу</td></tr>
<tr><td>Від 6 вересня 2024
до 15 вересня 2024</td><td>Аналіз методів і нейромережевих моделей машинного навчання для автоматизованого перекладу тексту</td></tr>
<tr><td>Від 16 вересня 2024
до 25 вересня 2024</td><td>Пошук даних для навчання моделі автоматизованого перекладу текстів на основі алгоритмів машинного навчання</td></tr>
</table>

		Від 26 вересня 2024 до 10 жовтня 2024	Розробка моделі для автоматизованого перекладу на основі методів машинного навчання, зокрема штучних нейронних мереж
		Від 11 жовтня 2024 до 20 жовтня 2024	Тестування моделі для автоматизованого перекладу та аналіз отриманих результатів
		Від 21 жовтня 2024 до 30 жовтня 2024	Розробка рекомендацій щодо застосування моделі для автоматизованого перекладу
		Від 31 жовтня 2024 до 15 листопада 2024	Оформлення пояснювальної записки
		Від 15 листопада 2024 до 30 листопада 2024	Представлення кваліфікаційної роботи керівнику та рецензенту
8.	Порядок контролю і приймання програмного продукту (моделі)	1. Перевірку ходу розробки програми виконувати раз в 3 тижні. 2. Захист розробленої моделі провести на засіданні Атестаційної комісії. 3. Пояснювальну записку подати на паперових носіях в 1 примірнику і в електронному вигляді в 1 примірнику.	

Виконавець
студент групи КУ- 61
Шарапов І. М._____

Замовник
к. т. н.,
Стрілець В. Є. _____

Додаток В

Програма і методика випробувань програмного виробу «МОДЕЛЬ АВТОМАТИЗОВАНОГО ПЕРЕКЛАДУ ТЕКСТІВ НА ОСНОВІ АЛГОРИТМІВ МАШИННОГО НАВЧАННЯ»

1 Об'єкт випробувань

1. Назва програмного виробу: «Модель автоматизованого перекладу текстів на основі алгоритмів машинного навчання»
2. Галузь застосування : Автоматизація перекладу текстів, інформаційні технології, машинне навчання, обробка природної мови.
3. Перераховані відомості запозичуються з відповідних розділів Технічного завдання.

2. Мета випробувань

Перевірка відповідності функціональності програмної реалізації системи заявленим функціональним можливостям в технічному завданні (Додаток Б до пояснювальної записки до кваліфікаційної роботи).

3. Загальні положення

1) Підстави для проведення випробувань

Підставою для проведення випробувань є наказ про призначення атестаційної комісії.

2) Місце і тривалість випробувань

Приймальні (приймально-здавальні) випробування проводяться на базі комп'ютерного класу кафедри в період роботи атестаційної комісії.

3) Обсяг випробувань

Приймальні випробування програмного вироби проводяться в обсязі відповідному цієї програми і методики випробувань.

4) Організації, які беруть участь у випробуваннях

Приймальні випробування проводяться атестаційною комісією напередодні засідання (або в процесі засідання) за участю Замовника, Виконавця та інших осіб, присутніх на засіданні.

4. Вимоги до програми або програмного виробу

Модель повинна задовольняти наступним вимогам:

1) працювати на основних операційних системах: Windows, Linux, MacOS;
2) вимоги до надійності;
3) передбачити захист від некоректних дій користувача;
4) сумісність з іншими програмними продуктами;
5) зменшити об’єм програмного коду необхідного для створення веб-додатків;
6) бути легко розширюваною;
7) елементи програми повинні бути ізольовані одне від одного для зменшення їх впливу на роботи програми під час редагування програмного коду;

8) вимоги до складу і параметрів технічних засобів;
9) вимоги до маркування та упаковки (не висуваються);
10) вимоги до транспортування і зберігання (не висуваються).

Спеціальні вимоги (не пред'являються).

5. Вимоги до програмної документації

Програмною документацією до виробу «Модель автоматизованого перекладу текстів на основі алгоритмів машинного навчання» вважати:

1. Технічне завдання на розробку програми (представити як Додаток Б до пояснювальної записки до кваліфікаційної роботи);
2. Програму і методику випробувань розробленої програми (представити як Додаток В до пояснювальної записки до кваліфікаційної роботи);
3. рекомендацій щодо застосування створеної програмної стандартизації у проектах (представити в Розділі 3 пояснювальної записки до кваліфікаційної роботи).
4. Текст програми(представити як Додаток Г до пояснювальної записки до кваліфікаційної роботи).

6. Засоби і порядок випробувань

6.1 Засоби випробувань

Для проведення випробувань необхідно: середовище Google Colaboratory з доступом до інтернету, набір даних для навчання моделі перекладу, тестовий набір даних з реченнями англійською мовою та їх правильними перекладами на українську.

6.2 Порядок проведення випробувань

Як правило, випробування проводяться в два етапи:

-ознайомчий (1-й етап);

-випробування програмного виробу (2-й етап).

Перелік перевірок, що проводяться на 1 етапі випробувань, включає в себе:

1. Перевірку комплектності програмної документації.
2. Перевірка комплектності складу програмної документації здійснюється за критерієм наявності зазначеної в ТЗ документації.
3. Перевірку комплектності складу технічних і програмних засобів.
4. Методику проведення перевірок на 1 етапі випробувань.
5. Якість програмної документації перевіряється на відповідність вимогам стандартів ЕСПД.

Перелік перевірок, що проводяться на 2 етапі випробувань, включає в себе:

1. перевірку відповідності технічних характеристик програми вимогам технічного завдання;
2. перевірку ступеня виконання функціональних вимог до програми;
3. перевірка точності перекладів на основі тестових даних.

1. Програма працює відповідно до умов експлуатації операційних систем MS Windows, Linux та MacOS.

2. Для роботи необхідний компілятор мови програмування python, версії не нижчої ніж 3.0

3. Порядок проведення випробувань:

3.1. Запуск програми в Google Colab;

3.2. Виконання навчання моделі на тестовому наборі даних;

3.3. Перевірка точності моделі на валідаційному наборі даних;

3.4. Після натискання на екрані з’явиться результат перекладу.

Для проведення випробувань пропонується тест 1, тест 2 та тест 3.

Тест 1

1. Введення даних для створення моделі трансформер;
2. Перевірка виконання навчання моделі на тренувальному наборі даних

Model: "transformer"

Layer (type)	Output Shape	Param #	Connected to
encoder_inputs (InputLayer)	(None, None)	0	-
positional_embedding (PositionalEmbedding)	(None, None, 256)	128,005,120	encoder_inputs[0][0]
decoder_inputs (InputLayer)	(None, None)	0	-
transformer_encoder (TransformerEncoder)	(None, None, 256)	5,258,752	positional_embedding[…
functional_3 (Functional)	(None, None, 500000)	265,971,232	decoder_inputs[0][0], transformer_encoder[0…

```
 Total params: 399,235,104 (1.49 GB)
 Trainable params: 399,235,104 (1.49 GB)
 Non-trainable params: 0 (0.00 B)
Epoch 1/40
/usr/local/lib/python3.10/dist-packages/tensorflow/python/framework/indexed_slices.py:446: UserWarning: Converting sparse IndexedSlic
  warnings.warn(
1993/1993 ━━━━━━━━━━━━━━━━━━━━ 219s 86ms/step - accuracy: 0.8068 - loss: 5.5652 - val_accuracy: 0.8155 - val_loss: 1.6904
Epoch 2/40
1993/1993 ━━━━━━━━━━━━━━━━━━━━ 109s 54ms/step - accuracy: 0.8225 - loss: 1.5381 - val_accuracy: 0.8286 - val_loss: 1.4329
Epoch 3/40
1993/1993 ━━━━━━━━━━━━━━━━━━━━ 106s 53ms/step - accuracy: 0.8322 - loss: 1.4043 - val_accuracy: 0.8737 - val_loss: 1.0972
Epoch 4/40
1993/1993 ━━━━━━━━━━━━━━━━━━━━ 107s 54ms/step - accuracy: 0.8855 - loss: 1.0213 - val_accuracy: 0.9092 - val_loss: 0.8651
Epoch 5/40
1993/1993 ━━━━━━━━━━━━━━━━━━━━ 107s 54ms/step - accuracy: 0.9112 - loss: 0.8407 - val_accuracy: 0.9197 - val_loss: 0.7703
Epoch 6/40
1993/1993 ━━━━━━━━━━━━━━━━━━━━ 108s 54ms/step - accuracy: 0.9211 - loss: 0.7498 - val_accuracy: 0.9287 - val_loss: 0.6975
```

Рис. В.1 Тест 1

Model: "transformer"

Layer (type)	Output Shape	Param #	Connected to
encoder_inputs (InputLayer)	(None, None)	0	-
positional_embedding_2 (PositionalEmbedding)	(None, None, 256)	12,805,120	encoder_inputs[0][0]
decoder_inputs (InputLayer)	(None, None)	0	-
transformer_encoder_1 (TransformerEncoder)	(None, None, 256)	5,258,752	positional_embedding_…
functional_7 (Functional)	(None, None, 50000)	35,121,232	decoder_inputs[0][0], transformer_encoder_1…

Total params: 53,185,104 (202.89 MB)
Trainable params: 53,185,104 (202.89 MB)
Non-trainable params: 0 (0.00 B)

```
Epoch 1/70
997/997 ━━━━━━━━━━━━━━━━━━━━ 134s 111ms/step - accuracy: 0.8038 - loss: 5.0580 - val_accuracy: 0.8213 - val_loss: 1.4646
Epoch 2/70
997/997 ━━━━━━━━━━━━━━━━━━━━ 85s 86ms/step - accuracy: 0.8211 - loss: 1.4566 - val_accuracy: 0.8260 - val_loss: 1.3660
Epoch 3/70
997/997 ━━━━━━━━━━━━━━━━━━━━ 83s 83ms/step - accuracy: 0.8265 - loss: 1.3676 - val_accuracy: 0.8334 - val_loss: 1.3013
Epoch 4/70
997/997 ━━━━━━━━━━━━━━━━━━━━ 82s 82ms/step - accuracy: 0.8345 - loss: 1.2933 - val_accuracy: 0.8596 - val_loss: 1.1123
Epoch 5/70
997/997 ━━━━━━━━━━━━━━━━━━━━ 81s 82ms/step - accuracy: 0.8709 - loss: 1.0412 - val_accuracy: 0.9051 - val_loss: 0.8235
Epoch 6/70
997/997 ━━━━━━━━━━━━━━━━━━━━ 81s 81ms/step - accuracy: 0.9077 - loss: 0.7931 - val_accuracy: 0.9211 - val_loss: 0.6913
Epoch 7/70
997/997 ━━━━━━━━━━━━━━━━━━━━ 81s 81ms/step - accuracy: 0.9229 - loss: 0.6706 - val_accuracy: 0.9347 - val_loss: 0.6041
```

Рис. В.2 Тест 1

Тест 2

1. Перевірка виконання перекладу моделі: Запустити функцію перекладу з новим, раніше не баченим реченням англійською мовою

2. Введення даних: Подати моделі векторизоване речення та цільовий токен;

3. Очікуваний результат: Модель повинна згенерувати послідовність токенів, що представляють переклад речення.

```
Eng: Tom needs some serious help.
Ukr: [start] Тома потрібна серйозна допомога. [end]

Eng: I just finished the work.
Ukr: [start] Я щойно закінчив роботу. [end]

Eng: I think she is over forty years old.
Ukr: [start] Я думаю, що їй понад сорок років. [end]

Eng: I thought that Tom would make us breakfast.
Ukr: [start] Я думав, що Том змусить нас снідати. [end]

Eng: I will abolish capital punishment.
Ukr: [start] Я скасую смертну кару. [end]

Eng: I rescued the cat.
Ukr: [start] Я врятував кішку. [end]

Eng: We're going to have a birthday party for Tom Monday evening.
Ukr: [start] Ми будемо проводити день народження для Тома в понеділок ввечері. [end]

Eng: You seem busy.
Ukr: [start] Ви, здається, зайняті. [end]
```

Рис. В.3 Тест 2

Тест 3

1. Користувачеві пропонується ввести текст англійською мовою.

2. Система перекладає введений текст на українську мову та відображає переклад.

3. Тест продовжується доти, доки користувач не введе команду завершення (q).

```
Введіть текст на англійській мові (або 'q' для завершення): I visited Australia
Переклад: Я відвідав Австралію.
Введіть текст на англійській мові (або 'q' для завершення): I think that Tom didn't tell us the truth.
Переклад: Я думаю, Том сказав нам неправду.
Введіть текст на англійській мові (або 'q' для завершення): If you see a mistake, then please correct it.
Переклад: Якщо ти бачиш помилку, виправ її, будь ласка.
Введіть текст на англійській мові (або 'q' для завершення): The decision has not yet been
Переклад: Рішення ще не прийнято.
Введіть текст на англійській мові (або 'q' для завершення): turned on my computer
Переклад: Я ввімкнула свій комп'ютер.
Введіть текст на англійській мові (або 'q' для завершення): don't kill me
Переклад: Не цькуйте мене.
Введіть текст на англійській мові (або 'q' для завершення): q
```

Рис. В.4 Тест 3

Тест вважається пройденим, якщо відбуваються вказані операції і їх відображення у програмному продукті.

Висновки: тест 1 успішно пройшов випробування, тест 2 успішно пройшов випробування і тест 3 успішно пройшов випробування. Випробування пройшло успішно.

Виконавець: студент групи КУ-61, Шарапов І. М 

Додаток Г

Фрагмент коду розробленої моделі автоматизованого перекладу текстів на основі алгоритмів машинного навчання

```
[ ] import os
    import pathlib
    import random
    import string
    import re
    import numpy as np
    import tensorflow as tf
    from tensorflow import keras
    from tensorflow.keras import layers
    from tensorflow.keras.layers import TextVectorization
    import nltk
    from nltk.stem import WordNetLemmatizer
    import tensorflow.strings as tf_strings

    # Set Keras backend
    os.environ["KERAS_BACKEND"] = "tensorflow"
```

Завантаження даних Ми працюватимемо з набором даних перекладу з англійської на украънську наданим Anki . Завантажимо його:

```
# Завантаження даних
if not os.path.exists("ukr.txt"):
    !wget https://www.manythings.org/anki/ukr-eng.zip
    !unzip ukr-eng.zip

text_file = pathlib.Path(".") / "ukr.txt"
```

```
--2024-12-03 11:53:44--  https://www.manythings.org/anki/ukr-eng.zip
Resolving www.manythings.org (www.manythings.org)... 173.254.30.110
Connecting to www.manythings.org (www.manythings.org)|173.254.30.110|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 4896171 (4.7M) [application/zip]
Saving to: 'ukr-eng.zip'
```

```
ukr-eng.zip         100%[===================>]   4.67M  22.3MB/s    in 0.2s

2024-12-03 11:53:45 (22.3 MB/s) - 'ukr-eng.zip' saved [4896171/4896171]

Archive:  ukr-eng.zip
  inflating: ukr.txt
  inflating: _about.txt
```

Розбір даних Кожен рядок містить англійське речення та відповідне йому українське речення. Англійське речення є вихідною послідовністю , а українське - цільовою послідовністю . "[start]"Ми додаємо лексему перед "[end]" українською мовою.

```
[ ] text_file = pathlib.Path(".") / "ukr.txt"
    # Читання даних з файлу
    with open(text_file) as f:
        lines = f.read().split("\n")[:-1]

    text_pairs = []
    for line in lines:
        parts = line.split("\t")
        eng = parts[0]
        ukr = parts[1]  # Беремо лише другий елемент
        ukr = ukr
        text_pairs.append((eng, ukr))
```

Ось як виглядають наші пари речень:

```
[ ] for _ in range(5):
        print(random.choice(text_pairs))
```

Тепер розділимо пари речень на навчальний набір, набір перевірки та тестовий набір.

```
[ ] random.shuffle(text_pairs)
    num_val_samples = int(0.1 * len(text_pairs))
    num_train_samples = len(text_pairs) - 2 * num_val_samples
    train_pairs = text_pairs[:num_train_samples]
    val_pairs = text_pairs[num_train_samples : num_train_samples + num_val_samples]
    test_pairs = text_pairs[num_train_samples + num_val_samples :]

    print(f"{len(text_pairs)} total pairs")
    print(f"{len(train_pairs)} training pairs")
    print(f"{len(val_pairs)} validation pairs")
    print(f"{len(test_pairs)} test pairs")
```

```
159432 total pairs
127546 training pairs
15943 validation pairs
15943 test pairs
```

Векторизація текстових даних

```
# Download NLTK resources
nltk.download('wordnet')
lemmatizer = WordNetLemmatizer()

def custom_standardization(text):
    """Custom text standardization function."""
    text = text.lower()  # Convert to lowercase
    return text

def lemmatize_text(text):
    """Lemmatize text in English."""
    words = text.split()
    lemmas = [lemmatizer.lemmatize(word) for word in words]
    return ' '.join(lemmas)

def preprocess_text(text):
    """Preprocess text including lemmatization."""
    if isinstance(text, tf.Tensor):
        text = text.numpy().decode('utf-8')

    text = custom_standardization(text)
    text = lemmatize_text(text)
    return text

# Prepare texts for vectorization
train_eng_texts = [pair[0] for pair in train_pairs]
train_ukr_texts = [pair[1] for pair in train_pairs]
```

```
[ ] # Vectorization parameters
    strip_chars = string.punctuation + "¿"
    vocab_size = 500000
    sequence_length = 20
    batch_size = 64


    def custom_standardization_tf(input_string):
        """TensorFlow custom text standardization."""
        lowercase = tf_strings.lower(input_string)
        return tf_strings.regex_replace(lowercase, "[%s]" % re.escape(strip_chars), "")

    eng_vectorization = TextVectorization(
        max_tokens=vocab_size,
        output_mode="int",
        output_sequence_length=sequence_length,
    )
    ukr_vectorization = TextVectorization(
        max_tokens=vocab_size,
        output_mode="int",
        output_sequence_length=sequence_length + 1,
        standardize=custom_standardization_tf,
    )

    # Adapt vectorizers
    eng_vectorization.adapt(train_eng_texts)
    ukr_vectorization.adapt(train_ukr_texts)
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
```

Далі збираємо наскрізну модель.

```
[ ] embed_dim = 256
    latent_dim = 2048
    num_heads = 16

    encoder_inputs = keras.Input(shape=(None,), dtype="int64", name="encoder_inputs")
    x = PositionalEmbedding(sequence_length, vocab_size, embed_dim)(encoder_inputs)
    encoder_outputs = TransformerEncoder(embed_dim, latent_dim, num_heads)(x)
    encoder = keras.Model(encoder_inputs, encoder_outputs)

    decoder_inputs = keras.Input(shape=(None,), dtype="int64", name="decoder_inputs")
    encoded_seq_inputs = keras.Input(shape=(None, embed_dim), name="decoder_state_inputs")
    x = PositionalEmbedding(sequence_length, vocab_size, embed_dim)(decoder_inputs)
    x = TransformerDecoder(embed_dim, latent_dim, num_heads)(x, encoded_seq_inputs)
    x = layers.Dropout(0.5)(x)
    decoder_outputs = layers.Dense(vocab_size, activation="softmax")(x)
    decoder = keras.Model([decoder_inputs, encoded_seq_inputs], decoder_outputs)

    decoder_outputs = decoder([decoder_inputs, encoder_outputs])
    transformer = keras.Model(
        [encoder_inputs, decoder_inputs], decoder_outputs, name="transformer"
    )
```