Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Центр Післядипломної освіти
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Вирішення задачі виправлення граматичних помилок в текстах
з використанням сучасних технологій глибинного навчання
(тема)

Виконав:
студент 2 курсу, групи СШІздм-21-1
Яковлева О. М.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

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

Освітня програма Системи штучного інтелекту
(повна назва спеціалізації)

Керівник доц. Шевченко О.Ю.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ В.О. Філатов
(підпис) (прізвище, ініціали)

2023 р.

Харківський національний університет радіоелектроніки

Центр Післядипломної освіти
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

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

Освітня програма Системи штучного інтелекту (СШІ)
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____» _____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Яковлевій Ользі Миколаївні
(прізвище, ім'я, по батькові)

1. Тема роботи Вирішення задачі виправлення граматичних помилок в текстах з використанням сучасних технологій глибинного навчання

затверджена наказом університету від 31 березня 20 23 р. № 73Стз
2. Термін подання студентом роботи до екзаменаційної комісії 23 травня 20 23 р.
3. Вихідні дані до роботи NLP, GEC, UA-GEC, Браунівський корпус, Lang-8, NuCLE, Seq2Seq, mT5, MaxMatch, GLEU, BERT, Python, Stanza, пермутація, комбінація, трансферне навчання, оптимізація гіперпараметрів, тонке налаштування

4. Перелік питань, що потрібно опрацювати в роботі
1) Аналіз предметної галузі
2) Seq2Seq підхід, трансферне навчання
3) UA-GEC корпус, оптимізація ресурсів GPU при навчанні моделі
4) Налаштування гіперпараметрів
5) Метрика n-грам, метрика MaxMatch
6) Публічні датасети, аугментовані дані, пермутація

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри)_____

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	03.04.2023	Виконано
2	Аналіз завдання та пошук літератури за темою	04.04.2023-09.04.2023	Виконано
3	Вибір моделей і вхідних даних	10.04.2023-11.04.2023	Виконано
4	Експериментальне моделювання та навчання моделей	12.04.2023-25.04.2003	Виконано
5	Оцінка результатів	26.04.2023-28.04.2023	Виконано
6	Написання пояснювальної записки	29.03.2023-05.05.2023	Виконано
7	Попередній захист	13.05.2023	Виконано
8	Захист перед ЕК	23.05.2023	Виконано

Дата видачі завдання 3 квітня 2023 р.

Студент _____ _____
(підпис)

Керівник роботи _ _____ доц. Шевченко О.Ю.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 105 с., 60 рис., 10 табл., 18 формул, 2 дод., 70 джерел.

ВИПРАВЛЕННЯ ГРАМАТИЧНИХ ПОМИЛОК, ЗГОРТКОВА НЕЙРОННА МЕРЕЖА, ПОСЛІДОВНІСТЬ ДО ПОСЛІДОВНОСТІ, СТАТИСТИЧНИЙ МАШИННИЙ ПЕРЕКЛАД, ТРАНСФОРМАТОРІВ, ШТУЧНИЙ ІНТЕЛЕКТ.

Метою кваліфікаційної роботи є розробка інтелектуальної системи виправлення граматичних помилок для української мови на основі глибинних рекурентних нейронних мереж з використанням анотованих та синтетичних наборів даних.

Об’єктом дослідження є процес виправлення граматичних помилок в українській мові на основі українського анотованого корпусу з додатковим використанням синтетичних даних.

Предметом дослідження є глибинні рекурентні мережі, призначені для вирішення задач виправлення граматичних помилок, що навчаються як на анотованих, так і на синтетичних наборах даних.

Методи дослідження – теорія оптимізації, теорія штучних нейронних мереж, теорія глибинних нейронних мереж, теорія систем опрацювання природної мови.

Ця кваліфікаційна робота зосереджена на досліджені та аналізі методів та системи для створення GEC застосунку української мови, використовуючи один з найсучасніших підходів – нейромережевого машинного перекладу, яке інтерпретує завдання GEC як одномовне переписування помилкового тексту в граматично коректний.

ABSTRACT

Explanatory note: 105 p., 60 fig., 10 tabl., 18 formulas, 2 ann., 70 sources.

ARTIFICIAL INTELLIGENCE, BIDIRECTIONAL ENCODER REPRESENTATIONS FROM TRANSFORMERS, BILINGUAL EVALUATION UNDERSTUDY, CONVOLUTIONAL NEURAL NETWORK, GRAMMAR ERROR CORRECTION, SEQUENCE-TO-SEQUENCE, STATISTICAL MACHINE TRANSLATION.

The purpose of the qualification work is to develop an intelligent grammar correction system for the Ukrainian language based on deep recurrent neural networks using annotated and synthetic data sets.

The object of the study is the process of correcting grammatical errors in the Ukrainian language based on the Ukrainian annotated corpus with the additional use of synthetic data.

The subject of the research is deep recurrent networks designed to solve grammar correction problems trained on both annotated and synthetic data sets.

The research methods are optimization theory, artificial neural network theory, deep neural network theory, and natural language processing systems theory.

This qualification work focuses on the research and analysis of methods and systems for creating a GEC application for the Ukrainian language, using one of the most modern approaches – neural network machine translation, which interprets the GEC task as a monolingual rewriting of erroneous text into grammatically correct text.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів 8
Вступ 9
1 Аналіз предметної галузі та постановка задачі дослідження 11
1.1 Опрацювання природної мови 11
1.2 Базові етапи опрацювання природної мови 11
1.3 Задачі які вирішує NLP 15
1.4 Основні підходи до вирішення завдання виправлення граматичних помилок (GEC) 17
1.5 Текстові набори даних 30
1.6 Використання аугментованих даних 32
1.7 Постановка задачі дослідження 32
2 Методи для навчання системи виправлення граматичних помилок 34
2.1 Векторизація слів 34
2.2 Контекстуалізована векторизація слів на основі моделі ELMo 37
2.3 Модель BERT 39
2.4 Вирішення завдання GEC методом нейронного машинного перекладу 39
2.5 Домена адаптація моделей 43
2.6 Налаштування гіперпараметрів з використанням градієнтних методів оптимізації 44
2.7 Метрики оцінювання якості моделей виправлення граматичних помилок 48
3 Використання аугментованих даних в задачах gec 52
3.1 Аугментація даних 52
3.2 Методи створення створення аугментованих даних 53
3.3 Створення аугментованих даних за допомогою бібліотеки scikit- learn 55
3.4 Генерація даних довільної випадкової форми 60
3.5 Створення додаткових даних за допомогою бібліотеки pydbgen 62

4 Імітаційне моделювання 64
4.1 Середовище та бібліотеки 64
4.2 Застосування анотованих корпусів для навчання моделей 65
4.3 Створення аугментованих даних 67
4.4 Аугментація даних на основі комбінації 70
4.5 Векторизація текстових даних 72
4.6 Опис базової моделі для виправлення граматичних та синтаксичних помилок 74
4.7 Гіперпараметри для тонкого налаштування моделі 75
4.8 Результати навчання системи виправлення граматичних та синтаксичних помилок на корпусах української мови 77
4.9 Тренування моделей на аугментованих даних 83
4.10 Порівняльний аналіз якості GEC систем 88
Висновки 90
Перелік джерел посилання 91
Додаток А Вихідний код програми 99
Додаток Б Відомість кваліфікаційної роботи 105

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

AI – Artificial Inteligence – штучний інтелект (ШІ);

BERT – Bidirectional Encoder Representations From Transformers – двоспрямовані кодувальні представлення з трансформерів;

BLEU – Bilingual Evaluation Understudy – оцінка двомовного розуміння;

CBOW – Continuous-Bag-Of-Words – неперервна торба слів;

CNN – Convolutional Neural Network – згорткові нейронні мережі;

ELMo – Embeddings From Language Model – вбудування з мовної моделі;

GEC – Grammar Error Correction – виправлення граматичних помилок;

GRU – Gated Recurrent Unit – вентильні рекуреентні вузли;

GLEU – General Language Evaluation Understanding – оцінка загального розуміння мови;

LSTM – Long Short-Term Memory – довга короткочасна пам'ять;

LM – Language Model – мовна модель;

NER – Named-Entity Recognition – розпізнавання іменованих об’єктів;

NMT – Neural Machine Translation – нейронний машинний переклад;

NLP – Natural Language Processing – обробка природної мови;

POS – Part-Of-Speech – граматичні позначення;

RNN – Recurrent Neural Network – рекурентні нейронні мережі;

Seq2seq – Sequence-To-Sequence – метод послідовного навчання;

SMT – Statistical Machine Translation – статистичний машинний переклад;

SoftMax – нормована експоненційна функція;

SVM – Support Vector Machine – метод опорних векторів;

VRAM – Video Random Access Memory – внутрішня оперативна пам'ять графічного контролера.

ВСТУП

Протягом останніх років зі стрімким розвитком галузі обробки природної мови значної актуальності набула проблема вдосконалення людського правопису. Написання твору, публікації, листу клієнту чи повідомлення у чаті – в цих процесах люди постійно стикаються з неповним знанням усіх мовних правил, що призводить до допущення помилок. Особливо гостро ця проблема виражена для тих, хто не являється рідним носієм певної мови, однак її використання необхідне для виконання письмових завдань й досягненню робочих чи особистих цілей. Також до категорії тих, хто може бути більш схильний до написання текстів з граматичними помилками, являються діти та особи з порушеннями мовлення (дисграфія).

У порівнянні з англійською, українська мова може бути складнішою для розуміння комп'ютером. Як пишуть розробники першого українського анотованого набору даних GEC [1], однією з цікавих проблем є її розвинута морфологія. Для прикладу, в англійській мові слово «see» («бачити») має п'ять залежних від часу форм, які називаються дієвідмінами: see, sees, saw, seen і seeing. Водночас в українській мові дієслово «бачити» має двадцять шість можливих форм, які залежать від часу, роду, особи та інших параметрів. На відміну від англійської мови, українська є синтетичною мовою, що означає утворення синтаксичних зв'язків за допомогою флексій. В українській мові можна також довільно змінювати порядок слів, оскільки граматичне значення передають флексії.

Це багате різноманіття мови значно ускладнює вирішення завдання GEC, тому що ставить велику кількість параметрів та умов, які вручну прописати дуже складно, не враховуючи важливості попереднього досвіду в лінгвістичній експертизі. У цій роботі буде запропоновано одне з найсучасніших рішень, яке розглядає GEC як завдання нейронного машинного перекладу.

Зі стрімкою популяризацією української мови значної актуальності набула проблема вдосконалення правопису серед населення. Одним з її ефективних рішень є створення інтелектуальної системи для автоматичного виправлення граматичних, орфографічних та пунктуаційних помилок (grammar error correction, GEC). Якщо для англійської мови такі системи вже успішно реалізовані у виді кінцевих продуктів та користуються великим попитом, то український сегмент перебуває на стадії бурного розвитку. Ця робота зосереджена на розробці системи GEC для української мови, використовуючи один з найсучасніших підходів – нейронного машинного перекладу, яке інтерпретує завдання GEC як одномовне переписування помилкового тексту в граматично коректний.

Було проведено порівняльний аналіз існуючих GEC систем для української мови та огляд основних методів при їх реалізації на усіх етапах розробки, включаючи попередню обробку даних, математичне представлення слів для розуміння комп’ютером, методи трансферного навчання, оптимізації гіперпараметрів, метрик оцінювання якості. Набори даних є ключовими при навчанні, тому було проведено огляд усіх доступних GEC датасетів для української мови та виявлена необхідність у використанні власноруч згенерованих аугментованих (синтетичних) даних з метою покращення попередньо отриманих результатів.

Результатом застосування вищезазначених підходів та методів є різні варіації розробленої GEC системи для української мови, якість яких відповідно зростала з розміром моделей, яку було обрано за основу при навчанні. Використовуючи суміш анотованих та аугментованих даних було досягнуто приросту якості систем. Проведено порівняльний аналіз з аналогічними GEC системами для інших мов з невеликими ресурсами й задано майбутні напрямки для покращення наявних результатів української системи GEC.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1 Опрацювання природної мови

Опрацювання природної мови (Natural Language Processing – NLP) це поєднання таких наук, як лінгвістика, інформатика та штучний інтелект.

NLP поєднує обчислювальну лінгвістику – моделювання людської мови на основі правил, зі статистичними моделями, моделями машинного навчання та глибокого навчання. Разом ці технології дозволяють комп’ютерам обробляти людську мову у вигляді тексту або голосових даних і «розуміти» її повне значення, враховуючи наміри та настрої мовця чи письменника.

NLP дозволяє комп’ютерам розуміти природну мову так само, як це роблять люди. Незалежно від того, розмовна чи письмова мова, обробка природної мови використовує штучний інтелект, щоб приймати вхідні дані реального світу, обробляти їх і розуміти їх у спосіб, зрозумілий комп’ютеру. Подібно до того, як люди мають різні датчики сприйняття – наприклад, вуха, щоб чути, і очі, щоб бачити, – комп’ютери мають програми для читання та мікрофони для збору звуку. І так само, як у людей є мозок для обробки цих введених даних, у комп’ютерів є програма для обробки відповідних даних. У певний момент обробки введені дані перетворюються на код, який може зрозуміти комп’ютер.

1.2 Базові етапи опрацювання природної мови

NLP характеризується як складна проблема в інформатиці. Людська мова рідко буває точною або відкритою. Зрозуміти людську мову означає зрозуміти не лише слова, а й поняття та те, як вони пов’язані між собою, щоб створити значення. Незважаючи на те, що мова є однією з

найпростіших речей для людського розуму, багатозначність мови робить обробку природної мови складною проблемою для комп’ютерів.

Існує два основних етапи опрацювання природної мови: попередня обробка даних і розробка алгоритму. Попередня обробка даних включає підготовку та «очищення» текстових даних для комп’ютерів, щоб мати можливість їх аналізувати. попередня обробка трансформує дані в працездатну форму і виділяє в тексті особливості, з якими може працювати алгоритм. Це можна зробити кількома способами, зокрема:

Сегментація (Sentence Segmentation) – це можливість розбити текст на окремі речення. Поділ може відбуватись за допомогою звичного способу, розбитті коли можемо бачити розділовий знак. У сучасних реаліях часто використовують складніші методи, які працюють, навіть якщо документ не відформатований чітко. На рисунку 1.1 наведено приклад сегментації.

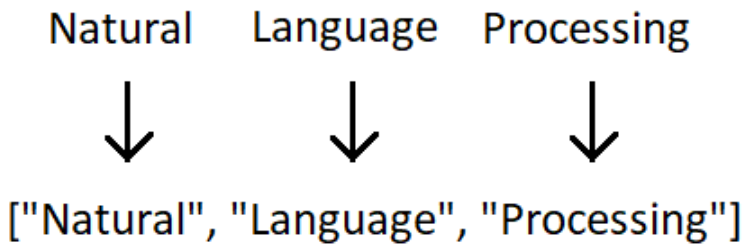


Рисунок 1.1 – Приклад сегментації речення “Natural Language Processing”

Токенізація (Tokenization) – це спосіб поділу фрагмента тексту на менші одиниці, які називаються токенами (рисунок 1.2). Коли ми розбили наш текст на речення ми можемо обробляти їх по одному. Розбиття речення на слова, ось за що відповідає токенізація. Це коли текст розбивається на менші одиниці для роботи.

Позначення частини мови (Part-of-speech tagging). Токенізувавши кожне речення, ми можемо надати певний маркер для кожного слова. Кожне слово можна визначити та виокремити до певної частини мови. Знання ролі

кожного слова в реченні допоможе нам зрозуміти, про що йдеться у речені. Для цього ми можемо вводити кожне слово у попередньо навчену модель класифікації частин мови. Модель повністю заснована на статистиці, тому вона насправді не розуміє, що означають слова, так само, як люди. Вона просто вміє вгадувати частину мови на основі схожих речень і слів, які бачила раніше. На рисунку 1.3 зображено приклад розподілу слів на відповідні частини мови.

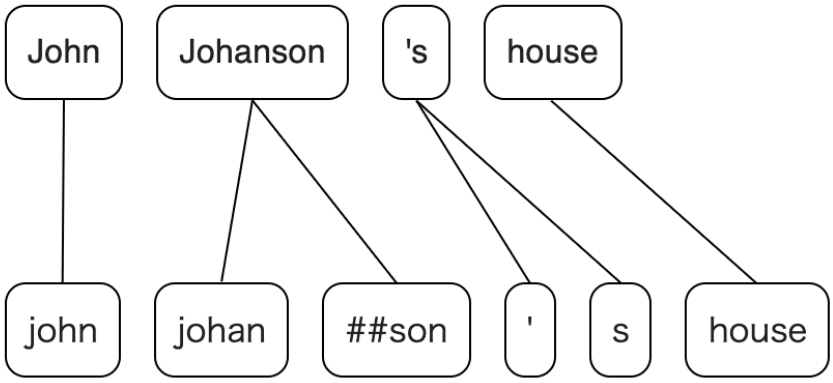


Рисунок 1.2 – Приклад токенізації слів, та утворення токенів

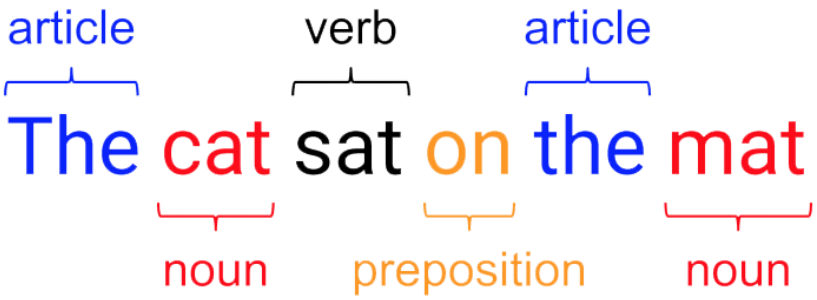


Рисунок 1.3 – Позначення слів на частини мови

Видалення стоп-слів (Stop word removal). Ідея полягає в тому, щоб просто видалити слова, які зазвичай зустрічаються в усіх документах у корпусі. Під час виконання статистичних даних щодо тексту ці слова вносять багато шуму, оскільки вони з’являються набагато частіше, ніж інші

слова. Артиклі та займенники класифікуються як стоп-слова. Такі слова видаляються з тексту, і залишаються унікальні слова, які пропонують найбільшу інформацію про текст. На рисунку 1.4 зображено результат використання процесу видалення стоп-слів.

Sample text with Stop Words	Without Stop Words
GeeksforGeeks – A Computer Science Portal for Geeks	GeeksforGeeks , Computer Science, Portal ,Geeks
Can listening be exhausting?	Listening, Exhausting
I like reading, so I read	Like, Reading, read

Рисунок 1.4 – Таблиця видалення стоп-слів

Лематизація (Lemmatization) – визначення найпростішої форми кожного слова в реченні. Це коли слова зводяться до кореневих форм для обробки. Лематизація, як правило, здійснюється за допомогою таблиці пошуку форм слів на основі їхньої частини мови та, можливо, наявності деяких користувацьких правил для обробки слів, які ви ніколи раніше не бачили. На рисунку 1.5 схематично зображено процес лематизації.

Після попередньої обробки даних розробляється алгоритм їх обробки. Існує багато різних алгоритмів обробки природної мови, але зазвичай використовуються два основних типи:

Система на основі правил (Ruled-based system). Ця система використовує ретельно розроблені лінгвістичні правила. Цей підхід використовувався на початку розвитку обробки природної мови і використовується досі.

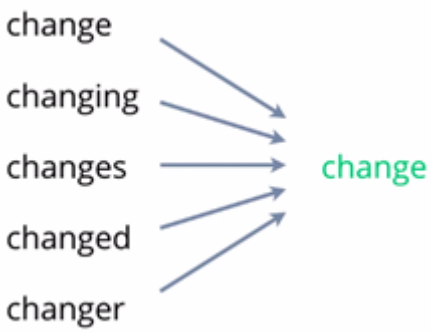


Рисунок 1.5 – Відображення лематизації слів

Система на основі машинного навчання (Machine learning-based system). Алгоритми машинного навчання використовують статистичні методи. Вони вчяться виконувати завдання на основі навчальних даних, які їм отримують, і коригують свої методи в міру обробки більше даних. Використовуючи комбінацію машинного навчання, глибокого навчання та нейронних мереж, алгоритми обробки природної мови відточують власні правила шляхом багаторазової обробки й навчання.

1.3 Задачі які вирішує NLP

Для чого використовується обробка природної мови? Деякі з основних функцій, які виконують алгоритми обробки природної мови:

Прогнозування тексту. Рішення NLP, такі як автовиправлення та автозаповнення, аналізують особисті мовні шаблони та визначають найбільш підходящі пропозиції для окремих користувачів. На рисунку 1.6

Результати пошукової системи. Пошукові системи використовують NLP, щоб краще розуміти, що шукають користувачі, і мати можливість швидко знаходити відповідну інформацію.

Класифікація тексту. Класифікація тексту має вирішальне значення для автоматизації розуміння, обробки та категоризації неструктурованого тексту. Моделі NLP дозволяють організувати дані за допомогою заздалегідь визначених тегів.

Вилучення тексту. Обробка природної мови здатна автоматично ідентифікувати конкретні ключові слова, назви продуктів, описи тощо в тексті.

Узагальнення тексту. NLP може швидко витягти важливу інформацію з тексту та узагальнити її на основі ключових фраз із тексту або визначених висновків.

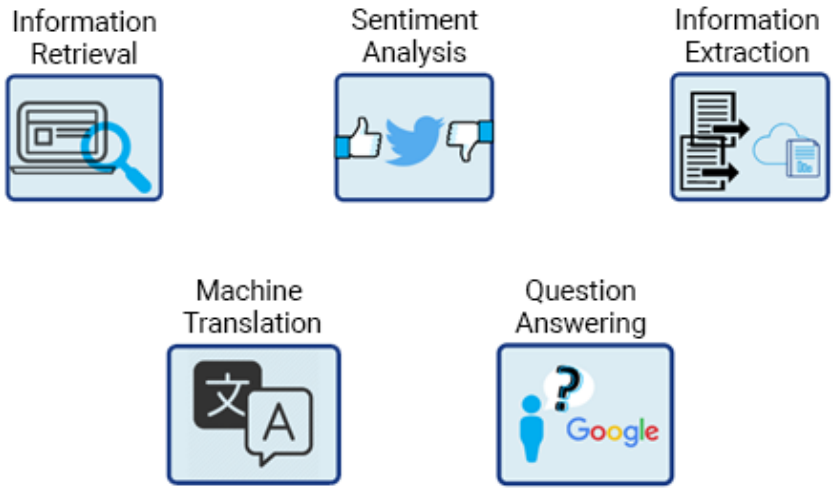


Рисунок 1.6 – Сфери застосування NLP

Фільтри електронної пошти. Фільтри електронної пошти є одним з найпоширеніших і основних видів використання NLP. Коли вони були вперше представлені, вони були не зовсім точними, але з роками машинного навчання на мільйонах зразків даних електронні листи сьогодні рідко потрапляють у неправильну папку вхідних.

Віртуальні помічники, голосові помічники або розумні колонки. Найпоширенішими є Siri від Apple і Alexa від Amazon, віртуальні помічники використовують технологію машинного навчання NLP для розуміння та автоматичної обробки голосових запитів. Алгоритми обробки природної мови дозволяють індивідуальним користувачам навчати помічників без

додаткового введення, вчитися на попередніх взаємодіях, згадувати пов’язані запити та підключатися до інших програм.

Чат боти. Чат-бот – це комп'ютерна програма, яка імітує людську розмову. Чат-боти використовують NLP, щоб розпізнати намір речення, визначити відповідні теми та ключові слова, навіть емоції та знайти найкращу відповідь на основі їхньої інтерпретації даних.

В даному випадку спектр можливостей NLP досить потужний, що наштовхує на усвідомлення можливості вдосконалення систем обробки природної мови та розвиток потенціалу разом із технологічним прогресом.

1.4 Основні підходи до вирішення завдання виправлення граматичних помилок (GEC)

Завдання на виправлення граматичних помилок (GEC) формулюють як знаходження та виправлення різного роду помилок в тексті, які насправді передбачають не тільки граматичні, а й орфографічні, пунктуаційні, та лексичні помилки. Система GEC приймає потенційно помилкове речення як входові дані та перетворює його на виправлену версію (рисунок 1.7).

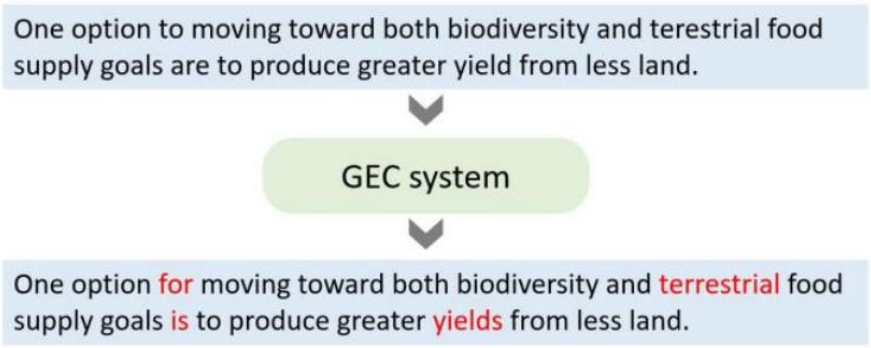


Рисунок 1.7 – Базове представлення системи GEC [2]

Для ознайомлення з історією вирішення цього завдання було проаналізовано кілька джерел, з яких найповнішу інформацію дають окремі публікації з оглядом різних підходів та останніх трендів [2], [3].

На основі правил. Перші дослідження задачі GEC проводились ще з 1980-их років, коли одну з її перших реалізацій, для англійської мови [1], успішно адаптували студенти університету для написання якісних текстів та наукових праць. У його розробці брали участь лінгвісти, працівники університету та студенти з хорошим знанням мови. Згодом, ці системи підлаштовували та значно розширювали для її використанням іноземними студентами, які вивчали англійську мову.

Вищезгаданий приклад використовував підхід на основі попередньо визначених правил (шаблони помилок), які оцінювали граматичну коректність тексту. Всі правила розроблялися вручну та вимагали цілої команди експертів, які постійно розширяли їх через комплексність мови та великої кількості винятків. Попри те, що була можливість поступово розширювати систему, саме використання залишилось контекстуально обмеженим та вимагало виконання значного обсягу роботи, не беручи вже до уваги створення подібних систем для менш вживаних мов.

На основі синтаксичного аналізу. Підхід перевірки на основі синтаксичного аналізу також можна віднести до одних з ранніх спроб реалізації систем GEC. Для неї використовується лексична база даних, розроблені аналізатори морфології та синтаксису. На кожне речення генерується синтаксична структура у вигляді дерева, яка при неуспішному аналізі повідомляє про помилковість тексту.

Перевагою цього підходу є можливість при розробці використовувати готові ресурси та знаходити більшу кількість помилок незалежно від характеру. Але необхідно врахувати те, що для вказання на конкретну помилку слід прописувати додатковий набір правил. Ознайомлюючись з прикладами реалізації цього підходу для мов, іншої ніж англійська, слід згадати систему GEC для арабської мови [2], оновленої системи GEC

другого покоління для латвійської [3] (перша використовувала підхід зі знаходженням шаблонів помилок), данської [4], грецької [5] та перших спроб реалізувати пенджабські GEC системи [6], [7] (рисунок 1.8).

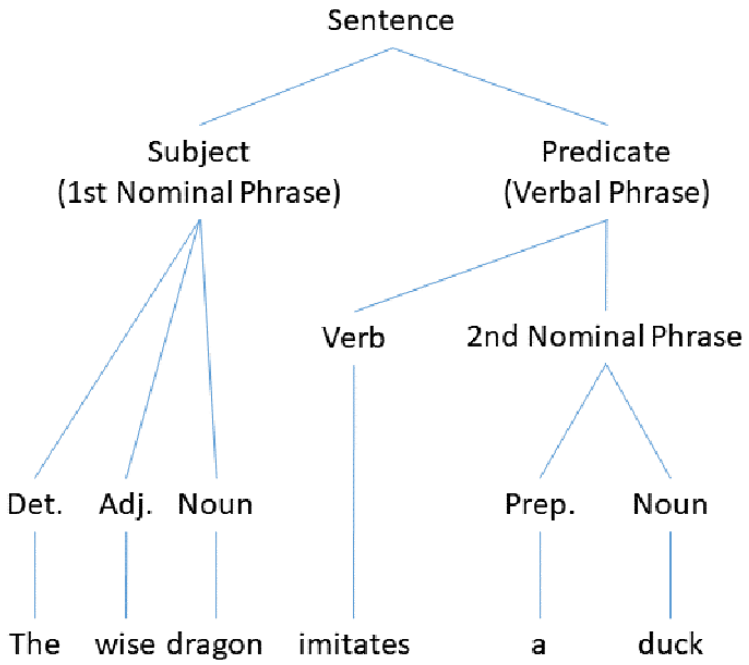


Рисунок 1.8 – Приклад синтаксичного дерево для речення [35]

На основі класифікації. Впродовж останніх двох десятиліть фокус поступово зміщувався в бік використання даних та методів машинного навчання. Одне з рішень інтерпретує GEC як завдання класифікації. У цьому підході виправлення приймаються за класові мітки, які необхідно передбачити, а в ролі ознак (фіч) використовуються токени (слова та пунктуація), які репрезентують контекст в якому помилка може виникнути. Туди включають n-грами (грама з грецького означає буква або знак), що можна пояснити як різні за кількістю токенів сусідні комбінації пар.

Також використовують POS (Part-of-speech) теги, які категоризують слова в тексті у відповідності з певною частиною мови залежно від значення слова та його контексту, та різні граматичні відносини – міжфункціональні зв’язки в складеному пункті (наприклад предмет, прямий об’єкт та

непрямий об’єкт). Усі названі фічі отримують векторну репрезентацію, за допомогою якої навчаються різні за методиками класифікатори, а саме методи максимальної ентропії, наївний байєсів та метод опорних векторів. При застосуванні навчений класифікатор здатний виявити та виправити помилку, зіставляючи вхідне слово з найоптимальнішим кандидатом, який він зуміє передбачити.

Але при реалізації таких систем як правило цього недостатньо, адже в реченні може бути кілька помилок й система виправить тільки одну з них, вважаючи всі інші частини граматично коректними. Одним з варіантів вирішення є побудова потоку в певному порядку з різних видів класифікаторів на кожний тип помилки, хоча при цьому залишається проблема з випадками залежних помилок [11].

Для вирішення проблеми залежних помилок було поєднано підхід на основі класифікації зі статистичним машинним перекладом. Реалізація була наступною: декодер ітераційно генерує виправлення нових гіпотез із поточних, оцінює їх на основі особливостей граматичної правильності та стилістики. Фічі включають оцінки різних класифікаторів для конкретних типів помилок, до прикладу артиклі чи прийменники. Процес повторюється до моменту, поки не стане можливим прогнозувати нові гіпотези, або не буде досягнуто максимальної кількості ітерацій [12].

В більш сучасних підходах методи глибинного навчання були поєднані з системою на основі класифікації (рисунок 1.9). Не спираючись на традиційний підхід з процесом розробки фіч, який потребує людської винахідливості та попередніх знань в галузі NLP, виникла спроба застосувати згорткові нейронні мережі (Convolutional neural network – CNN) для добування усіх фіч [20]. Вони репрезентують контекст в реченнях, завдяки якому можливо знаходити та виправляти помилки. Експерименти були проведені на наборі даних CoNLL-2013, які досягли 38,10% в F1 (F- міра), що перевершило кращу систему на той час (33,40%).

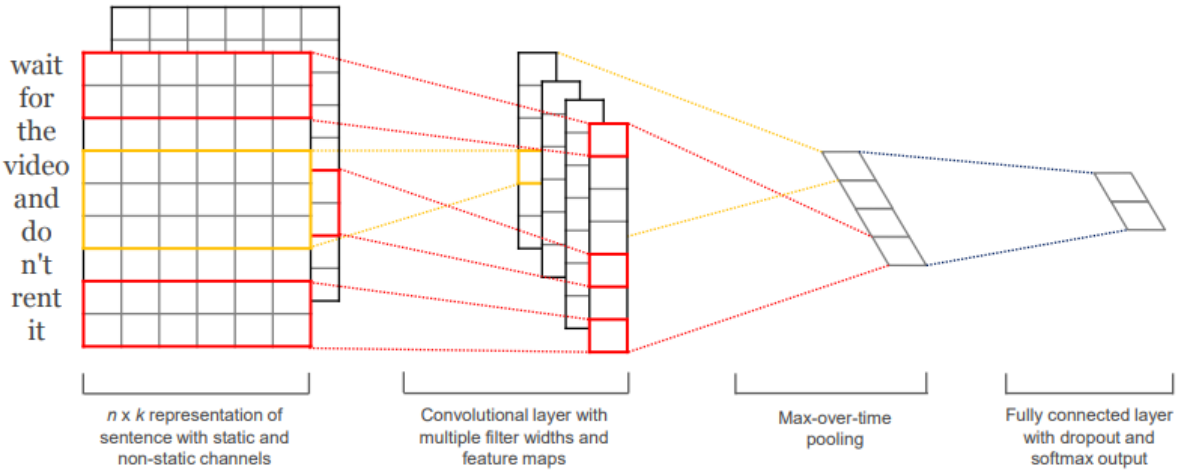


Рисунок 1.9 – Архітектура CNN з двома каналами для контекстуального представлення речення [36]

Дещо схожий підхід базується на репрезентації контексту за допомогою Gated Recurrent Units (GRUs) та використанні глибинних фіч напряму, уникаючи довгого та комплексного їх добування для кожного типу помилки самотужки [21]. В цій методиці відповідна модель навчає функцію вбудовування контекстів навколо певного слова, після чого має змогу передбачити його з допомогою навченої функції. Якщо передбачене слово відрізняється від оригінального, то оригінальне позначається як помилка, а передбачене використовується як виправлення.

На основі статистичного машинного перекладу (SMT). Підхід на основі статистичного машинного перекладу (SMT) був вперше використаний для виправлення великої кількості іменників знайдених у корпусі по вивченню китайської мови в 2006 році [13]. Модель досягла хорошого результату в 61,52% точності, що виправдало новий спосіб і продемонструвало можливість в майбутньому використовувати набори з паралельних даних (пари з неправильних та виправлених речень) для вирішення більш узагальнених проблем.

Через певний проміжок часу спроби розробити SMT систему для вирішення завдання GEC поновились. Однією з причин стало

опублікування великого навчального паралельного корпусу з веб-сервісу Lang-8, у якому англійські речення іноземних студентів коректувались вільними носіями мови. Проводились дослідження стосовно ефективності таких даних на прикладі навченої SMT моделі для виправлення різних типів помилок [14], яка довела ефективність, хоча викликом залишились помилки, які потребують довготривалої контекстної інформації. Згодом подібні системи досягнули значної продуктивності у вирішенні завдань на виправлення граматики CoNLL-2013 та CoNLL-2014 [15], [16], [17].

Для покращення результатів GEC систем з SMT підходом крім використання загальних методів машинного перекладу були також запропоновані рішення, які були направлені виключно на кращі результати у завданні на виправлення граматики.

Відстань Левенштейна на рівні символів та слів використовуються як фічі з глибоких шарів, через яку можливо моделювати відношення між виходовими та входовими реченнями, а особливо фіксувати операції редагування, тобто корекції [17].

Розширення системи Moses, яка займається машинним перекладом, новою операцією (OSM). При перекладі речення правильні конструкції збігаються, а ті слова, які є різними на виході та вході, є замінами, вставками і видаленнями [18].

Мовна модель класу слова (WCLM) була розроблена завдяки створенню 200 слів у виді класів інструментом Word2vec [19]. Набір слів був витягнутий з проаналізованого монокорпусу Wikipedia. Використання WCLM моделі дозволить фіксувати залежності на більшій відстані та семантичні аспекти в SMT моделях.

На основі нейронного машинного перекладу (NMT). Хоча підхід, заснований на SMT, отримує переваги від його здатності використовувати велику кількість паралельних даних та моно корпусів, він все ще страждає від відсутності контекстуальної інформації та обмеженої здатності до узагальнення. Як рішення, почали проводитись дослідження [23], [24] щодо

застосування підходу на основі NMT (Neural Machine Translation) для GEC завдання. Зі збільшенням продуктивності, отриманої моделями нейронного кодера-декодера у машинному перекладі, прийняті та модифіковані моделі на основі нейронних кодерів-декодерів. У порівнянні з системами GEC на основі SMT, моделі NMT мають дві переваги:

– модель нейронного кодера-декодера вивчає зв'язки від входового до виходового речення безпосередньо з процесу навчання на паралельних даних, а не з необхідних функцій у SMT для фіксації закономірностей;

– системи на основі NMT здатні виправляти набагато рідкісніші помилкові фрази та речення більш ефективно, ніж підходи на основі SMT, підвищуючи здатність до узагальнення.

RNN підхід. Вперше підхід на основі NMT моделі для вирішення завдання GEC був застосований в 2016 році [25]. У запропонованій роботі кодер кодує входове речення x = {x1, x2, … , xTx} як вектор v. Потім вектор передається в декодер для генерування корекції у через вираз (формула 1.1):

$$p(y) = \prod_{t=1}^{T} p(y_t | y_1, \dots, t-1, v). \qquad (1.1)$$

На кожному часовому кроці виходове слово передбачається з вектором та раніше згенерованими словами. Кодер та декодер є RNN – и (Recurrent Neural Network), які складаються з блоків GRU, або LSTM (Long short-term memory). Механізм Attention (механізм уваги) застосовується для покращення результату на кожному кроці декодування *i* шляхом вибіркового фокусування на найбільш відповідному контексті *ci* в оригінальному тексті. Прихований стан в декодері обчислюється за допомогою наступного виразу (формула 1.2):

$$s_i = f(s_{i-1}, y_{i-1}, c_i), \tag{1.2}$$

де s_{i-1} – останній крок прихованого стану;

y_{i-1} – слово, створене останнім кроком;

$f(\cdot)$ – нелінійна функція.

Змінна c_i обчислюється за формулою 1.3:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j, \tag{1.3}$$

де h_j – прихований стан кодера на кроці j.

Значення α_{ij} обчислюється як показано наступним виразом (формула 1.4):

$$\alpha_{ij} = \frac{e^{m_{ij}}}{\sum_{k=1}^{T_x} e^{m_{ik}}}. \tag{1.4}$$

Змінна m_{ik} є результатом відповідності між s_{i-1} і h_k.

CNN підхід. Система GEC на основі моделі NMT стала першою, яка перевершила SMT-подібні GEC системи [26]. Розроблено багатошарову згорткову архітектуру [27] і застосовано Attention на основі CNN для навчання «послідовніть до послідовності» (sequence-to-sequence learning). У кожному шарі кодера оригінальне речення спочатку представлене з допомогою вбудовування слів та вбудовуваної позиції як $S \in \mathbb{R}^{\mathrm{h}\times|\mathrm{S}|}$, де $|S|$ – кількість токенів у входовому реченні, а потім лінійно перетворюється в $H^0 \in \mathbb{R}\mathrm{h} \times |\mathrm{S}|$ перед подачею в перший шар кодера. Вихід l-го шару кодера розраховується наступним чином (формула 1.5):

$$H^l = GLU\left(Conv(H^{l-1})\right) + H^{l-1}, \quad (1.5)$$

а вихід кінцевого шару кодера H^l перетворюється у вихід кодера $E \in \mathbb{R}^{d \times |S|}$.

Під час декодування, вбудовування для згенерованих виходових n слів $T \in \mathbb{R}^{d \times n}$ обчислюється так само, як і S. l-ий рівень декодера обчислює проміжне представлення (формула 1.6):

$$Y^l = GLU\left(Conv(G^{l-1})\right), \quad (1.6)$$

який використовується для розрахунку Attention (формула 1.7):

$$\begin{gathered} Z^l = Lin(Y^t) + T, \\ X^l = (E + S) \times SoftMax(E^T \times Z^l), \\ C^l = Lin(X^l). \end{gathered} \quad (1.7)$$

Тоді вихід l-го шару декодера розраховується наступним чином (формула 1.8):

$$G^l = Y + C^l + G^{l-1}. \quad (1.8)$$

Матриця GL лінійно перетворюється в $D \in \mathbb{R}^{d \times n}$. Останній стовпець D потім зіставляється з розміром словникового запасу, щоб передбачити слово $(n + 1)$.

Перевагою моделей на основі CNN перед RNN-подібними моделями полягає в тому, що CNN більш ефективно фіксує локальний контекст і тим самим виправляє ширший спектр граматичних помилок. Довгострокова контекстна інформація може бути охоплена багаторівневою архітектурою.

Ще одна перевага CNN полягає в тому, що на відміну від RNN, де кількість операцій нелінійності над вихідними реченнями збільшується лінійно в міру довжини речень, буде проводитися лише фіксована кількість нелінійних операцій, і таким чином можна використовувати більше семантичної інформації (рисунок 1.10).

Трансформер. Після виходу публікації «Attention is all you need» («Увага це все, що нам потрібно») [28], багато систем GEC на основі NMT замінили традиційний кодер-декодер на основі RNN з назвою Transformer [29], [30], [31].

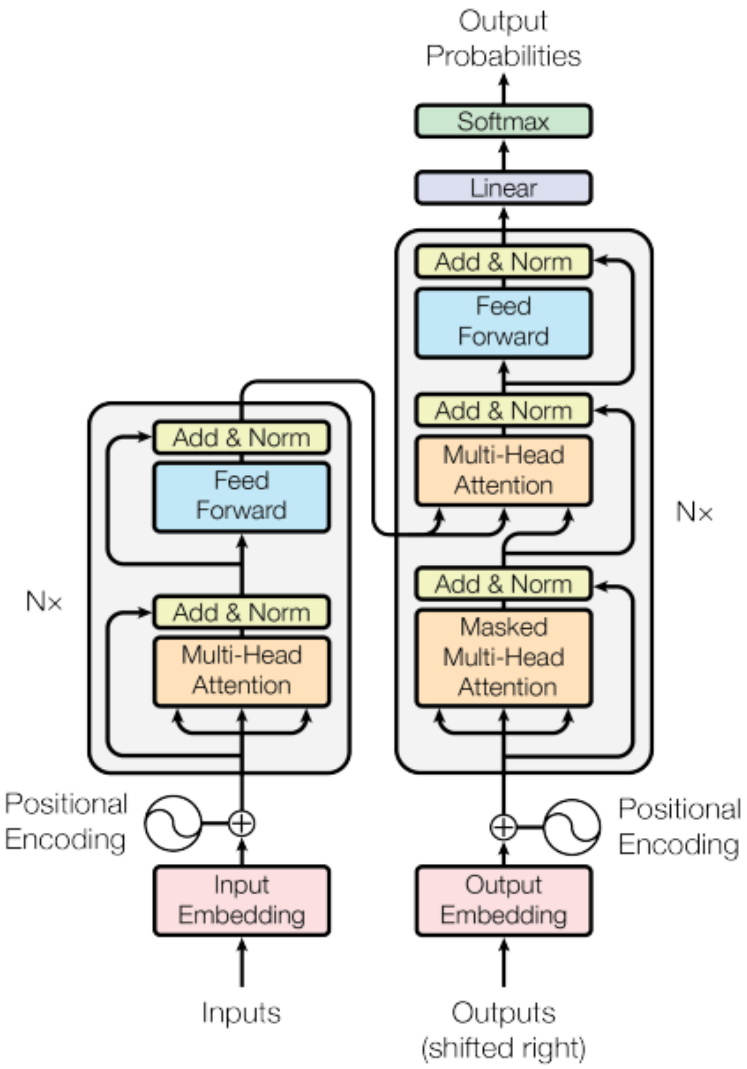


Рисунок 1.10 – Архітектура моделі Transformer [28]

Трансформер спочатку кодує входове речення у прихований стан за допомогою нашарування з кількох ідентичних блоків, кожен із яких складається з шару множинної самоуваги (multi-head self-attention layer) та шару прямого поширення (forward layer). Шар множинної самоуваги або, як його ще називають, масштабована сумарна самоувага (Scaled Dot-Product attention), розраховується за формулою 1.9:

$$Attention(Q, K, V) = SoftMax\left(\frac{QK^T}{\sqrt{d_k}}\right). \quad (1.9)$$

де Q, K, V представляють матрицю запиту, ключову матрицю та матрицю значень, відповідно, і обчислюються шляхом лінійного перетворення на входових векторах.

Змінна d_k – це розмірність векторів-стовпців у Q і K. Декодер має ту ж архітектуру, що кодер, але з додатковим шаром самоуваги, який додається до прихованого стану (рисунок 1.11).

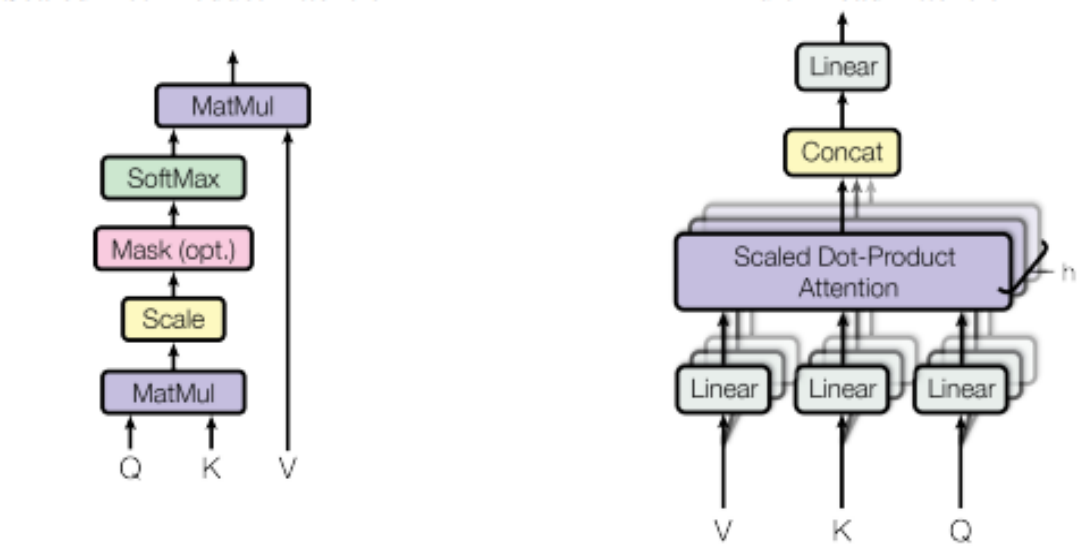


Рисунок 1.11 – (зліва) Scaled Dot-Product Attention, (справа) Multi-Head Attention складається з кількох паралельно робочих рівнів Attention [28]

У порівнянні з традиційними рекурентними мережами (GRU або LSTM), однією з переваг шару самоуваги в Transformer є те, що він дозволяє виконувати паралельні обчислення для скорочення часу витраченого на навчання моделей GEC. Незважаючи на те, що Transformer зчитує речення без інформації про порядок слів, позиційне кодування може значною мірою сприяти навчанню такої репрезентації.

На основі мовних моделей (LM). Методики, які використовують моделі SMT та NMT, вимагають великих обсягів паралельних даних для навчання, які містять неправильні та виправлені речення. Їх досить складно отримати, особливо для менш вживаних мов, тому був запропонований підхід на основі мовних моделей (LM – Language Model). Його найбільшою перевагою є те, що використовуючи невеликий за розміром анотований корпус можна також досягти високих результатів. В результаті, системи GEC на основі LM переважно розробляються для мов з невеликою кількістю ресурсів.

Важливим принципом в LM підході є використання окремих згенерованих даних «плутанин», які базуються на викривленні оригінальних речень, для прикладу:

– підміною слів їх неправильними версіями;

– додавання морфологічних помилок;

– зміна артиклів та прийменників. Для правильних підмін використовуються відповідні онлайн-словники.

Одна з робіт [22], яка зробила переоцінку LM методики, пояснює головну ідея мовного моделювання в GEC. Вона полягає в тому, що речення з низькою ймовірністю, швидше за все, містять граматичні помилки, ніж речення з високою. І мета такої GEC моделі полягає у визначенні як правильно здійснити перетворення, тобто виправлення, на основі LM ймовірностей.

У реалізації використовується спрощений алгоритм вже згаданого автора з попередніх робіт [12], який складається з наступних кроків:

1. обчислити нормовану логарифмічну ймовірність вхідного речення;

2. побудувати набір з «плутанин» для кожного токена в реченні;

3. повторно оцінити речення, замінюючи кожного кандидата в кожній групі з «плутанин»;

4. повторити кроки 1-4.

На основі зміни операцій. Окрім моделей прямого перекладу речень, розглянутих вище, відносно свіжим трендом стало застосування операцій редагування входових речень для виправлення граматичних помилок. Першою такою є модель з назвою PIE [32], яка навчена позначати входові токени як необхідні до редагування, включаючи операції копіювання, видалення, заміну 2-грамовою послідовністю, додавання 2-грамової послідовності та морфологічні трансформації. Необхідне навчання забезпечується шляхом порівняння входових та виходових речень з паралельних даних. PIE модель базується на архітектурі BERT з додаванням додаткових входів та механізму самоуваги для кращого прогнозування заміни та додавання.

Супутньою роботою є розробка системи GECToR («Grammatical Error Correction: Tag, Not Rewrite») [33], яка на сьогоднішній день є однією з передових у вирішені завдання GEC (третє місце в завданні CoNLL-2014 станом на травень 2022 року). Головними особливостями стало використання спеціальних граматичних трансформацій над токенами та розбивання етапу тонкого налаштування на два кроки: спочатку на тільки помилкових реченнях і в подальшому на невеликому високоякісному наборі даних, що містить як помилкові, так і безпомилкові речення. З покращеним підходом [34] до тегування послідовностей система GECToR змогла на цей момент досягнути першості у вирішенні завдання на виправлення граматики в англійській мові BEA-2019.

1.5 Текстові набори даних

Українська мова – це мова з розвиненою морфологією. На відміну від англійської, кожне слово тут має багато словоформ («книга», «книгою», «книгами»). Методи NLP, розроблені для англійської, не завжди будуть оптимальними для української. Потрібно оглянути спектр доступних датасетів, які є у публічному доступі. На рисунку 1.12 зображено розвиток розвиток у створенні моделей які працюють для інших мов.

Рисунок 1.12 – Хронологія розвитку іноземних моделей

Основними загальнодоступними наборами даних, які використовуються для вивчення моделей GEC, є Lang-8, NUCLE, FCE, W&I + LOCNESS. Нещодавно був створений український GEC – UA-GEC який варто також взяти до уваги. Вище перечисленні набори даних складаються з паралельних пар помилкових і граматично правильних речень.

Lang-8 Corpus of Learner English – це набір даних, заснований на даних Lang-8, веб-сайт з вивчення мови онлайн, де носії мови виправляють граматику в есе опубліковано тими, хто вивчає мову. Цей набір даних містить величезну кількість навчальних зразків GEC. Проте речення в цьому

наборі даних були виправлені анотаторами з різними рівнями кваліфікації, що вплинуло на якість даних. Важливим аспектом є те, що він містить незначні дані українською мовою.

UA-GEC – перший анотований GEC-корпус української мови. Корпус містить в собі колекцію текстів, що написали звичайні люди: есеї, дописи в блогах та соц. мережах, відгуки, листи тощо. Нариси розподілені на набори для підготовки (train) та тестування (test).

NUS Corpus of Learner English (NUCLE) є першим набором даних GEC, який вільно доступний для дослідницьких цілей. NUCLE складається з есе, написаних студентами Національного університету Сінгапуру (NUS). Нариси були написані як курсові завдання на різні теми, як-от технологічні інновації чи охорона здоров'я. Набір даних був анотований викладачами NUS.

The First Certificate in English Corpus (FCE) є загальнодоступною частиною приватного Cambridge Learner Corpus (CLC), і це набір речень, написаних англійською мовою. Студенти, які вивчають мову, відповідають на екзаменаційні запитання FCE для верхнього середнього рівня. FCE містить різноманітні знання про речення та широкі теми.

Write Improve Corpus (WI) і LOCNESS Corpus – нові набори даних. WI складається з 3600 анотованих есе від Write Improve, онлайн веб-платформа, яка допомагає студентам, які не є носіями англійської мови, писати. Нариси розділені на набори для підготовки (train), розробки (development) та тестування (test) з 3000, 300 і 300 зразками.

LOCNESS Corpus – це збірка з близько 400 есе, написаних британськими та американськими студентами. Він містить лише набори для розробки (development) та тестування (test).

1.6 Використання аугментованих даних

Аугментовані дані – це інформація, яка створюється штучно, а не генерується реальними подіями. Ці дані генеруються за допомогою комп’ютерного алгоритму та за рахунок певних математичних перетворень.

Дослідження використання штучних даних показують, що це може бути так само добре або навіть краще для навчання моделей машинного навчання, ніж дані, засновані на реальних об’єктах, подіях або людях.

Герерація додаткових аугментованих даних в першу чергу обумовлена тим, що не завжди є необхідна кількість інформації, або дані є спотвореними. Також, на етапі розробки можуть виникати проблеми пов’язані з тим, що математична модель починає вироджатись, тобто вона звикає до певних даних, або для підвищення якості моделі треба додати нові дані, а часу на додаткову розмітку «вручну» часу не вистачає. Для цього, ми можемо зібрати усі дані які нам необхідні, і на їх основі генерувати штучні дані, які будуть використані для тренування моделей. Для того щоб створювати аугментовані дані, варто використовувати Python. Python є однією з найпопулярніших мов, особливо для науки про дані.

1.7 Постановка задачі дослідження

Метою кваліфікаційної роботи є розробка інтелектуальної системи виправлення граматичних помилок для української мови на основі глибинних рекурентних нейронних мереж з використанням анотованих та синтетичних наборів даних.

Об’єктом дослідження є процес виправлення граматичних помилок в українській мові на основі українського анотованого корпусу з додатковим використанням синтетичних даних.

Предметом дослідження є глибинні рекурентні мережі, призначені для вирішення задач виправлення граматичних помилок, що навчаються як на анотованих, так і на синтетичних наборах даних.

Методи дослідження – теорія оптимізації, теорія штучних нейронних мереж, теорія глибинних нейронних мереж, теорія систем опрацювання природної мови.

Провівши попередній аналіз предметної галузі, огляд даних для навчання та порівняння існуючих підходів для вирішення задачі на виправлення граматичних помилок, сформовано подальші напрямки досліджень та визначені основні завдання:

– оглянути основні методи та їх порівняння для навчання системи GEC на різних етапах розробки, які включають в себе представлення слів у векторі, методи трансферного навчання, методи оптимізації гіперпараметрів та метрики оцінювання якості системи GEC;

– порівняти існуючі GEC системи для української мови з метою виявлення їх недостатків на основі використаних підходів та розглянути детальніше підхід нейронного машинного перекладу при реалізації системи GEC;

– проаналізувати типи та стратегії генерування синтетичних даних з метою отримання кращих результатів при навчанні моделі;

– провести попередню обробку даних та застосувати розглянуті методи генерування синтетичних даних з метою отримання різних наборів даних для навчання;

– за допомогою вищезгаданих методів реалізувати GEC системи для української мови з використанням як і виключно анотованих, так і синтетичних даних. З метою отримання порівняльних результатів навчання проводитимуться на групі моделей з різною кількістю параметрі;

– провести порівняльний аналіз усіх отриманих результатів з системами GEC для інших мов з невеликою кількістю ресурсів й визначити майбутні напрями дослідження з метою покращення якості системи.

2 МЕТОДИ ДЛЯ НАВЧАННЯ СИСТЕМИ ВИПРАВЛЕННЯ ГРАМАТИЧНИХ ПОМИЛОК

2.1 Векторизація слів

Тривалий час для розуміння комп'ютером людської мови використовувались відносно прості у створенні чисельні репрезентації для слів. Найпростішим є метод нормалізації даних – One-hot кодування, який використовує принцип категоріальних ознак (фіч) у представленні слова як окремого вектора з нулів та одиниці. Розмір вектора залежить від словника, який створюється в процесі застосування алгоритму. На рисунку 2.1 наведено приклад представлення речення у найпростішій векторній формі [18].

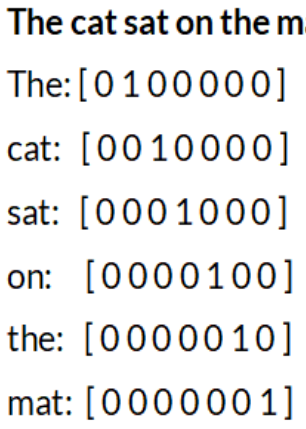


Рисунок 2.1 – Приклад кодування речення у векторну форму

Цей метод кодування слів має значні недоліки в складності до масштабування та відсутності будь-яких синтаксичних чи контекстуальних зв'язків між словами. Вирішити цю проблемо можна завдяки використанню методу вбудовування слів (word embeddings).

Модель word2vec була розроблена Міколовим в 2013, головна ідея якої представлення слів у формі векторів з вбудованим локальним

контекстом [19]. Його архітектура це плоска двошарова прихована нейронна мережа. Модель Word2vec приймає на вхід великий корпус тексту й виробляє векторний простір, зазвичай з кількома сотнями вимірів, де кожному унікальному слову з цього корпусу призначено відповідний вектор у цьому просторі. Векторні представлення слів розташовуються в цьому векторному просторі таким чином, що слова, які поділяють спільний контекст у корпусі, розташовуються близько одне до одного в цьому просторі. На рисунку 2.2 зображено приклад векторного простору [18].

Модель word2vec створює векторний простір за допомогою 2 методів:

1) неперервна торба слів, або CBOW (Continuous-bag-of-words) передбачає середнє слово в контексті навколишніх, тобто намагається заповнити пропуски словами, які найбільше підходять. Він є більш ефективний з меншими наборами даних;

2) неперервний пропуск-грам (Continuous skip-gram) намагається передбачити навколишні контекстні слова з цільового слова. Цей метод має тенденцію відпрацьовувати краще з більшими об’ємами даних, однак відповідно збільшується час на тренування.

Яскравим прикладом ефективності моделі word2vec є можливість за допомогою простих арифметичних операцій над векторами отримувати очікувані результати, спираючись на людське розуміння значень слів. На рисунку 2.3 зображено приклад того, як працює ця модель векторизації текстових даних.

Таким чином, головними перевагами моделі word2vec є:

– здатність фіксувати синтаксичні та семантичні зв’язки між словами;

– гнучкість розміру вектора з можливістю легкого масштабування;

– навчання відбувається без вчителя, що дозволяє уникнути певних людських зусиль в анотуванні даних.

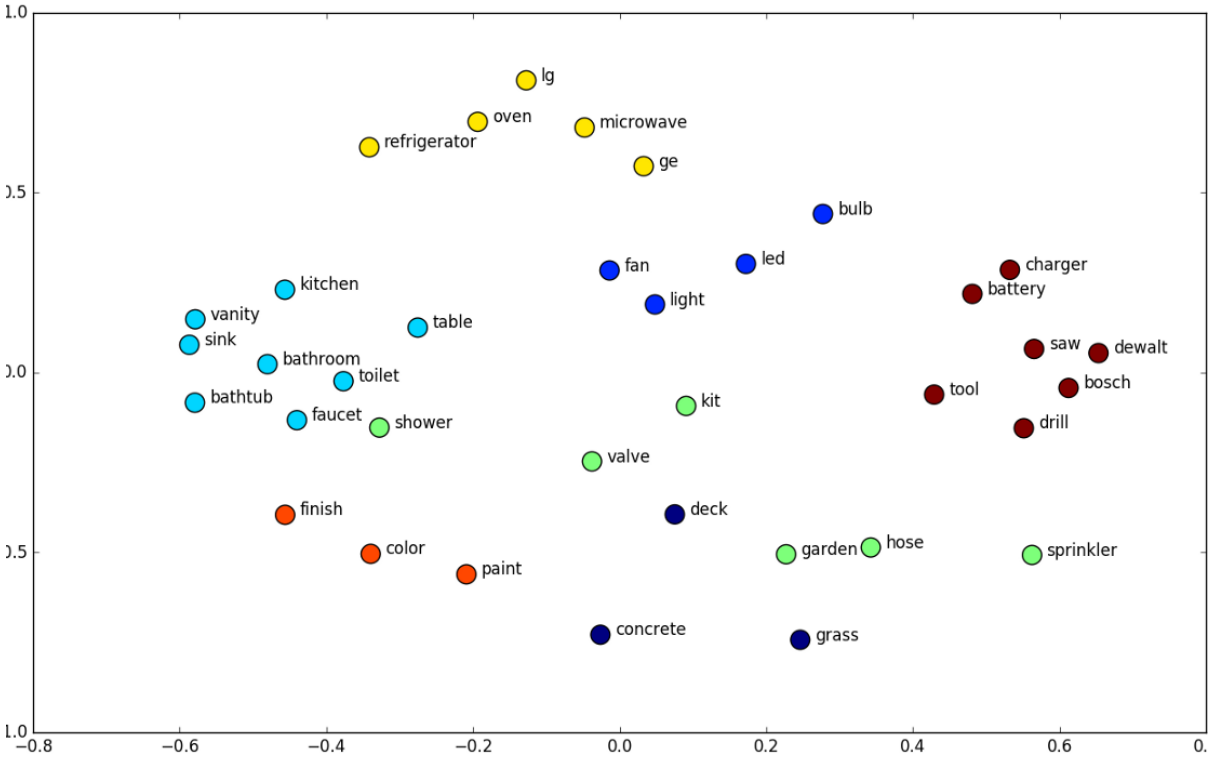


Рисунок 2.2 – Векторний простір слів

$$\text{Let } \phi \text{ be a word embedding mapping } W \to \mathbb{R}^n$$
$$\text{where W is the word space and}$$
$$\text{R}^n \text{ is an n dimensional vector space then:}$$

$$\phi(\text{"}king\text{"}) - \phi(\text{"}man\text{"}) + \phi(\text{"}woman\text{"}) = \phi(\text{"}queen\text{"})$$

Рисунок 2.3 – Арифметичні дії над векторними представленнями слів, які в результаті породжують новий векторний простір даних

Однак існує проблема неможливості однозначно визначити значення слова, яке опирається виключно на обмежене коло сусідніх пар слів. Також навчену модель word2vec неможливо використовувати для інших «нових» даних, тобто використання трасферного навчання в цій моделі відсутнє. Для використання моделей word2vec їх треба кожного разу навчати з нуля.

2.2 Контекстуалізована векторизація слів на основі моделі ELMo

Традиційний метод векторизації слів, як word2vec, враховує всі речення, де є ключове слово, щоб створити для нього глобальне векторне представлення. Однак певні слова можуть мати абсолютно різні значення в контекстах. Наприклад, розглянемо два речення: «Князь із дружиною вирушає в похід» та «Дружина Івана Франка була громадською діячкою». Значення слова «дружина» в обох реченнях відрізняються одне від одного, тому що в першому випадку ми маємо на увазі військо. Для вирішення цієї проблеми логічно було б мати по кілька векторів на кожне слово. Сучасна концепція мовних моделей приймає це міркування та відходить від принципу глобального представлення слів у бік контекстного.

Векторизація мовної моделі (ELMo) [20] є одним із таких методів, який реалізує глибокі контекстні вбудовування слів. ELMo генерує такі представлення для кожного контексту, де використовується слово, таким чином дозволяючи його різне розуміння.

Модель ELMo використовує представлення, отримані з двонаправленої мовної моделі (biLM), яка складається з двох компонент:

1) мовна модель (Learning Model – LM), прямого поширення;

2) LM зворотного поширення.

LM прямого поширення приймає на вхід значення x_k^{LM} для кожного k^{th} токена та передає його через L шарів прямої рекурентної нейромережі для отримання значень $\vec{h}_{k,j}^{LM}$, де $j = 1, \dots, L$. Кожне з цих значень, будучи прихованими репрезентаціями рекурентних нейронних мереж, є контекстуально залежним. Пряму LM можливо розглянути як метод моделювання спільної ймовірності для послідовності токенів: $p\,(t_1, t_2, \dots, t_N) = \prod_{k=1}^{N} p(t_k | t_1, t_2, \dots, t_{k-1})$. На часовому кроці $k-1$ пряма LM передбачає наступний токен t_k з урахуванням попередніх токенів $t_1, t_2, \dots, t_k$. Це досягається завдяки додаванню функції активації типу SoftMax поверх останнього (виходового) шару LSTM в прямій LM.

З іншого боку, зворотна LM моделює аналогічну спільну ймовірність, передбачаючи попередній токен з урахуванням майбутніх: $p\ (t_1, t_2, \dots, t_N) = \prod_{k=1}^{N} p(t_k | t_{k+1}, t_{k+2}, \dots, t_N)$. Іншими словами, зворотна LM подібна до прямої LM, однак обробляє послідовності зворотним порядком. Навчання моделі biLM включає моделювання логарифмічної ймовірності для обох напрямів LM, після чого приховані представлення об'єднуються, щоб скласти кінцеві вектори токенів.

Для кожного токену, ELMo бере з шарів biLM проміжні представлення та виконує лінійну операцію, базуючись на кінцевій задачі. $\underline{L}$-шар в biLM містить $2L + 1$ набір представлень, як показано нижче (формула 2.1):

$$R_k = \{x_k^{LM}, \vec{h}_{k,j}^{LM}, \overleftarrow{h}_{k,j}^{LM} | j = 1, \dots, L\} = \{h_{k,j}^{LM} | j = 0, \dots, L\}, \tag{2.1}$$

де $h_{k,0}^{LM}$ це представлення токена на найнижчому рівні.

Для інших значень j це

$$h_{k,j}^{LM} = \left[\vec{h}_{k,j}^{LM}, \overleftarrow{h}_{k,j}^{LM}\right] \forall_j = 1, \dots, L. \tag{2.2}$$

ELMo об'єднює всі шари в один вектор (формула 2.3):

$$ELMo_k^{task} = E(R_k; \sigma^{task}) = \Upsilon^{task} \sum_{j=0}^{L} s_j^{task} h_{k,j}^{LM}. \tag{2.3}$$

У рівнянні (2.3), s_j^{task} є softmax-нормованим вектором ваги для поєднання представлень різних шарів. Υ^{task} це окремий гіперпараметр, який допомагає в оптимізації та масштабуванні конкретного завдання представлення ELMo. Отже, ELMo створює різноманітні представлення для одного слова в різних реченнях. В окремому дослідженні [37] було доведено, що вигідним є поєднання представлення слів ELMo зі

стандартними глобальними представленнями слів як word2vec чи Glove, що є дещо покращеною версією word2vec.

2.3 Модель BERT

Нейромережева архітектура, яка має назву трансформер [38], має багато переваг у порівнянні зі стандартними глибинними рекурентними мережами (LSTM, RNN, GRU). Однією з них є більш ефективне моделювання довгострокової пам’яті між токенами в тимчасових послідовностях та ефективніше навчання моделі шляхом усунення послідовної залежності від попередніх токенів. Модель BERT, яка була запропанована в [21], використовує архітектура трансформерів для попереднього навчання мовної моделі з метою вилучення контекстних представлень слів. На відміну від ELMo, BERT виконує інші підходи для попереднього навчання. В одному із них BERT випадково маскує певний відсоток слів у реченні й потім намагається передбачити лише ці замасковані слова.

В іншому підході, BERT передбачає наступне речення за допомогою попереднього. Мета полягає у моделюванні відносин між двома реченнями, які не вираховуються традиційними biLM. Це дозволяє моделі BERT перевершити інші мережі та системи у завданні «питання-відповідь» та інших, де розуміння взаємозв’язку між двома реченнями є дуже важливим.

2.4 Вирішення завдання GEC методом нейронного машинного перекладу

Сучасні підходи часто інтерпретують завдання виправлення граматичних помилок як одномовне переписування одного тексту в інший, а саме помилковий у граматично та синтаксично коректний. Цю проблему можливо розглянути як завдання нейромережевого машинного перекладу,

яка за основу використовує сімейство підходів послідовного навчання (Seq2seq), реалізації яких будуть далі проаналізовані.

Модель Seq2seq приймає на вхід послідовність елементів (слів, букв, ознак зображення тощо) і виводить іншу послідовність елементів. У нейромережевому машинному перекладі послідовність – це серія слів, оброблених одне за одним.

Архітектура моделі складається з енкодера і декодера. Енкодер обробляє кожен елемент у входовій послідовності та перетворює інформацію, яку отримує, у вектор-контексту. Після опрацювання всієї послідовності енкодер відправляє контекст декодеру, який починає створювати виходову послідовність елемент за елементом. На рисунку 2.4 зображено схематично послідовність перетворення входового вектола до виходового [22].

Рисунок 2.4 – Передача контексту в моделі енкодер-декодер

Енкодер та декодер є рекурентними глибинними нейронними мережами, а контекст представлений окремим вектором, розмір якого відповідає кількості прихованих одиниць в енкодері RNN. Як правило ці значення відповідають розмірам 256, 512, або 1024.

По типу своєї архітектури RNN приймає два входи на кожному кроці навчання: вхід (у випадку енкодера, одне слово з входовго речення) і прихований стан. Слова повинні бути представлені як вектори, використовуючи механізм вбудовування слів (word embedding). На наступному кроці навчання RNN приймає другий вектор на вхід і прихований стан 1, щоб створити інший виходовий результат. Таким чином,

на кожному кроці RNN навчання прихований стан постійно оновлюється, що власне і являється контекстом, який передається в декодер. Декодер використовує цей прихований стан, який теж передається з одного кроку навчання до наступного.

Однак залишається проблема неможливості передачі якісного вектору контексту декодеру, а саме в умовах опрацювання довгих послідовностей, тобто при опрацювання великих даних. При утворенні енкодером контексту він з кожним кроком все більше вироджається та втрачає попередні властивості, після чого ця версія контекстного вектору передається на вхід декодеру. Декодеру при цьому набагато складніше розпізнати потрібний контекст та згенерувати правильну послідовність. Вирішенням цієї проблеми може стати підхід на основі уваги, яка здатна значно покращити якість систем нейромережевого машинного перекладу. Механізм уваги дозволяє моделі зосередитися на відповідних частинах входової послідовності, якщо це необхідно. На рисунку 2.5 схематично представлено яким чином впливає механізм уваги на кінцевий результат [26].

По-перше, енкодер передає декодеру набагато більше даних. Замість передачі останнього прихованого стану етапу кодування, енкодер передає всі стани декодеру. По-друге, декодер уваги робить додатковий крок, перш ніж згенерувати вихід. Для зосередження на різних частинах входової послідовності, які мають відношення до певного кроку під час декодування, декодер виконує наступні кроки:

– розглянути набір прихованих станів енкодера, який він отримав, кожен прихований стан енкодера найбільше асоціюється з певним словом у вхідному реченні;

– надати кожному прихованому стану ваговий коефіцієнт;

– помножити кожен прихований стан на його SoftMax оцінку, таким чином підсилюючи приховані стани з великою вагою і заглушаючи приховані стани з малою вагою.

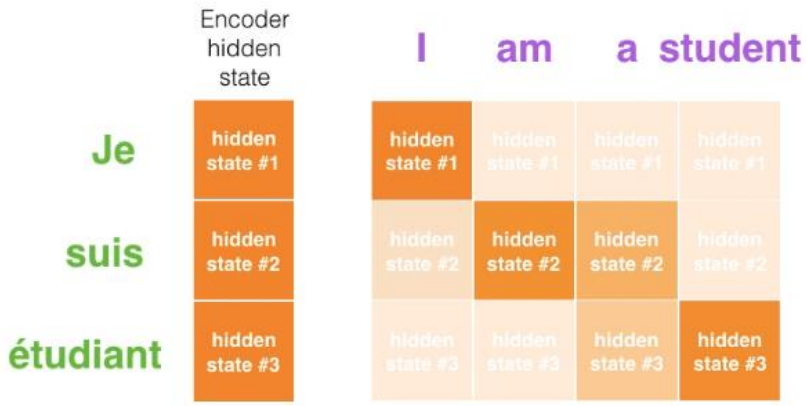


Рисунок 2.5 – Вплив механізму уваги на результати нейронного машинного перекладу

Доречним буде зазначити, що на етапі навчання мережа опановує спосіб як правильно поєднати мовні пари між різними мовами (входові послідовності енкодера та декодера), тому що порядок слів при перекладі може змінюватись. Прикладом того, наскільки точним може бути механізм уваги, наведено схематично на рисунку 2.6 [26].

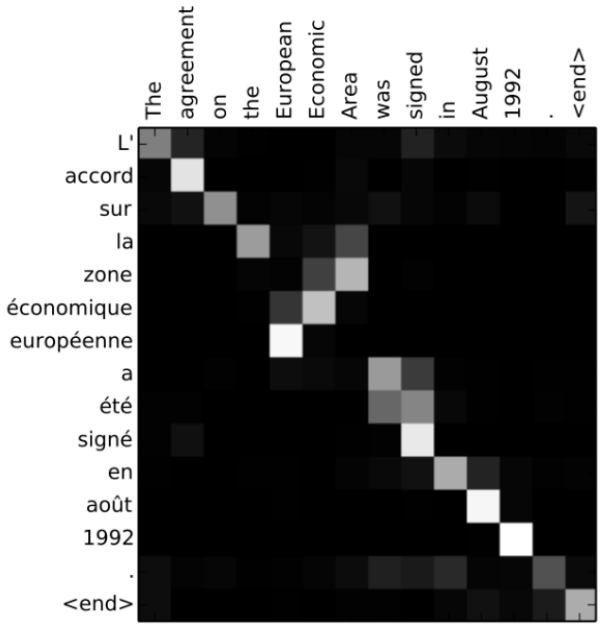


Рисунок 2.6 – Приклад моделі на основі механізму уваги

2.5 Домена адаптація моделей

Метод трансферного навчання – це техніка, де глибинна нейронна мережа, навчена на великому наборі даних, використовується для виконання подібних завдань на іншому наборі даних. Таку модель глибинного навчання називають попередньо навченою (претренованою). Найвідомішими прикладами попередньо навчених моделей є моделі згорткових нейронних мереж для опрацювання зображень та відео потоків, навчені на наборах даних, таких як ImageNet, VOC та COCO. В галузі опрацювання природньої мови (текстових даних) трансферне навчання набуло великої популярності з приходом моделі BERT, яка з її виходом була попередньо натренована у двох варіантах:

1) $BERT_{BASE}$ зі 110 мільйонами параметрів;

2) $BERT_{LARGE}$ з 340 мільйонами параметрів.

Обидві моделі було натреновано на великих наборах даних BooksCorpus з 800 мільйонами слів, та на одній з версій англійської Вікіпедії з 2500 мільйонами слів.

Досить складно, а в деяких випадках взагалі неможливо самостійно натренувати велику мовну модель, тому простіше взяти вже попередньо навчену як відправну точку для вирішення специфічного завдання. При цьому час на тренування значно скорочується, мережа навчається якісніше, та як правило вимагає набагато менші обсяги даних. На рисунку 2.7 схематично зображено як відбувається трансферне навчання [27].

Так як архітектура попередньо навченої мережі як правило складається з великої кількості шарів, процес подальшого навчання, який називається тонким налаштуванням гіперпараметрів мережі (Fine-tuning), може мати різні стратегії.

Тренування всієї мережі – додатково навчити всю попередньо навчену модель глибинної нейромережі на відповідному наборі даних та передавати виходові дані на рівень SoftMax. У цьому випадку метод зворотного

поширення помилки буде працювати через всю архітектуру, тому попередньо навчені вагові коефіцієнти мережі будуть оновлюватись протягом всього процесу навчання.

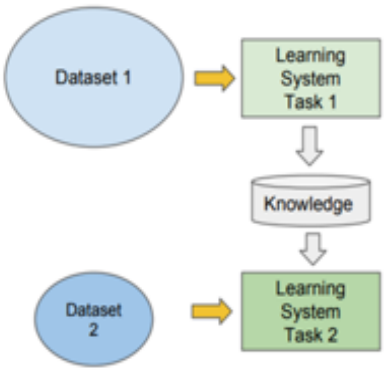


Рисунок 2.7 – Схематичне представлення трансферного навчання

Часткове тренування мережі – навчити модель частково. Заморожуються вагові коефіцієнти початкових шарів моделі і перенавчаються лише останні шари (наприклад, на практиці дуже часто заморожують всі шари, крім сіми останіх). Для різних результатів є доцільним спробувати різні комбінації кількості заморожених шарів і тих, які будуть навчатись.

Заморозити всю мережу – можливо заморозити всі шари моделі та приєднати кілька власних шарів нейронної мережі та навчити цю модель на оновленій архітектурі. Лише ваги приєднаних шарів будуть оновлюватися під час навчання моделі.

2.6 Налаштування гіперпараметрів з використанням градієнтних методів оптимізації

Для керування процесом навчання моделі використовують окремий набір параметрів, які ще називають гіперпараметрами. На відміну від інших

параметрів (наприклад вагових коефіцієнтів), які потрібно оптимізувати, гіперпараметри задають на початку тренування. Тому важливо знати наперед, яка комбінація з цих параметрів дасть найкращий результат. Для оптимального їх знаходження використовують методи оптимізації гіперпараметрів, які здатні вирішити проблему автоматичного підбору найкращих в сенсі результативності мережі параметрів.

Методи градієнтного або, як ще їх називають, методи найшвидшого спуск – це процес отримання найменшого значення похибки на основі ітераційного розрахунку значень антиградієнтів функції втрат, які необхідно налаштувати.

Процес навчання відбувається наступним чином (формула 2.5):

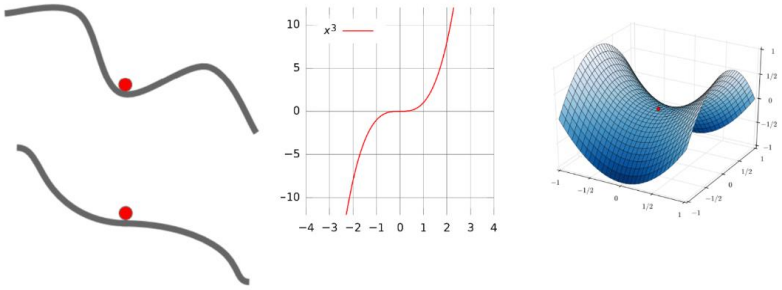
$$x(k+1) = x(k) - \eta(k)\nabla_x f\big(x(k)\big), \tag{2.5}$$

де $\eta(k)$ – параметр кроку навчання (learning rate parameter);

$\nabla_x f(x(k))$ – вектор градієнтів параметрів, які треба налаштувати.

Головним недоліком класичного методу градієнтного спуску є те, що цей метод може «застрягти» в одному з локальних екстремумів, це трапляється коли функція має форму складної кривої, як зображено на рисунку 2.8 (а).

Також при застосуванні таких методів навчання виникає проблема з сідловими точками, коли в одному напрямку функція зростає, а в іншому – зменшується. Найпростіший приклад двовимірної сідлової точки – нуль на графіку y = x3. У тривимірному просторі утворюється поверхня, яка нагадує сідло, через що і з'явилася така назва. При попаданні в сідлову точку метод найшвидшого спуск загальмовується потрапивши в локальний мінімум. І незалежно від числа подальших ітерацій похибка практично не зменшиться. Схематичний вигляд функції з сідловою точкою зображено на рисунку 2.8 (б).



а) б)

Рисунок 2.8 – Проблеми навчання, при використання методу градієнтного спуску

При використанні класичного методу градієнтного спуску також можливі проблеми через забруднені дані, які можуть у великій кількості знаходитесь в пакеті зразків, що зазвичай використовується для стохастичного градієнтного спуску.

Adam – це адаптивний алгоритм оптимізації, який можна використовувати замість класичної процедури стохастичного градієнтного спуску для ітераційного оновлення вагових коефіцієнтів мережі на основі навчальних даних. Якщо стохастичний градієнтний спуск завжди використовує одне і те ж значення кроку навчання (learning rate) для всіх оновлень вагових коефіцієнтів, і воно не змінюється в процесі навчання, то в алгоритмі Adam швидкість навчання є різною для кожного вагового коефіцієнта (параметра) моделі, значення яких постійно підналаштовується.

Алгоритм Adam спочатку оновлює експоненціальні ковзні середні градієнта (m_t) і квадрата градієнта (v_t), які є оцінками першого та другого моменту. Гіперпараметри β1, β2 ∈ [0, 1) контролюють експоненціальну швидкість спаду цих ковзних середніх (формула 2.6):

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t,$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2, \quad (2.6)$$

Ковзні середні ініціалізуються як 0, що призводить до оцінок моменту, які зміщені приблизно до 0, особливо під час початкових ітерацій навчання мережі. Цьому зміщенню ініціалізації можна легко протидіяти, що призводить до оцінок, виправлених зміщенням (формула 2.7):

$$\widehat{m}_t = \frac{m_t}{1 - \beta_1^t}, \qquad (2.7)$$

$$\widehat{v}_t = \frac{V_t}{1 - \beta_2^t}.$$

Оновлення параметрів відбувається наступним чином (формула 2.8):

$$\theta_{t+1} = \theta_t - \frac{\eta \widehat{m}_t}{\sqrt{\widehat{v}_t + \varepsilon}}. \qquad (2.8)$$

На практиці, на сьогоднішній день Adam часто використовують як базовий алгоритм оптимізації, який вбудовано в глибинні нейронні мережі, які можна використовувати як «рішення з коробки», тому що завдяки проведеним дослідженням [30], [31] було доведено його ефективність. На рисунку 2.9 схематично зображено порівняння різних алгоритмів оптимізації [31], де за певну кількість ітерацій Adam досягнув нижчих значень функції втрат.

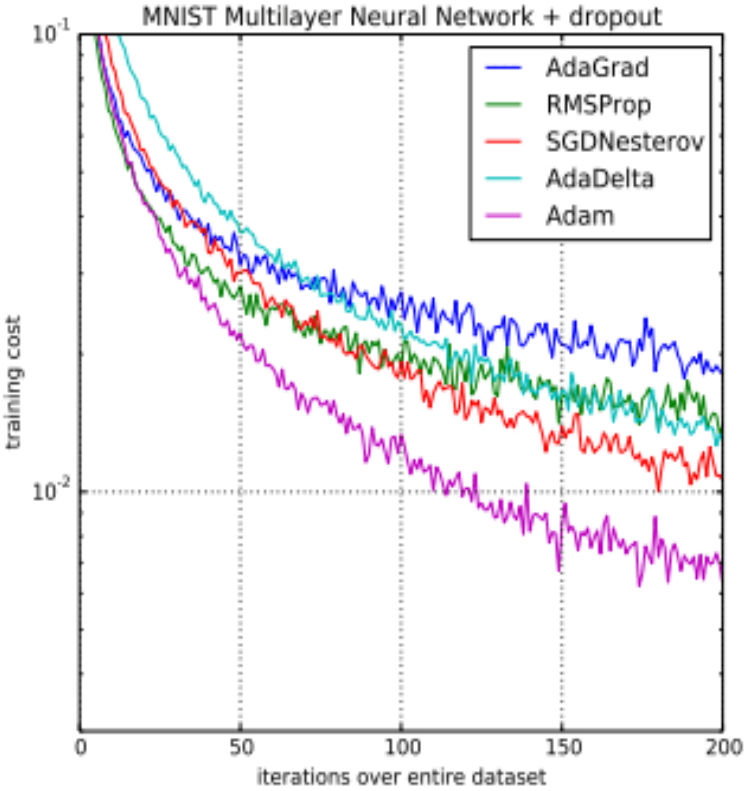


Рисунок 2.9 – Порівняння оптимізатора Adam з іншими оптимізаторами на основі градієнтного пошуку

2.7 Метрики оцінювання якості моделей виправлення граматичних помилок

BLEU (BiLingual Evaluation Understudy) – це метрика для автоматичного оцінювання систем для нейромережевого перекладу тексту. Незважаючи на те, що ця метрика розроблена для оцінки завдання перекладу тексту, її можна використовувати і для оцінювання різного роду згенерованого тексту (одномовний переклад), в тому числі і для систем і, що вирішують задачу виправлення граматичних та синтаксичних помилок. Значення 0 означає, що результат машинного перекладу не збігається з еталонним перекладом (низька якість), тоді як значення 1 означає, що є повна схожість з еталонним перекладом (висока якість).

Метрика працює на основі метрики Precision (точність), а саме розрахунку відповідних n-грам у перекладі-кандидаті до n-грам у довідковому тексті, де 1-грам або уніграм це кожний токен, а порівняння біграм – кожна пара слів. Після чого всі слова-кандидати, які були знайдені у довідковому тексті, діляться на загальну кількість слів у перекладі-кандидаті. В таблиці 2.1 наведено приклад оцінювання речення-гіпотези, використовуючи метрику BLEU.

Речення-гіпотеза: «А правда крилатим ґрунту не треба».

Довідкове речення: «А й правда, крилатим ґрунту не треба».

Таблиця 2.1 – Приклад оцінювання речення-гіпотези метрикою BLEU

Модель	Набір n-грам	Оцінка
Уніграм	«А», «правда», «крилатим», «ґрунту», «не», «треба»	$\frac{6}{6} = 1$
Біграм	«А правда», «крилатим ґрунту», «не треба»	$\frac{2}{3} = 0.67$

В деяких випадках може виникнути ситуація, що система при перекладі може зациклитись на одному слові й вивести його кілька раз, при тому кінцева оцінка буде високою, що не відповідає дійсності. Для вирішення цього процес розрахунку відповідних n-грам є модифікованим, аби переконатися, що він бере до уваги присутність слів у довідковому тексті, не винагороджуючи при цьому переклад-гіпотезу, який містить велику кількість зайвих слів. Це називається модифікованою n-грамовою точністю.

Хоч і BLUE є найпопулярнішою метрикою у галузі нейромережевого машинного перекладу, вона також має ряд недоліків:

– не враховую метрику Recall (повнота), через що не можливо визначити чи усі слова з входового речення були згадані в перекладеному;

– не розуміє семантики речень;

– складніше оцінити, якщо певні слова часто повторюються.

Метрика GLEU. Проблема застосування метрики BLUE пов’язана з тонкою, але важливою різницею між завданнями нейромережевого машинного перекладу та одномовного переписування тексту. У машинному перекладі неперекладене слово чи фраза майже завжди є помилкою, але при виправленні граматичних помилок це не так. Деякі, але не всі регіони входового речення слід змінити. Це спостереження наштовхнуло на невелику модифікацію BLEU, яка обчислює n-грамову точність щодо еталону, але призначає більшу вагу n-грамам, які були правильно змінені у входову реченні. Ця переглянута метрика, а саме загальне розуміння мовної оцінки (GLEU), винагороджує виправлення, а також правильно оцінює незмінний входовий текст [32].

Метрика m2. Ця метрика найчастіше використовуєтьяс для оцінки якості систем виправлення граматичних та синтаксичних помилок [33]. Метрика m2 покладається на анотації еталонного стандарту, у яких зазначені помилки в конкретних місцях. Зміни оцінюються відповідно до еталонних у мірі F_β. Припустимо, витягнутий набір еталонних змін для речення i дорівнює g_i, тоді як набір редагування системної гіпотези для речення i дорівнює e_i. R, точність P і F_β визначаються таким чином, як у формулі 2.9.

$$
\begin{gathered}
R = \frac{\sum_{i=1}^{n} |g_i \cap e_i|}{\sum_{i=1}^{n} |g_i|}, \\
P = \frac{\sum_{i=1}^{n} |g_i \cap e_i|}{\sum_{i=1}^{n} |e_i|}, \\
F_\beta = \frac{(1+\beta^2) \times R \times P}{R + \beta^2 + P}, \\
g_i \cap e_i = \{e \in e_i | \exists g \in g_i match(g, e)\},
\end{gathered}
\tag{2.9}
$$

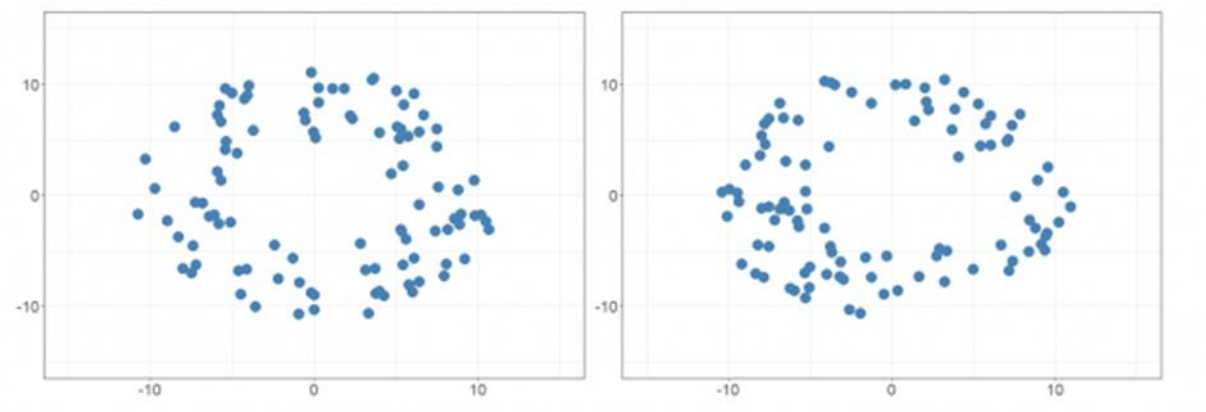
де $match(g, e)$ означає, що e і g мають однакове зміщення та поправку. $F_{0.5}$ наголошує на точності вдвічі більше, ніж повноті, а точність важливіша за повноту в GEC.

Це пояснюється тим, що бажаніше коли правками системної гіпотези будуть насправді виправлення граматичних помилок, аніж деякі правки виявляться помилковими виправленнями.

3 ВИКОРИСТАННЯ АУГМЕНТОВАНИХ ДАНИХ В ЗАДАЧАХ GEC

3.1 Аугментація даних

Аугментовані дані – це дані, які створені штучно, а не створені реальними подіями. Такі дані створюються за допомогою алгоритмів та математичних законів і використовуються для широкого спектру завдань пов’язаних зі створенням інтелектуальних систем опрацювання даних, в тому числі як тестові дані для нових продуктів та інструментів, для перевірки моделі та в навчанні моделі машинного навчання. Аугментовані дані є одним із методів надавання інваріантності моделі нейромережі. Аугментовані дані відрізняються від доповнених і випадкових даних – вони імітують реальні спостереження. На рисунку 3.1 зображено приклад реальних та аугментованих даних.



а) б)

Рисунок 3.1 – Генерація аугментованих даних: а) реальні дані; б) аугментованідані

Алгоритми машинного навчання вимагають великої кількості даних для обробки, щоб створити надійну модель. Без використання

додаткових (синтетичних) даних створити таку величезну кількість даних було б важко, але зі синтетичними даними це стає набагато простіше.

Генерування аугментованих даних забезпечує гнучкість налаштування їх природи та середовища, коли це потрібно, щоб покращити продуктивність моделі. Досягнути високої точності моделі на реальних даних, які треба додатково розмічати, а цей процес може привести до помилок в розмітці і займає багато часу, може дорого коштувати, тоді як навчання моделі на частково аугментованих даних даних може дати хорошим результатом та зменьшити час на тренування моделі. В деяких випадках відсоток аугментованих даних в навчальному наборі даних може досягати 80%, це пов’язано з тим, що є задачі, де знайти реальні дані майже не можливо.

3.2 Методи створення створення аугментованих даних

Визначаючи найкращий метод створення аугментованих даних, важливо спочатку визначити типи цих даних. На вибір є три широкі категорії, кожна з яких має свої переваги та недоліки. За своїм складом аугментовані дані можна розділити на три групи:

Повністю аугментовані дані – це дані є повністю штучно згенерованими і не мають нічого спільного з оригінальними даними. Це означає, що повторна ідентифікація будь-якої окремої одиниці майже неможлива. Генератор даних для цього типу зазвичай визначає функцію щільності об’єктів у реальних даних і оцінює їх параметри. Пізніше для кожної функції захищені конфіденційністю ряди генеруються на випадковій основі з оцінених функцій щільності. Якщо для заміни синтетичними даними вибирається лише кілька ознак реальних даних, то захищені ряди цих ознак відображаються на інші ознаки реальних даних, щоб охопити захищений ряд і реальний ряд у тому самому порядку. Кілька класичних методів, які використовуються для генерування повністю синтетичних

даних, можуть бути методами завантаження та множинними мутаціями. На рисунку 3.2 наведено основні типи аугментованих даних.

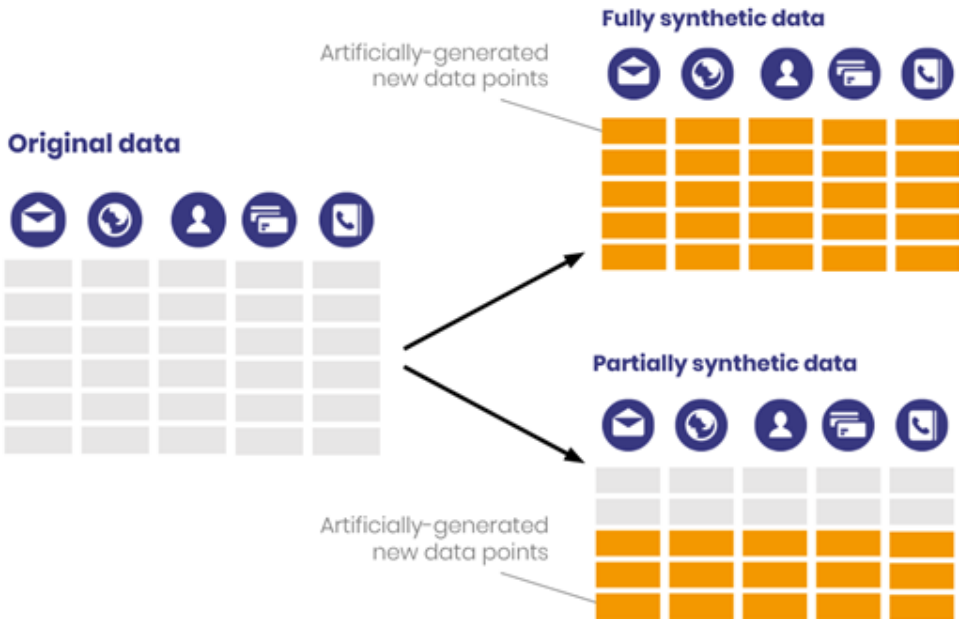


Рисунок 3.2 – Типи аугментованих даних

Частково аугментовані дані – ці дані замінюють лише значення деяких вибраних конфіденційних властивостей синтетичними значеннями. Реальні цінності в цьому випадку замінюються лише в тому випадку, якщо вони містять високий ризик розкриття. Це призводить до зменшення залежності моделі, але означає, що деяке розкриття інформації можливе завдяки справжнім значенням, які залишаються в наборі даних. Методи, що використовуються для генерування частково синтетичних даних, є множинними методами мутації та моделями. Ці методи також корисні для введення відсутніх значень у реальні дані.

Гібридні штучні дані – ці дані генеруються з використанням як реальних, так і синтетичних даних. Забезпечуючи взаємозв’язок і цілісність між іншими змінними в наборі даних, досліджується основний розподіл вихідних даних і формується найближчий сусід кожної точки даних. Для кожного випадкового запису реальних даних вибирається близький запис у

синтетичних даних, а потім обидва об’єднуються для формування гібридних даних. Це забезпечує переваги як повністю, так і частково синтетичних даних.

В машинному навчання існує два способи створення аугментованих даних:

1) генерація даних по функції щільності розподілу. Ідея цього методу полягає в тому, щоб відтворити приклади даних, які будуть мати схожий розподіл з розподілом реальних даних. Такі дані можна створювати на основі генеративних моделей;

2) моделювання на основі агентів. Ключова ідея полягає в тому, щоб створити фізичну модель яка буде повторювати поведінку реального явища або об’єкта. Вона зосереджена на розумінні впливу взаємодії між агентами, що безпосередньо впливає на систему в цілому.

3.3 Створення аугментованих даних за допомогою бібліотеки scikit- learn

Scikit-learn – одна з найбільш широко використовуваних бібліотек Python для класичних завдань машинного навчання. Ця бібліотека дозволяє виконувати безліч операцій та надає безліч алгоритмів. Scikit-learn пропонує чудову документацію про свої класи, методи та функції, а також опис алгоритмів, що використовуються. Scikit-learn пропонує можливість підготовки даних до моделей машинного навчання: стандартизації або нормалізації даних, кодування категоріальних змінних та багато іншого. Бібліотека має дві базові функції для генерування випадкової регресії та задачі класифікації. Вони інтегровані в API sklearn.dataset.

Dataset.make_regression: генерувати задачу випадкової регресії. Виходові дані генеруються шляхом використання (потенційно зміщеної) моделі випадкової лінійної регресії з певною кількістю не нульових

регресорів до попередньо згенерованого входових даних і деяким гаусівським нормалізованим випадковим шумом.

Dataset.make_classification: генерувати випадкову модель класифікації n-кластерів. Для початку створюються кластери точок, нормально розподілених в середині n-вимірного гіперкуба, далі необхідно визначитись з кількістю кластерів.

Лінійна регресія – це метод пошуку кореляції між ознаками (фічами) та таргет змінною. Метод dataset.make_regression може створити модель випадкової регресії з довільною кількістю входових функцій, виходових цілей і контрольованим коефіцієнтом інформаційного зв'язку між ними. Він також може змішувати гауссовий шум. На рисунку 3.3 зображено модель, яка генерує дані для задачі регресії.

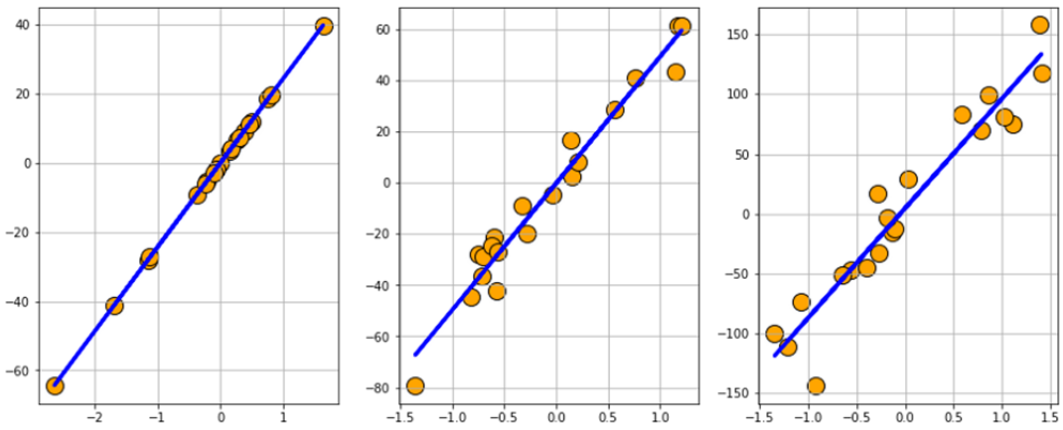


Рисунок 3.3 – Генерація даних за допомогою scikit-learn із різним ступенем шуму для задачі регресії

Генерація даних для задачі класифікації: подібно до моделі регресії, dataset.make_classification генерує випадкову множину точок даних для класифікації кількох класів (набір даних) з регульованим поділом класів і коефіцієнтом шуму. Можна випадковим чином розмножити будь-який відсоток входових даних, щоб створити складніший набір даних для задачі

класифікації. На рисунку 3.4 наведено генерацію даних для задачі класифікації.

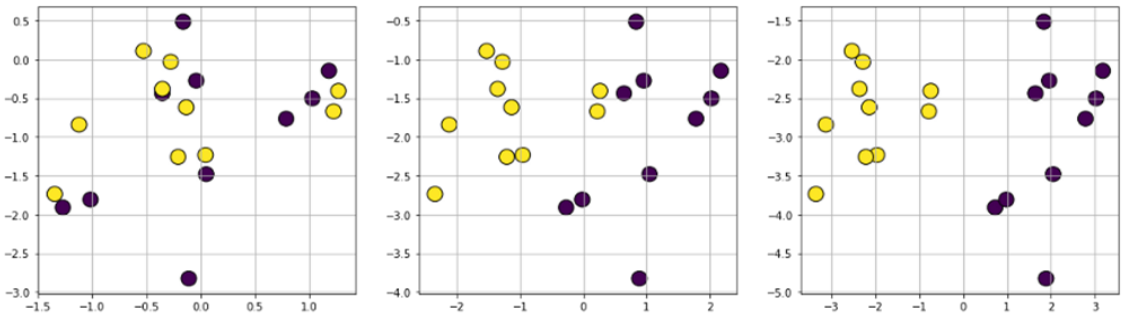


Рисунок 3.4 – Генерація додаткових даних для задачі класифікації за допомогою scikit-learn з різною кількістю класів

Генерація додаткових даних для задачі кластерування: існує досить багато функцій для створення кластерів даних довільної форми. Найпростішим є datasets.make_blobs, який генерує будь-яку кількість кластерів з регуляризацією параметру відстані між кластерами. На рисунку 3.5 наведено генерацію даних довільної форми для задачі кластерування.

Генерація анізотропних кластерів: за допомогою простого перетворення з використанням множення матриці можна створити кластери, які вирівняні вздовж певної осі або анізотропно розподілені. На рисунку 3.6 зображено генерацію анізотропних кластерів.

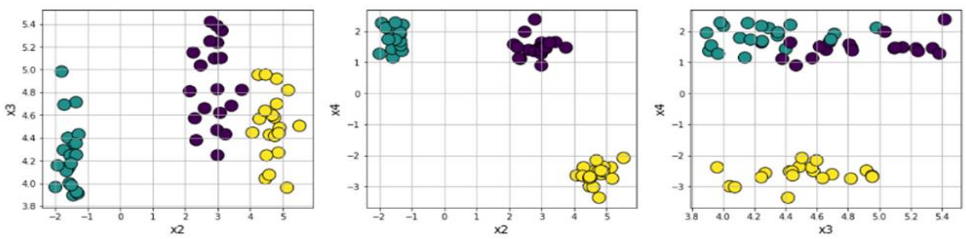


Рисунок 3.5 – Генерування даних для вирішення задачі кластерування за допомогою scikit-learn

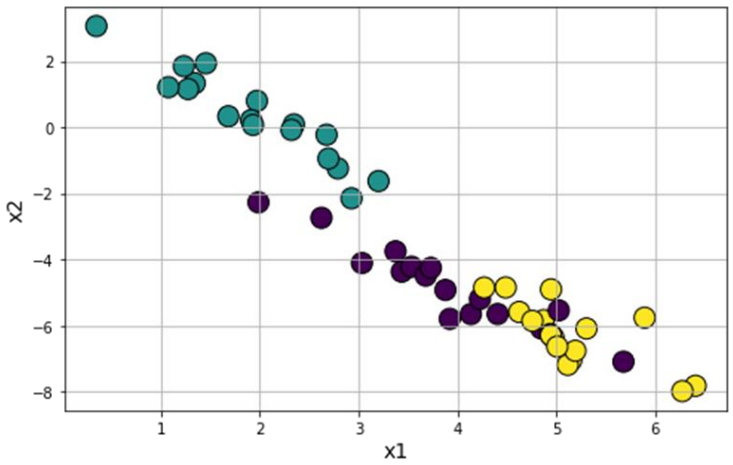


Рисунок 3.6 – Генерація вирівняних анізотропних даних, за допомогою scikit- learn

Генерація даних еліпоподібної (кільцевої) форми: для тестування моделей для кластерування на основі спорідненості або моделей гавсівського розподілу корисно створювати кластери еліпсоїдної форми. Для цього використовується функцію datasets.make_circles. На рисунку 3.7 зображено генерацію даних еліпсоїдної форми, що є лінійно не роздільними.

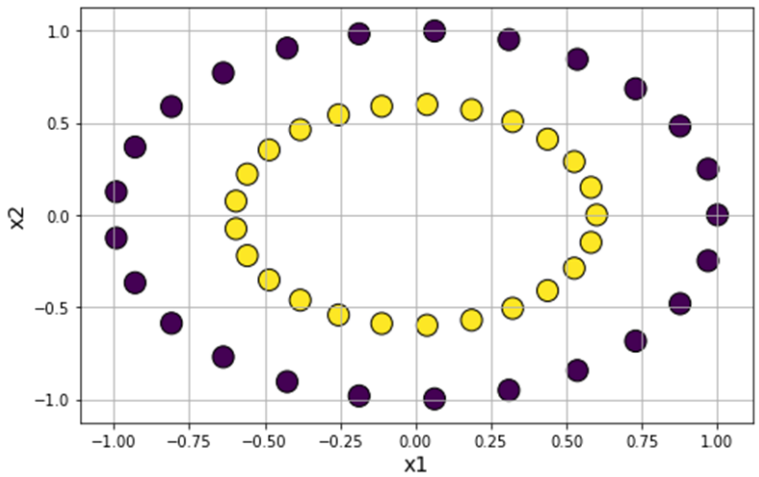


Рисунок 3.7 – Генерацію лінійно нероздільних даних еліпсоїдної форми, за допомогою scikit-learn

Також можна додати невеликий відсоток шуму до даних, щоб зробити модель більш інваріантної до різких змін в природі даних, що надходять на опрацювання. На рисунку 3.8 наведено дані еліпсоїдної форми з невеликим відсотком гавсівського шуму.

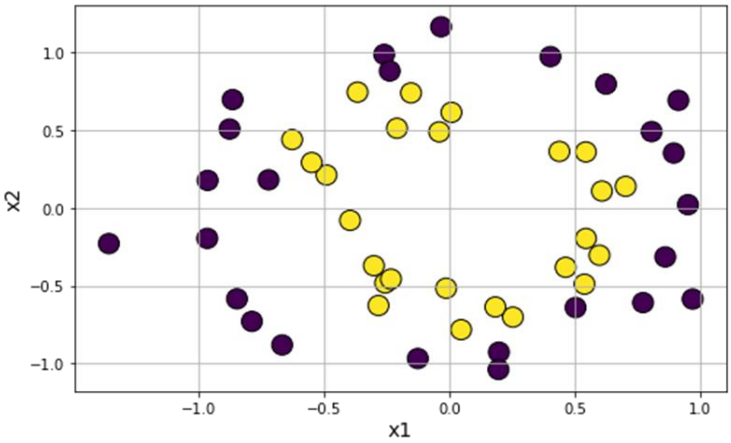


Рисунок 3.8 – Візуалізація двох кластерів еліпсоїдної форми з невеликим відсотком шуму

Генерація кластерів у формі місяця: є випадки коли треба генерувати дані у формі місяця для тестування алгоритмів із регульованим шумом за допомогою функції datasets.make_moons. На рисунку 3.9 зображено генерацію кластерів у формі місяця.

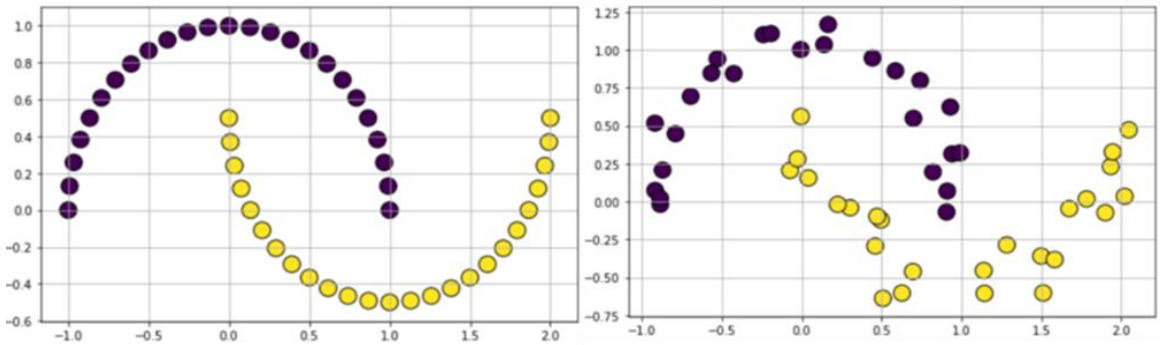


Рисунок 3.9 – Генерація кластерів місячної форми, за допомогою scikit- learn

3.4 Генерація даних довільної випадкової форми

У випадку, коли необхідно оцінити ефективність декількох класифікаторів SVM з різним відсотком роздільності класів (від лінійних до нелінійних) або продемонструвати обмеження лінійних моделей стандартними функціями scikit-learn це зробити дуже важко. Вихід моделі регресії не є остаточною функцією входових даних, тому вони вважаються частково випадковими.

Хоча стандартних методів генерації та аугментації даних може бути достатньо для вирішення багатьох задач машинного навчання, виникають ситуації, коли може знадобитися контрольований спосіб створення даних на основі наперед визначеної функції. На рисунку 3.10 зображено набір згенерованих даних випадкового розподілу.

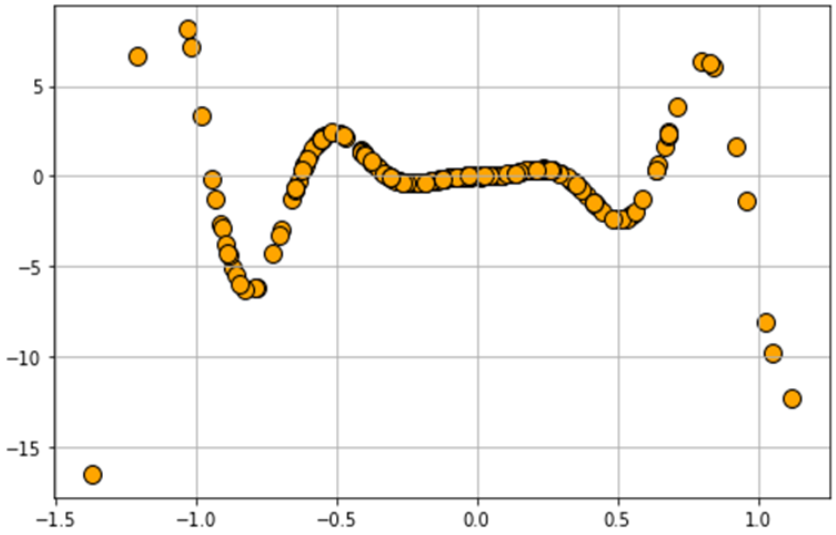


Рисунок 3.10 – Набір випадкових даних для задачі регресії.

Також можна створити набір даних на основі границь нелінійної есіпсоїдної форми для задачі класифікації для тестування моделей нейронних мереж рисунки 3.11, 3.12, 3.13.

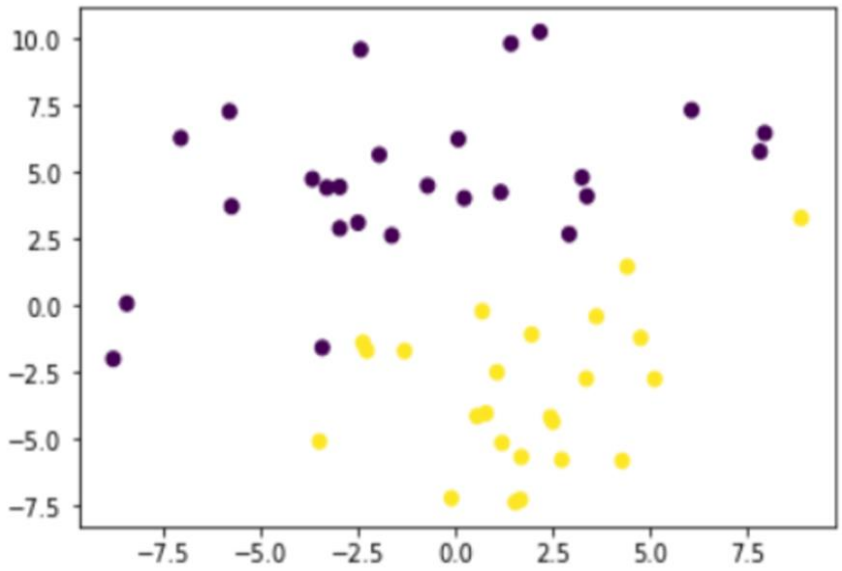


Рисунок 3.11 – Набір лінійно роздільних даних без шуму

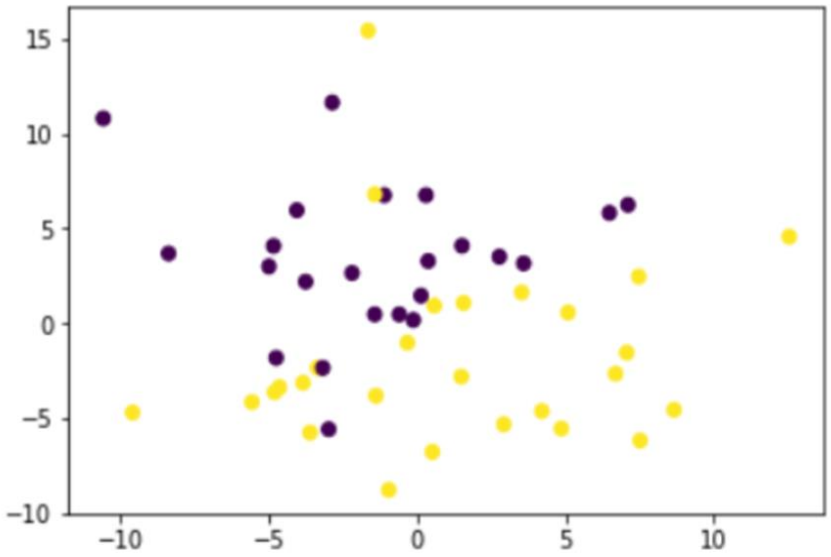


Рисунок 3.12 – Набір лінійно роздільних даних з невеликим коефіцієнтом шуму

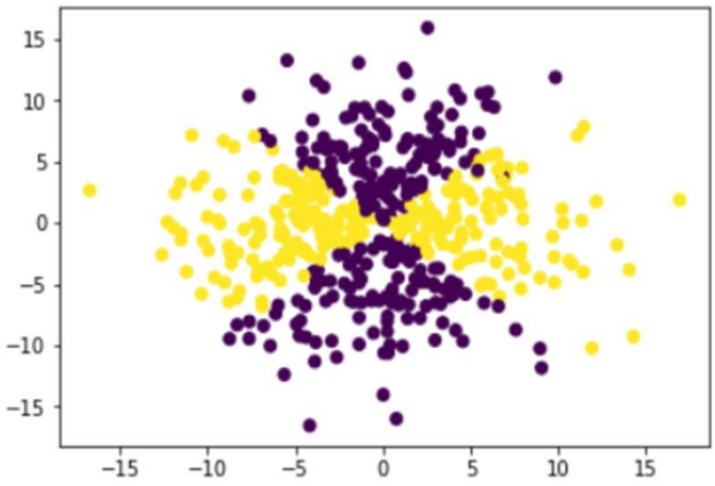


Рисунок 3.13 – Набір випадкових лінійно нероздільних даних

3.5 Створення додаткових даних за допомогою бібліотеки pydbgen

Pydbgen – це Python бібліотека розроблена для створення випадкових корисних записів (наприклад, ім’я, адреса, номер кредитної картки, дата, час, назва компанії, посада, номерний знак тощо) та збереження їх у будь-якому об’єкті Pandas dataframe, або у вигляді таблиці SQLite у файлі бази даних, або у файлі MS Excel. На рисунках 3.14, 3.15, 3.16 зображено приклади роботи функцій бібліотеки pydbgen.

```
generator.gen_data_series(num=10,data_type='name')
```

```
0         Pamela Cook
1      Lindsey Wright
2       Rachel Steele
3     Latasha Jenkins
4     Reginald Harris
5      Timothy Conner
6        Tina Johnson
7       Michael Smith
8        Lucas Hansen
9      Elizabeth Cruz
dtype: object
```

Рисунок 3.14 – Генерація випадкових імен бібліотекою pydbgen

```
generator.gen_data_series(num=10,data_type='phone_number_full')
```

```
0              754-279-1802
1            1-761-599-9477
2          460-649-0208x548
3      1-522-294-6949x75198
4          +18(8)9084634782
5              (278)795-4557
6          +32(7)5595839493
7        645-121-0018x53889
8              230-079-7069
9         (753)057-2266x606
dtype: object
```

Рисунок 3.15 – Генерація телефонних номерів

```
df10 = generator.gen_dataframe(fields=['name','street_address','ssn','email','date'])
```

```
df10
```

	name	street_address	ssn	email	date
0	Tiffany Brown	88274 Susan Gateway Suite 656	342-54-8064	TiffanyBrown43@yandex.com	1972-12-25
1	Troy Fowler	0185 Marcus Rapids	257-09-0833	Troy_Fowler12@hotmail.com	1993-11-22
2	Linda Smith	7566 Roberts Road Suite 027	389-50-9696	LindaSmith@aol.com	2012-04-18
3	Brian Fisher MD	87648 Jose Parkway	399-60-2669	BMD@yandex.com	2003-07-17
4	Erica Harris	42393 Mills Branch Apt. 716	561-05-8951	EHarris88@outlook.com	1983-11-20
5	John Gonzalez	939 Ferguson Heights	386-99-3545	John_G@att.com	1990-05-15
6	Christopher Levy	5955 Graves Camp	363-53-9313	Christopher.Levy66@hotmail.com	1979-05-30
7	Emma Morton	328 Zimmerman Harbor Apt. 885	458-07-2763	Emma_M@yandex.com	1990-03-02
8	Donna Cameron	766 Ingram Throughway Suite 540	238-99-3537	DonnaCameron@verizon.com	1976-02-21
9	Diane Taylor MD	90039 Donald Road Apt. 532	290-73-1196	DianeMD@outlook.com	1985-04-16

Рисунок 3.16 – Створення випадкового набору даних з заданими критеріями даних, що мають бути згенеровані

Використання аугментованих або частково аугментованих даних дозволяє покращити результати моделей і не дозволити моделям машинного навчання перенавчитись на тренувальному наборі.

4 ІМІТАЦІЙНЕ МОДЕЛЮВАННЯ

4.1 Середовище та бібліотеки

При розробці систем в галузі машинного навчання, використання мови програмування Python може надати розробнику великий вибір готових бібліотек та фреймворків для їх застосування на усіх етапах створення проекту. Для аналізу даних та попередньої обробки використовуються такі бібліотеки, як pandas та sklearn. В застосунках, що пов’язані з опрацюванням природної мови для навчання моделей та їх оцінюванням найпопулярнішим рішенням є використання фреймворку Hugging Face.

Бібліотека transformers, цього фрайворку, надає доступ до великої кількості попередньо навчених моделей, до яких в процесі дотренування можливо застосувати інтеграцію з бібліотекою Tune для оптимізації гіперпараметрів. Для аналізу та моніторингу навчання моделей використовуються сервіс Weights & Biases.

Через те, що за основу архітектури системи виправлення граматичних та синтаксичних помилок буде використано попередньо навчену модель з великою кількістю параметрів, для її дотренування на текстових даних української мови та подальшого тонкого налаштування гіперпараметрів необхідно враховувати кількість комп’ютерних ресурсів, здебільшого для таких задач моделі тренуються на відеокартах. Через це, доцільним є використання хмарних ресурсів компанії Google з можливістю створення віртуальної машини із необхідними характеристиками.

Не менш важливим параметром є задання середовища розробки, а саме Python версії 3 та попередньо встановлених драйверів CUDA 11.0, які дозволять використовувати потужності відеокарти в процесі навчання системи виправлення граматичних та синтаксичних помилок.

4.2 Застосування анотованих корпусів для навчання моделей

UA-GEC перший анотований GEC-корпус української мови. Корпус поділяється на 2 частини: навчальна та тестова. В загальному підрахунку, перша версія корпус, що була опублікована в 2020 році містила більше 1000 статей, понад 20000 речень, які можна використовувати задля генерування синтетичних даних. В листопаді 2022 року було опубліковано в загальний доступ розширену та вдосконалену версію корпусу 2.0. Набір даних UA-GEC був збільшений до майже 34 000 речень і відтепер містить два варіанти анотації. Це дає змогу використовувати його у двох різних завданнях: виправленні тільки граматики та виправленні граматики й стилю. Також було розроблено більш деталізовану класифікацію помилок: зокрема категорії «граматика» і «стиль» розділили ще на 13 і 5 підкатегорій відповідно (і додали підкатегорію для помилок, які не потрапили в жодну з цих підкатегорій). В таблиці 4.1 наведено базові характеристики корпусу UA-GEC.

Таблиця 4.1 – Характеристик корпусу UA-GEC

Розділи	Документи	Речення	Токени	Автори
Навчальна	851	18,255	285,247	416
Розробницька	160	2,490	43,432	76
Загалом	1,011	20,715	328,779	492

Після того, як було проведено аналіз наборів даних, було прийнято рішення використовувати анотовані дані з другої версії UA-GEC. Все навчання буде проходити на розмічених даних, також для покрашення результатів було зроблено агументацію існуючих навчальних даних, для того щоб надати моделі властивості інваріантності.

Для аугментації даних було використано скрип, який розуміє скільки файлів знаходиться у директорії, виймає дані, які записані у кожному файлі,

та зберігає в один основний файл. У фінальному результаті, був готовий один робочий файл, який містив дані для, які можна було використовувати для генерації синтетичних даних.

Крім створеного компанією Grammarly корпусу UA-GEC, також існує ще один корпус української мови. Браунський корпус української мови – відкритий, збалансований за жанрами українських текстів набір обсягом 1 млн слововживань. Цей набір текстів був побудований на основі відомого корпусу англійської мови Brown. Усі дані в цьому корпусі поділені по категоріям: good – перевірені фрагменти, написані літературною українською мовою, so-so – перевірені фрагменти, що містять помилки, bad – перевірені фрагменти, що зовсім не відповідають вимогам та unprocessed – фрагменти, що чекають на перевірку. З цього корпусу для імітаційного моделювання було віддібрано категорію good, оскільки модель буде донавчатись також на аугментованих даних, а для реалізації цієї функції треба щоб набір, на основі якого буде проводитись аугментація був високої якості і майже не містив помилок.

Ще одним із етапів попередньої обробки даних було створення анотованого набору даних з Браунівського корпусу, це було зроблено через те, що перший корпус, що було обрано (UA-GEC) містить анотації, тому для навчання моделі треба було перевести і другу частину в анотований вигляд.

Для автоматизації цього процесу було обрано розробку компанії Grammarly, що є у відкритому доступі. Їх кастомно створена функція для автоматичної анотації побудована на основі бібліотеки stanza. Stanza – це бібліотека аналізу природної мови, створена Стенфордською групою NLP. Це набір інструментів NLP, які можна використовувати для створення нейромережевих конвеєрів для аналізу тексту. Ця кастомна бібліотека підтримує такі функціональні можливості, як токенізація, розширення лексем з кількох слів, лемматизація, частина мови (POS), тегування морфологічних ознак, розбір залежностей, розпізнавання іменованих об’єктів (NER) та аналіз настроїв.

4.3 Створення аугментованих даних

Для того щоб розширити набори тренувальних даних було створено аугментовані дані на основі реальних. Ідея полягає в тому, щоб на основі реальних даних, які було підготовлено на етапі збору датасетів, згенерувати та розширити обсяг даних, задля покращення якості моделі та надання інваріантних властивостей. Набір текстових даних для імітаційного моделювання, на якому буде тренуватись модель, будуть складатись з наступних частин: 30 тис аугментованих даних з UA-GEC корпусі на основі пермутації та комбінації, 30 тис аугментованих даних з Браунівського корпусу на основі пермутації та оригінальні дані з UA-GEC для фінальної перевірки моделей.

Перед аугментацією даних, було проведено попередній аналіз, який складається з наступних етапів:

– видалення усіх розділових знаків, та форматування текст таким чином, щоб не було зайвих відступів;

– переведення всіх текстів у нижній регістр, щоб не відбувалась перестановка слів;

– видалення усіх речень, які містять інші мови (в наборах даних, що було використано за замовчування були присутні слова англійською мовою).

При генеруванні першої партії аугментованих даних, на основі UA-GEC було прийнято рішення будувати нові граматичні речення. Для цього використовувався метод пермутації (метод перестановки). Цей метод полягає у зміні розташування слів у реченні. На рисунку 4.1 зображено процес пермутації речення. У вигляді входу в систему передається правильне речення, на виході отримується набор речень з всією можливою перестановкою слів.

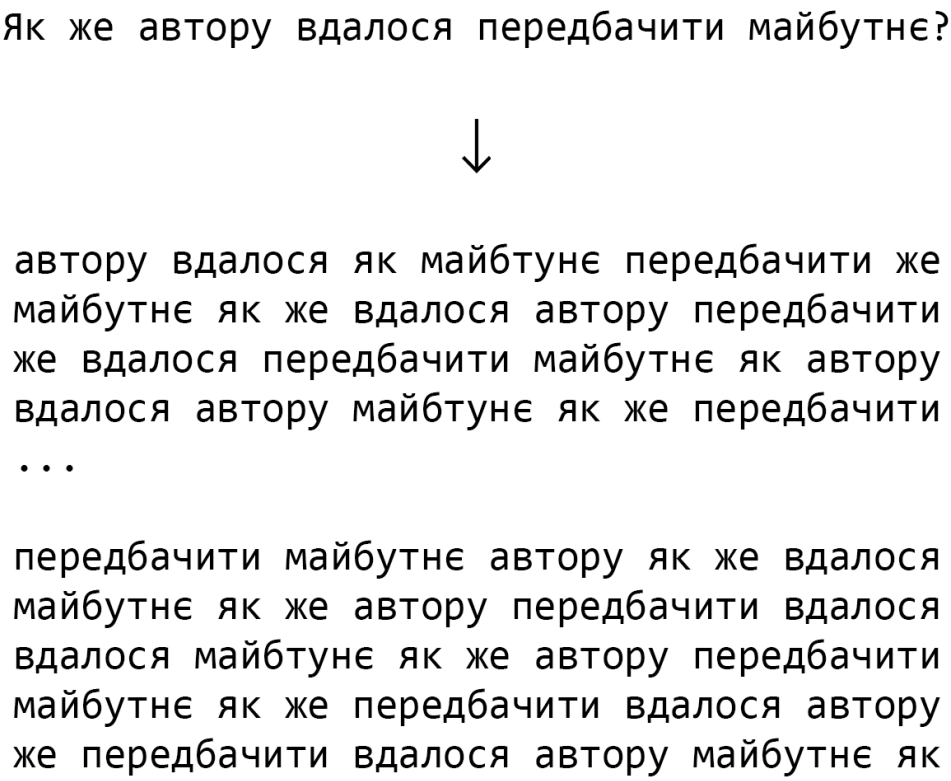


Рисунок 4.1 – Пермутація речень для створення нових прикладів

Задля створення аугментованих додаткових множин даних, було вирішено використовувати вбудований метод permutation з пакету itertools який знаходиться у Python, та переописати його для можливості зберігання даних одразу на етапі аугментації даних. Нова кастомна функція, яка працює на основі методу пермутації використовує принцип так званого барабану. На основі речення, яке надходить як параметр, посуваючи кожне слово на 1 крок вправо – створюються нові штучно синтезовані речення. На рисунку 4.2 схематично представлено принцип барабану, який використовується для створення аугментованих речень.

```
Як же автору вдалося передбачити майбутнє?

                        ↓

майбутнє як же автору вдалося передбачити
передбачити майбутнє як же автору вдалося
вдалося передбачити майбутнє як же автору
же автору вдалося передбачити майбутнє як
```

Рисунок 4.2 – Представлення принципу барабану, що застосовано для створення нових прикладів речень

Процес творення нових речень йде по колу, за рахунок використання принципу барабану. Цикл буде тривати допоки кожне слово не буде розташоване у всіх можливих позиціях. На рисунку 4.3 зображено зміну позицій слів, та повторний процес барабану.

```
же як автору вдалося передбачити майбутнє?

                        ↓

майбутнє же як автору вдалося передбачити
передбачити майбутнє же як автору вдалося
вдалося передбачити майбутнє же як автору
автору вдалося передбачити майбутнє же як
як автору вдалося передбачити майбутнє же
```

Рисунок 4.3 – Принцип барабану, який використовується для створення нових речень, із зміненою позицією слів

В результаті було отримано новий набір даних, в якому містяться пари речень, правильне речення та відповідна йому аугментація. На рисунку 4.4 наведено набір пар речень.

```
1,як же автору вдалося передбачити майбутнє,же автору передбачити вдалося майбутнє як
2,як же автору вдалося передбачити майбутнє,майбутнє же автору вдалося передбачити як
3,як же автору вдалося передбачити майбутнє,як передбачити автору вдалося же майбутнє
4,як же автору вдалося передбачити майбутнє,же автору вдалося передбачити майбутнє як
...
...
93,як же автору вдалося передбачити майбутнє,вдалося передбачити як майбутнє же автору
94,як же автору вдалося передбачити майбутнє,автору перебачити вдалося майбутнє як же
95,як же автору вдалося передбачити майбутнє,же як майбутнє автору вдалося передбачити
96,як же автору вдалося передбачити майбутнє,автору вдалося як майбутнє передбачити же
```

Рисунок 4.4 – Результату генерації нових речень у форматі: 1) індекс, 2) речення, на базі якого генерується речення, 3) нове згенероване речення

В кінцевому результаті, було згенеровано 3 файли із 100 тис, 500 тис та 1 млн згенерованими реченнями задля покращення тренування моделі. Також був підготовлений файл із 30 тис реченнями, які будуть використовуватись під час фінального навчання відповідно на основі UA-GEC та Браунівського корпусу.

4.4 Аугментація даних на основі комбінації

Перед процесом аугментації даних, було проведено попередній аналіз наборів даних. Для тренування на основі речень з пунктуаційними знаками, для створення другого набору аугментованих даних знаки пунктуації не видалялись з початкових речень.

Для аугментації речень зі знаками пунктуації, які створювались на основі корпусу UA-GEC, було використано метод комбінації. Цей метод полягає у розміщення знаків пунктуації відповідно від кількості слів у реченні. Робота методу починається з того, що одна кома ставиться в усіх можливих місцях. Після цього, кількість знаків (ком) збільшується на одну, і процес повторюється, допоки не залишиться нових варіанті. На рисунку 4.5 зображено метод комбінації, який було використано до речень. Входове речення передається в метод, як параметр, а в результаті отримуються всі можливі варіанти розташування ком у входовому реченні.

В наведеному на рисунку 4.5 прикладі показано цей процес для речень де переставлялась тільки одна та дві коми.

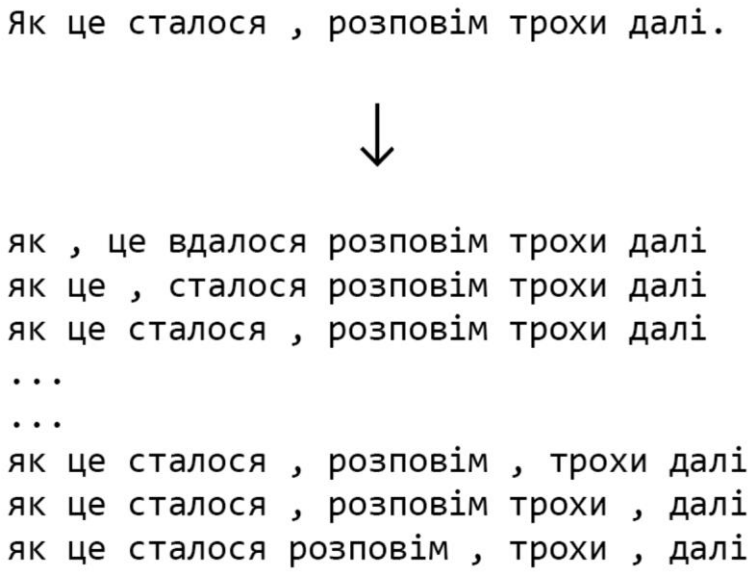


Рисунок 4.5 – Методу комбінації, що використовується для створення нових речень зі знаками пунктуації

По ітогу формується csv файл, який містить в собі коректне речення, та відповідне нове аугментоване. На рисунку 4.6 наведено результат, який записується csv файл.

```
1,"Як це сталося , розповім трохи далі","Як , це сталося розповім трохи далі"
2,"Як це сталося , розповім трохи далі","Як це , сталося розповім трохи далі"
3,"Як це сталося , розповім трохи далі","Як це сталося , розповім трохи далі"
4,"Як це сталося , розповім трохи далі","Як це сталося розповім , трохи далі"
...
...
27,"Як це сталося , розповім трохи далі","Як , це , сталося , розповім трохи , далі"
28,"Як це сталося , розповім трохи далі","Як , це , сталося розповім , трохи , далі"
29,"Як це сталося , розповім трохи далі","Як , це сталося , розповім , трохи , далі"
30,"Як це сталося , розповім трохи далі","Як це , сталося , розповім , трохи , далі"
31,"Як це сталося , розповім трохи далі","Як , це , сталося , розповім , трохи , далі"
```

Рисунок 4.6 – Приклад нових речень зі знаками пунктуації, що записуються у форматі: 1) індекс, 2) речення, на базі якого генерується речення, 3) нове згенероване речення

Було згенеровано 3 файли із 100 тис, 500 тис та 1 млн згенерованими реченнями задля розширення даних для тренування моделі. Також був підготовлений файл із 30 тис реченнями, які будуть використовуватись під час фінального навчання відповідно на основі UA-GEC та Браунівського корпусу.

В результаті попередньої обробки даних, було підготовлено аугментовані дані, та частково аугментовані дані, які складаються із згенерованих синтетичних даних з UA-GEC та Бранівського корпусу, та сам корпус UA-GEC. В загальному, частково аугментовані дані містять орієнтовно 150 тис. анотованих речень.

4.5 Векторизація текстових даних

Для вирішення завдання GEC на основі моделей нейромережевого машинного перекладу зібрані набори даних перед тим, як вони потраплять на входи нейромережевої моделі необхідно підготупати специфічним чином, так, щоб модель глибинної нейромережі змогла їх опрацювати. Для цієї підготовки недостатньо виконати послідовність етапів перетворення текстових даних.

По-перше кожне речення з базового текстового формату за допомогою токенізатора багатомовної моделі mT5 (яка буде використовуватись як базова модель для вирішення завдання виправлення граматичних та синтаксичних помилок) було розділено на окремі токени. Процес створення токенів зображено на рисунку 4.7.

```
Я все йду, самотній на землі, як сонце на небі.
```

↓

```
['_Я', '_все', '_', 'йду', ',', '_само', 'тн', 'і',
 'й', '_на', '_земл','і', ',', '_як', '_со', 'нце',
 '_на', '_не', 'б', 'і', '.', '</s>']
```

Рисунок 4.7 – Токенізація речення за допомогою токенізатора моделі mT5

Важливо зазначити, що при токенізації речення додатково генеруються спеціальні токени, які використовують моделі в процесі навчання (вони усуваються в процесі декодування). Токен </s> позначає кінець речення. Інший токен, <pad>, призначений для генерування міні-пакетів даних однакової довжини для більш ефективного використання машиною відео пам’яті. При неможливості токенізувати слово чи окремий символ використовується токен <unk>.

По-друге необхідно замінити кожен токен його ідентифікатором з таблиці вбудувань, яка є компонентною, отриманої з попередньо навченої моделі mT5. На рисунку 4.8 зображено схематично цей процес.

```
['_Я', '_все', '_', 'йду', ',', '_само', 'тн', 'і',
 'й', '_на', '_земл','і', ',', '_як', '_со', 'нце',
 '_на', '_не', 'б', 'і', '.', '</s>']
```

↓

```
[1673, 1132, 259, 17280, 261, 1660, 18905, 266, 543,
310, 10643, 266, 261, 1086, 917, 11946, 310, 401,
1067, 266, 260, 1]
```

Рисунок 4.8 – Заміна кожного токену його відповідним ідентифікатором для подальшої векторизації слів

За допомогою id-номеру можливо отримати з мовного словника моделі mT5 відповідну векторизацію слів, яка в подальшому буде використана для комп’ютерних обчислень та генерації виходових векторних представлень слів.

4.6 Опис базової моделі для виправлення граматичних та синтаксичних помилок

В якості базового підходу для вирішення задачі виправлення граматичних та синтаксичних помилок для української мови було використано підхід та модель, яку було описано в [41]. В цьому підході задачу GEC описано як завдання нейромережевого машинного перекладу з використанням аугментованих даних. За основу було взято мовну модель mT5 [64], багатомовну версію T5 [42], яка навчалась на 101 мові. Ця модель має архітектуру енкодер-декодер та побудовано на основі моделі трансформер. Модель, що використовується для цієї задачі передбачає та вставляє слова у попередньо зазначені проміжки тексту використовуючи спеціальні маркери.

За допомогою оптимізації гіперпараметрів моделі mT5 реалізується її адаптація до вирішення конкретнішого завдання, а саме завдання GEC для української мови. З точки зори програмної реалізації було використано технологію трансферного навчання нейромережевих моделей, цю технологію використовують для того щоб не навчати модель з нуля.

Для реалізації трансферного навчання було взято попередньо навчену модель трансформеру mT5 без будь-якого конкретного шару голови (на вихід дає необроблені приховані стани), на місце якого було додано власний, виходовий, шар мовного моделювання, який буде навчатись на даних української мови.

4.7 Гіперпараметри для тонкого налаштування моделі

При навчанні глибинних нейронних мереж необхідно визначити набір гіперпараметрів для отримання оптимальних та не зміщених оцінок результатів. Під час тренування (основної фази) та тонкого налаштування будуть задіяні такі гіперпараметри:

– крок навчання (learning rate), який задає розмір кроку на кожній ітерації пошуку мінімум функції втрат;

– зменшення ваги навчання (weight decay) працює шляхом додавання штрафного значення до функції втрат, що призводить до зменшення ваги під час зворотного поширення;

– розмір міні-пакетів даних (batch size), які використовуються за одну ітерацію навчання;

– оптимізатори – це алгоритми або методи, які використовуються для зміни атрибутів нейронної мережі, таких як ваги та параметр кроку навчання, з метою зменшення загальних втрат та підвищенню точності передбачення.

В попередніх розділах магістерської кваліфікаційної роботи було проведено аналіз існуючих систем для вирішення задачі виправлення граматичних помилок та систем нейромережевих машинних перекладачів. Протягом цього аналізу було виділено основні гіперпараметри, що будуть налаштовуватись при навчанні системи GEC, також було визначено набір допустимих початкових значень для гіперпараметрів, щоб завдяки цим значення ініціалізувати систему. Таких наборів початкових параметрів було створено 9 комбінацій. За базу було обрано метод оптимізації на основі мутації параметрів.

На рисунках 4.9 та 4.10 зображені результати навчань з мутуючими гіперпараметрами для створених комбінацій.

PopulationBasedTraining: 22 checkpoints, 10 perturbs
Resources requested: 4.0/4 CPUs, 1.0/1 GPUs, 0.0/6.68 GiB heap, 0.0/3.34 GiB objects (0.0/1.0 accelerator_type:T4)
Result logdir: /home/jupyter/ray_results/tune_transformer
Number of trials: 9/9 (8 PAUSED, 1 RUNNING)

Trial name	status	loc	w_decay	lr	train_bs/gpu	num_epochs	optimizer	eval_loss	epoch	training_iteration
_objective_61e93_00001	RUNNING	10.164.0.5:15983	0.0001	0.0008	64	8	adafactor	0.396403	4	4
_objective_61e93_00000	PAUSED	10.164.0.5:10987	0.0001	0.001	64	8	adafactor	0.412576	5	5
_objective_61e93_00002	PAUSED	10.164.0.5:17811	0.0001	0.0012	64	8	adafactor	1.11383	4	4
_objective_61e93_00003	PAUSED	10.164.0.5:20913	0.001	0.0008	64	8	adamw_hf	0.385278	4	4
_objective_61e93_00004	PAUSED	10.164.0.5:24119	0.001	0.001	64	8	adamw_hf	0.384401	4	4
_objective_61e93_00005	PAUSED	10.164.0.5:31008	0.5	0.0008	16	8	adamw_hf	0.37985	4	4
_objective_61e93_00006	PAUSED	10.164.0.5:1710	0.5	0.00096	16	8	adamw_hf	0.534785	4	4
_objective_61e93_00007	PAUSED	10.164.0.5:4828	0.001	0.0012	64	8	adamw_hf	0.50416	4	4
_objective_61e93_00008	PAUSED	10.164.0.5:7960	0.5	0.001	16	8	adamw_hf	0.379871	4	4

Рисунок 4.9 – Результати навчання системи GEC з мутуючими гіперпараметрами

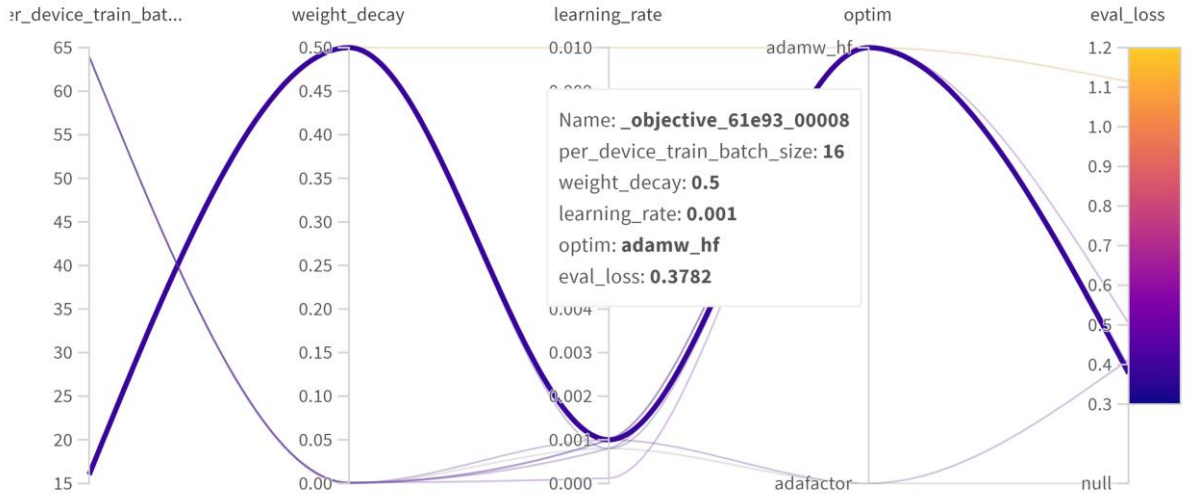


Рисунок 4.10 – Результати навчання моделей з мутуючими гіперпараметрами

Завдяки поставленому експерименту було виділено оптимальну комбінацію гіперпараметрів та їх найкращі значення для отримання найкращих з точки зору метрик оцінки якості результатів. Основні параметри, які впливають на метрики якості це значення параметру кроку

навчання та значення ваг. Об’єм даних по міні-пакетам не впливає на метрики якості, але пришвидшує навчання моделі та дозволяє ефективніше використовувати можливості відео карти.

Порівнюючи працездатність методів оптимізації – Adam і Adafactor дає можливість зробити висновок про те, що ці два отимізатори в задачі GEC дають майже однакові результати, порівнюючи їх по ефективності.

4.8 Результати навчання системи виправлення граматичних та синтаксичних помилок на корпусах української мови

Для підтвердження теоретичних досліджень існуючих систем для вирішення задачі виправлення граматичних та синтаксичних помилок, було проведено імітаційне моделювання моделі на основі Уваги на анотованих реченнях корпусу UA-GEC. На початку роботи дані було поділено на три частини: тренувальну множину (на цьому наборі моделі навчались), валіданціну множину (частина даних, на яких розраховувались метрики якості і значення функції втрат після кожної епохи навчання) та тренувальну множину (цей набір даних використовувався для отримання незміщених оцінок метрик якості та функції втрат).

Навчання проводились на групі моделей uk-mT5, основні статистичні характеристики всіх використаних моделей наведено у таблицях 4.2 та 4.3. Для фінального оцінювання було використано тестові дані. На рисунку 4.11 наведена необхідний час для тренування моделей виправлення помилок української мови.

Обрані моделі uk-gec-mT5$_{SMALL}$, uk-gec-mT5$_{BASE}$, uk-gec-mT5$_{BASE}$ навчались 1 годину 43 хвилин, 2 години 40 хвилин, 7 годин 17 хвилин відповідно.

Таблиця 4.2 – Характеристики моделей mT5 та їх українських аналогів

Модель	Кількість параметрів	Кількість токенів у словнику	Розмір моделі
$mT5_{SMALL}$	300 мільйонів	250 тисяч	1.1 Гбайт
uk-$mT5_{SMALL}$	75 мільйонів	8900	0.3 Гбайт
$mT5_{BASE}$	582 мільйонів	250 тисяч	2.2 Гбайт
uk-$mT5_{BASE}$	244 мільйонів	8900	0.95 Гбайт
$mT5_{LARGE}$	1.2 мільярди	250 тисяч	4.6 Гбайт
uk-$mT5_{LARGE}$	779 мільйонів	8900	2.9 Гбайт

Таблиця 4.3 – Порівняння результатів пробного навчання між групами моделей mT5 та uk-mT5

Модель	Тривалість навчання (секунди)	Екземплярів в секунду	Зайнято VRAM (Мбайт)
$mT5_{SMALL}$	52.40	19.09	11494
uk-$mT5_{SMALL}$	26.43	37.84	4264
$mT5_{BASE}$	119.71	8.35	14410
uk-$mT5_{BASE}$	61.76	16.19	7180
$mT5_{LARGE}$	451.87	2.21	14704
uk-$mT5_{LARGE}$	313.27	3.19	9770

Для оцінки якості навчання моделей було проведено серію експериментів. Протягом цих експериментальних досліджень для кожної моделі було розраховано значення функції втрат як на тренувальній множині даних, так і валідаційній, після кожної епохи навчання. Криві функції втрат наведено на рисунку 4.11 для кожної GEC моделі.

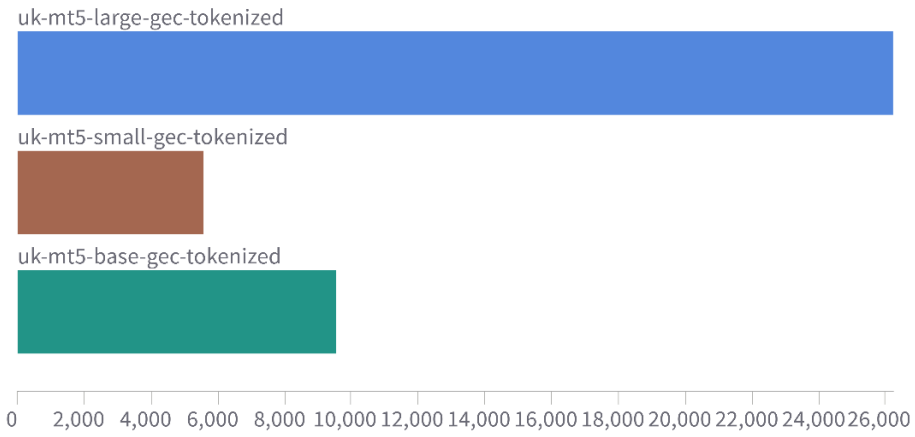


Рисунок 4.11 – Загальний час навчання трьох використаних моделей uk- mT5

По наведеним висче графікам кривих навчання та валідації можна зробити висновок, що під час робити всіх протестованих систем не виникла проблем з недонавчання (underfitting) або перенавчання (overfitting). На рисунках 4.12, 4.13 наведено порівняння значень функцій втрат на валідаційних даних при навчанні GEC моделей.

З наведених вище графіків змінення значень функції втрат можна зробити висновок, що зі збільшення кількості параметрів моделі, зменшується значення функції втрат. З усіх протестованих моделей GEC найкращі результати продемонструвала uk-gec-mT5$_{\text{LARGE}}$ її мінімальне значення по функції втрат – 0.27.

Частіше в якості метрики оцінки якості моделей машинного навчання прийнято використовувати F-міру, яка розраховується через розрахунок метрик влучності та повноти. Де влучність (precision) рахується як різниця між загальною кількістю правильно позитивних результатів та число всіх позитивних результатів, включаючи ті, що було визначено неправильно. Повнота (recall) це різниця між правильно визначеними дійсно позитивними результатами та сумарною кількістю всіх значень, що мають бути визначені як позитивні. Для розрахунку F-міри була застосована метрика, яка частіше

за все використовується для оцінки систем виправлення граматичних та синтаксичних помилок – MaxMatch, або m2 [45].

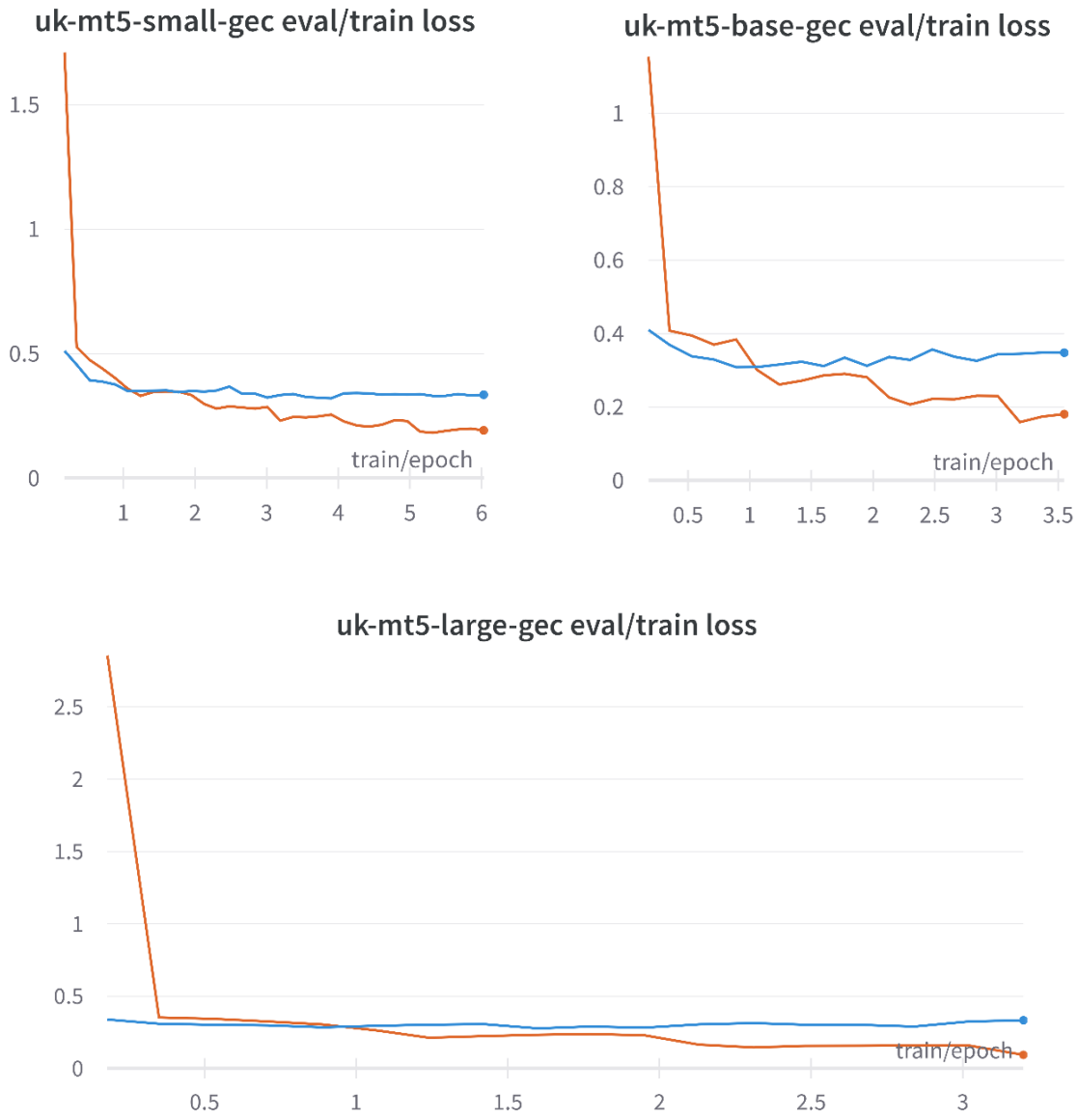


Рисунок 4.12 – Оцінка функції втрат на навчальному та валідаційному наборах даних в процесі навчання моделей групи uk-gec-mT5

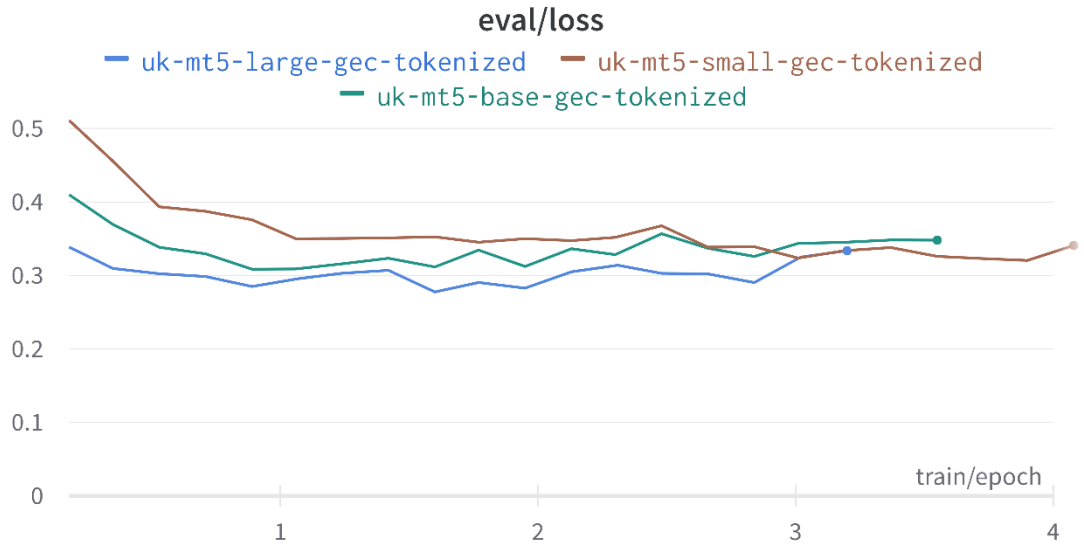


Рисунок 4.13 – Порівняння значень функції втрат на валідаційних даних GEC моделей

Для розрахунку оцінки m2 необхідно мати текстовий файл з, відкоригованими GEC системою, реченнями, та окремий «коректний» анотований файл. В цьому файлі зафіксовані всі виправлення до кожного входового «некоректного» речення, для того щоб отримати граматично правильно побудоване речення. Для творення анотованого файлу з текстовими даними для фінального оцінювання якості моделі було використано функцію (скріпт), що оптимізовано під мову Python. Цей скріпт було опубліковано у публічний доступ розробниками корпусу UA-GEC. На рисунку 4.14 наведено приклад згенерованого анотованого речення у форматі метрики m2.

Для розрахунку безпосередньо самої метрики m2 було використано скрип m2scorer.py, що було взято на офіційному Github репозиторії [46] авторів метрики m2. В таблиці 4.4 наведено значення метрик оцінки якості для кожної з натренованих моделей.

```
S Головне - що хоч мені вже й запізно , але кілька разів хотілось кинути
на деякий час фінтехи та компілятори і піти малювати ..
A 1 1|||Punctuation|||,|||REQUIRED|||-NONE-|||2
A 1 2|||Punctuation|||-NONE-|||REQUIRED|||-NONE-|||2
A 12 14|||F/Style|||хотілося покинути|||REQUIRED|||-NONE-|||2
A 24 24|||Punctuation|||.|||REQUIRED|||-NONE-|||2
```

Рисунок 4.14 – Приклад анотованого речення під формат метрики m2

Таблиця 4.4 – Кількісні значення метрик, які було розраховано m2 метрикою навчених GEC систем на анотованих даних

Модель	Влучність	Повнота	F-міра
uk-gec-mT5$_{SMALL}$	0.4703	0.1864	0.3605
uk-gec-mT5$_{BASE}$	0.4824	0.1982	0.3749
uk-gec-mT5$_{LARGE}$	0.4674	0.2488	0.3976

Метрика m2 насправді не є універсальною та досконалою для визначення якості систем виправлення граматичних помилок. Оцінка довгих коректних синтаксичних виправлень може оцінюватись так само, як виправлення тільки одно знаку пунктуації, це є нерівносильним по складності. Чисельні результати будуть сильно залежати від того, як було створено виходовий файл, в якому міститься опис виправлень, що було зроблено автоматично або вручну (дата-анотатором, в таких умовах рівень експертизи з лінгвістиці буде відігравати велику роль). Ще однією проблемою F-міра є те, що вона не враховує різницю між «без змін» і «неправильними змінами», зробленими системами.

Альтернативою F-міри є метрика GLEU, яка не вимагає долучення дата-анотаторів до розмітки текстів, тобто тут усувається людський фактор. Ця метрика побудована на базі модифікованих n-грам. У таблиці 4.5 представлені чисельні результати оцінювання натренованих моделей з різними значеннями максимального n-граму.

Таблиця 4.5 – Результати оцінювання моделей за допомогою метрики GLEU

Модель	Максимальна n-грама		
	2	3	4
uk-gec-mT5$_{SMALL}$	0.815	0.697	0.578
uk-gec-mT5$_{BASE}$	0.832	0.720	0.605
uk-gec-mT5$_{LARGE}$	0.903	0.789	0.636

По чисельним результатам можна зробити висновок, що зі збільшення кількості параметрів в моделі її якість вирішення завдання виправлення граматичних помилок збільшується, хоча на невеликий відсоток. Через обмеження по ресурсам відео картки в цій роботі моделі з 3.7 (mT5-XL) чи 13 (mT5-XXL) мільярдів параметрів, використовуючи які при реалізації GEC систем для інших мов (англійська, німецька, російська) були отримані значно кращі результати, використати було не можливо.

4.9 Тренування моделей на аугментованих даних

Під час імітаційного моделювання було проведено серію експериментів, використовуючи повністю аугментовані дані, а також ще одну серію навчання на частково аугментованих даних і анотованих наборах даних. Під час імітаційного моделювання було натреновано модель uk-gec-mT5$_{BASE}$. На рисунку 4.15 наведено час навчання моделі u-mT5-base на повністю аугментованих даних, кількістю 100.000 та 500.000 речень.

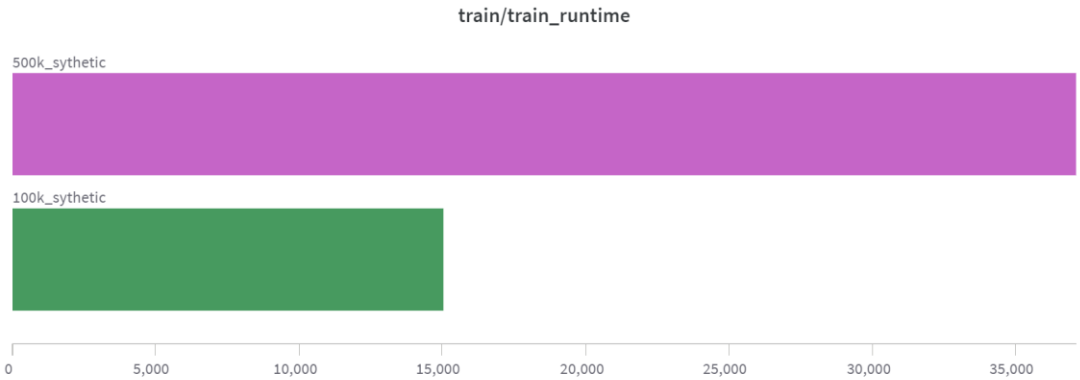


Рисунок 4.15 – Середній час навчання моделі uk-mT5$_{BASE}$ аугментованих даних

Загальний час навчання моделі uk-gec-mT5$_{BASE}$, що навчалась на повністю аугментованих даних, 4 години 2 хвилини та 9 годин 18 хвилин відповідно. На рисунку 4.16 наведено порівняння метрики оцінки функції втрат на валідаційних даних при навчанні моделі uk-gec-mT5$_{BASE}$ на повністю аугментованих даних.

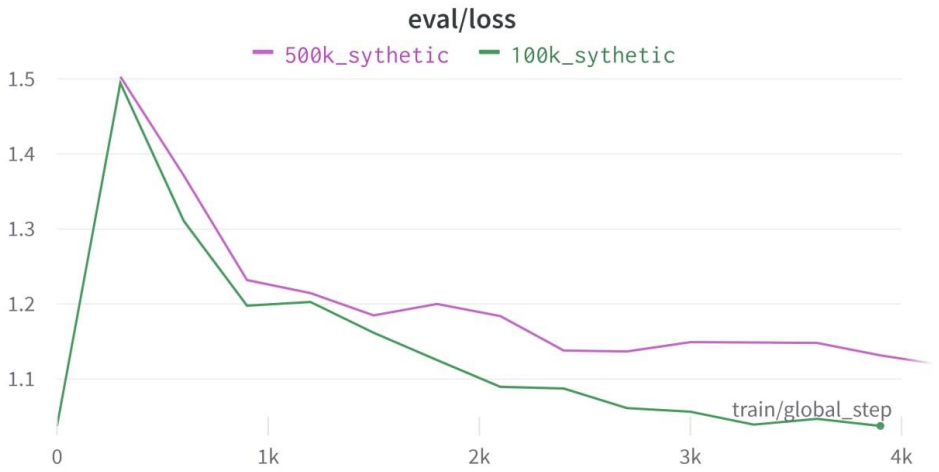


Рисунок 4.16 – Порівняння метрик оцінки функції втрат при навчанні моделі uk-gec-mT5$_{BASE}$ на повністю аугментованих даних

З графіку оцінки функції втрат, можна зазначити, що з великою кількістю аугментованих даних, процес оцінки якості моделі на валідаційних даних буде достатньо довго тривалим, а розрив між реальними результатами, що показала модель і тими результатами які бажано отримати буде великим.

Оцінивши результати навчання моделі, що наведено на рисунку 4.17, подальші експерименти проводились на частково аугментованих даних. Кількість аугментованих даних для подальшого навчання – 150 тис. речень. На наступному етапі імітаційного моделювання було навчено моделі uk-gec-mT5$_{BASE}$ та uk-gec-mT5$_{SMALL}$ на частково аугментованих даних. Навчання моделей uk-gec-mT5$_{BASE}$ та uk-gec-mT5$_{SMALL}$ зайняло орієнтовно 6 годин та 4 години відповідно. На рисунка 4.17 наведено графік навчання моделей.

На рисунку 4.18 зображено метрики оцінки функції втрат на валідаційних даних при навчанні моделей uk-gec-mT5$_{BASE}$ та uk-gec-mT5$_{SMALL}$ на частково аугментованих даних. Найкращім результатом оцінки функції втрат було 0.273.

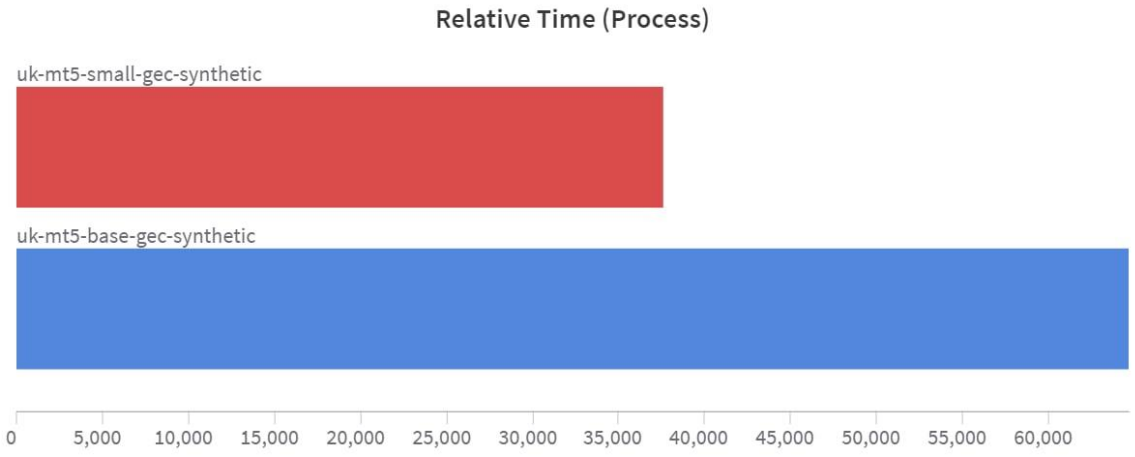


Рисунок 4.17 – загальний час навчання моделей uk-gec-mT5$_{BASE}$ та uk-gec-mT5$_{SMALL}$ на частково аугментованих даних

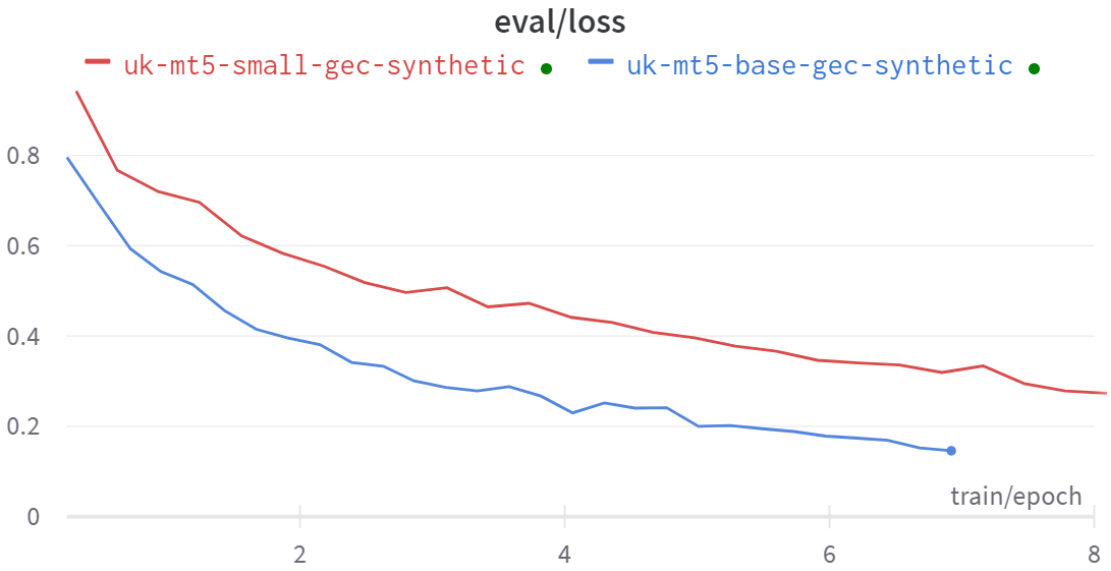


Рисунок 4.18 – Оцінка функції втрат на валідаційних даних при навчанні моделей з використанням частково аугментованих даних

На рисунку 4.19 наведено метрики оцінки функції втрат на валідаційних даних при навчанні моделей на анотованих та аугментованих даних. На наведеному графіку можна помітити, що результати навчання моделей, які тренувались на частково аугментованих даних мали менші значення, хоча для цього знадобилось більше епох навчання.

Якщо порівнювати отримані результати з результатами, які були отримані протягом навчання моделей виключно на анотованих даних, можна зробити висновок, що відбувається незначне покращення якості моделей. Але проблема полягає в тому, що для того щоб підготувати більшу кількість анотованих та аугментованих даних, а потім і для навчання більшої моделі необхідно використати і більшу кількість ресурсів. Чисельні результати наведено у таблиці 4.6 та 4.7.

Таблиця 4.6 – Результати оцінювання якості моделей метрикою m2 на частково аугментованих даних

Модель GEC	Влучність	Повнота	F-міра
uk-gec-mT5$_{SMALL}$	0.4948	0.2015	0.3832
uk-gec-mT5$_{BASE}$	0.5071	0.2203	0.4023

Таблиця 4.7 – Результати оцінювання якості моделей метрикою GLEU на частково аугментованих даних

Модель GEC	Максимальна n-грама		
	2	3	4
uk-gec-mT5$_{SMALL}$	0.876	0.723	0.608
uk-gec-mT5$_{BASE}$	0.915	0.774	0.643

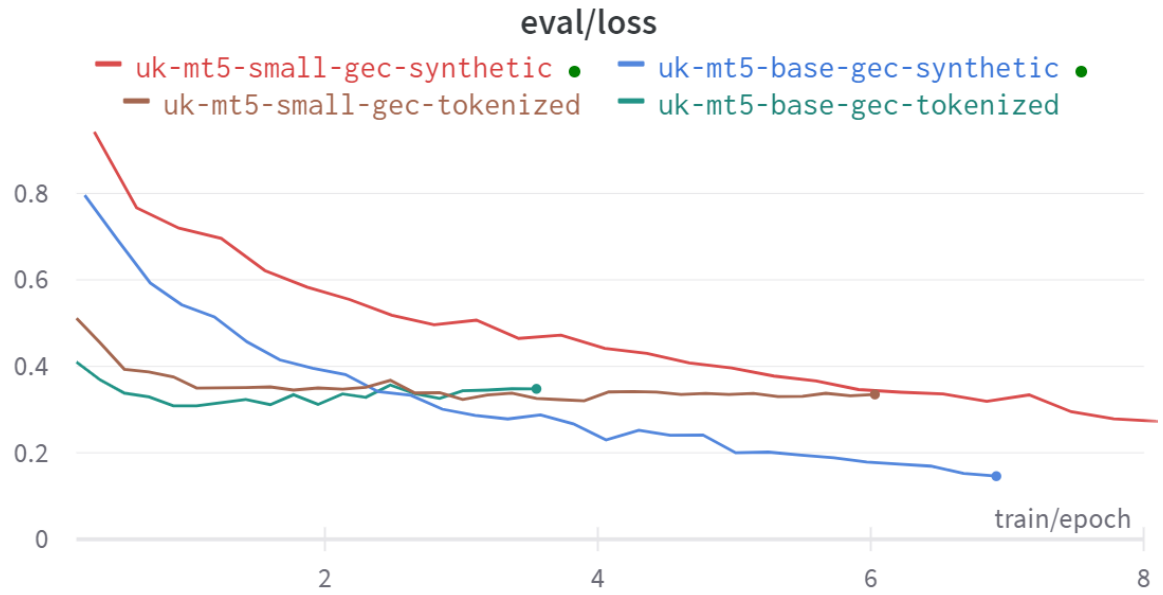


Рисунок 4.19 – Порівняння значень функції втрат на валідаційних даних при навчанні моделей з використанням анотованих та частково аугментованих даних

4.10 Порівняльний аналіз якості GEC систем

На сьогоднішній день провести порівняльний аналіз українських GEC систем складно через те, що таких систем під українську мову, що опубліковані у відкритому доступі не існує. Тому, для проведення порівняння якості створеної моделі з якимись аналогами, було взято декілька моделей GEC для інших мов (схожих за морфологією з українською) з обмеженими ресурсами (low-resource languages) [47]. У таблиці 4.8 наведена статистика GEC систем для інших мов. У таблиці 4.9 зведені фінальні результати різних варіацій навчених GEC систем для української мови.

Таблиця 4.8 – Статистика GEC систем для німецької та російської мов, оцінених метрикою m2

Модель	Кількість параметрів	Дані	Німецька	Польська
$mT5_{SMALL}$	300 мільйонів	cLANG-8 (Cleaned LANG-8 Corpus)	61.78	17.80
$mT5_{BASE}$	580 мільйонів		67.19	25.20
$mT5_{LARGE}$	1.2 мільярди		70.14	27.55
$mT5_{XL}$	3.7 мільярди		72.59	39.44
$mT5_{XXL}$	13 мільярдів		74.83	43.52

GEC моделі для німецької та польської мов навчались на 114 та 45 тисячах реченнях відповідно, кількість речень для української версії GEC моделі складає 139 тисяч. Спостерігаємо, що для усіх мов якість виправлення граматичних помилок збільшується з розміром моделі. Це свідчить про можливість подальшого покращення результатів, зокрема й для української мови.

Таблиця 4.9 – Фінальні результати GEC систем для української мови, навчених на анотованих та синтетичних реченнях

Модель	Кількість параметрів	Дані	Українська
uk-mT5$_{SMALL}$	75 мільйонів	UA-GEC анотовані + аугментовані та аугментовані Brown-UK	38.32
uk-mT5$_{BASE}$	244 мільйонів		40.23
uk-mT5$_{LARGE}$	779 мільйонів	UA-GEC анотовані	39.76

Пошук й використання більш спеціалізованих даних, для прикладу ділові листи чи наукові публікації, з попередньою обробкою та оцінкою експерта в лінгвістиці, дасть змогу виявляти та виправляти більший спектр помилок, включаючи можливість пропонувати конструкції певного стиля мовлення. Крім цього, для покращення якості системи можливо використовувати аугментовані дані, згенеровані комплекснішими алгоритмами.

ВИСНОВКИ

У кваліфікаційній роботі представлено результати, що відповідають меті дослідження, а саме – розробці інтелектуальної системи виправлення граматичних помилок для української мови. Проведені теоретичні та практичні дозволили зробити наступні висновки.

Було проаналізовано предметну галузь та існуючі підходи до вирішення задачі на виправлення граматичних помилок (GEC).

Було проведено аналіз методів та технологій для створення аугментованих даних для навчання моделі машинного навчання.

Згідно до поставленої мети кваліфікаційної роботи було виконано основні етапи розробки моделі глибинної нейромережевої системи для вирішення задачі виправлення граматичних помилок, яка навчається як на анотованому корпусі української мови, так і з використанням аугментованих та частково аугментованих даних.

Розроблена група систем GEC для української мови, де за основу взято модифіковані багатомовні моделі mT5.

Було проведено порівняльний аналіз отриманих протягом навчання та налаштування гіперпараметрів моделі результатів, як на анотованому корпусі української мови, так і з додаванням аугментованих даних, які в результаті змогли перевершити основні результати.

Було проведено порівняльний аналіз розробленої системи виправлення граматичних помилок української мови з існуючими системами для такого типу завдань, але навчених на інших мовах.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Smith, C.R., Kiefer, K.E. & Gingrich, P.S. Computers come of age in writing instruction. Comput Hum 18, 215–224 (1984). https://doi.org/10.1007/BF02267225.

2. Shaalan, Khaled. (2005). Arabic GramCheck: A grammar checker for Arabic. Software: Practice and Experience. 35. 643 – 665. 10.1002/spe.653.

3. Deksne, Daiga. «A New Phase in the Development of a Grammar Checker for Latvian.» Baltic HLT (2016).

4. Eckhard Bick. 2015. DanProof: Pedagogical Spell and Grammar Checking for Danish. In Proceedings of the International Conference Recent Advances in Natural Language Processing, pages 55–62, Hissar, Bulgaria. INCOMA Ltd. Shoumen, BULGARIA.

5. Gakis, Panagiotis & Panagiotakopoulos, Christos & Sgarbas, Kyriakos & Tsalidis, Christos & Verykios, Vassilios. (2016). Design and construction of the Greek grammar checker. Digital Scholarship in the Humanities. 32. fqw025. 10.1093/llc/fqw025.

6. Gill, Mandeep & Lehal, Gurpreet. (2008). A Grammar Checking System for Punjabi.. 149-152.

7. Kaur, Jaspreet & Garg, Kamal. (2014). Hybrid Approach for Spell Checker and Grammar Checker for Punjabi. International Journal of Computer Science and Software Engineering.

8. Wang, Yu & Wang, Yuelin & Liu, Jie & Liu, Zhuo. (2020). A Comprehensive Survey of Grammar Error Correction.

9. Naghshnejad, Mina & Nair, Vijayan. (2020). Recent Trends in the Use of Deep Learning Models for Grammar Error Handling.

10. Hyperparameter Optimization for Hugging Face Transformers, https://medium.com/distributed-computing-with-ray/hyperparameter-optimization-for-transformers-a-guide-c4e32c6c989b.

11. Rozovskaya, A., Chang, K.-W., Sammons, M., and Roth, D., «The university of illinois system in the conll-2013 shared task», in Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task, pp. 13–19, 2013.

12. Dahlmeier, D., and Ng, H. T., «A beam-search decoder for grammatical error correction», in Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (Association for Computational Linguistics). pp. 568–578, 2012.

13. Brockett, C., Dolan, W. B., and Gamon, M., «Correcting esl errors using phrasal smt techniques», in Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics (Association for Computational Linguistics). pp. 249–256, 2006.

14. Tomoya Mizumoto, Yuta Hayashibe, Mamoru Komachi, Masaaki Nagata, and Yuji Matsumoto. 2012. The Effect of Learner Corpus Size in Grammatical Error Correction of ESL Writings. In Proceedings of COLING 2012: Posters, pages 863–872, Mumbai, India. The COLING 2012 Organizing Committee.

15. I. Yoshimoto, T. Kose, K. Mitsuzawa, K. Sakaguchi, T. Mizumoto, Y. Hayashibe, M. Komachi, and Y. Matsumoto, «NAIST at 2013 CoNLL grammatical error correction shared task», in Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task. Sofia, Bulgaria: Association for Computational Linguistics, Aug. 2013, pp. 26–33.

16. M. J unczys-Dowmunt and R. Grundkiewicz, «The AMU system in the CoNLL-2014 shared task: Grammatical error correction by data-intensive and feature-rich statistical machine translation», in Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task. Baltimore, Maryland: Association for Computational Linguistics, J un. 2014, pp. 25–33.

17. M. Felice, Z. Yuan, Ø. E. Andersen, H. Yannakoudakis, and E. Kochmar, «Grammatical error correction using hybrid systems and type filtering», in Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task. Baltimore, Maryland: Association for Computational Linguistics, J un. 2014, pp. 15–24.

18. N. Durrani, A. Fraser, H. Schmid, H. Hoang, and P. Koehn, “Can Markov models over minimal translation units help phrase based SMT?” in Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers). Sofia, Bulgaria: Association for Computational Linguistics, Aug. 2013, pp. 399–405.

19. Mikolov, T., Chen, K., Corrado, G. and Dean, J. (2013) Efficient Estimation of Word Representations in Vector Space. Proceedings of Workshop at ICLR. arXiv:1301.3781v1.

20. C. Sun, X. J in, L. Lei, Y. Zhao, and X. W ang, Convolutional Neural Networks for Correcting English Article Errors. Springer International Publishing, 2015.

21. L. R. . L. H. W ang, C. , «Deep context model for grammatical error correction», in InterSpeech 2017, 2017, pp. 167–171.

22. Bryant, Christopher & Briscoe, Ted. (2018). Language Model Based Grammatical Error Correction without Annotated Training Data. 247-253. 10.18653/v1/W18-0529.

23. K. Cho, B. van Merrienboer, C¸. Gülc¸ehre, F. Bougares, H. Schwenk, and Y. Bengio, «Learning phrase representations using RNN encoder-decoder for statistical machine translation», CoRR, 2014.

24. Z. Yuan, «Grammatical error correction in nonnative English», University of Cambridge, Computer Laboratory, Tech. Rep., Mar. 2017.

25. Z. Yuan and T. Briscoe, «Grammatical error correction using neural machine translation», in Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human

Language Technologies. San Diego, California: Association for Computational Linguistics, Jun. 2016, pp. 380– 386.

26. S. Chollampatt and H. T. Ng, «A multilayer convolutional encoder-decoder neural network for grammatical error correction», CoRR, 2018.

27. J . Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, «Convolutional sequence to sequence learning», CoRR, 2017.

28. A. Vaswani, N. Shazeer, N. Parmar, J . Uszkoreit, L. J ones, A. N. Gomez, L. Kaiser, and I. Polosukhin, «Attention is all you need», CoRR, 2017.

29. M. Junczys-Dowmunt, R. Grundkiewicz, S. Guha, and K. Heafield, «Approaching neural grammatical error correction as a low-resource machine translation task», in Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers). New Orleans, Louisiana: Association for Computational Linguistics, J un. 2018, pp. 595–606.

30. J . Lichtarge, C. Alberti, S. Kumar, N. Shazeer, and N. Parmar, «Weakly supervised grammatical error correction using iterative decoding», CoRR, 2018.

31. Y. J . Choe, J . Ham, K. Park, and Y. Yoon, «A neural grammatical error correction system built on better pre-training and sequential transfer learning», in Proceedings of the Fourteenth W orkshop on Innovative Use of NLP for Building Educational Applications. Florence, Italy: Association for Computational Linguistics, Aug. 2019, pp. 213–227.

32. A. Awasthi, S. Sarawagi, R. Goyal, S. Ghosh, and V. Piratla, «Parallel iterative edit models for local sequence transduction», in Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International J oint Conference on Natural Language Processing (EMNLP-IJ CNLP). Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 4260– 4270.

33. Omelianchuk, Kostiantyn & Atrasevych, Vitaliy & Chernodub, Artem & Skurzhanskyi, Oleksandr. (2020). GECToR -- Grammatical Error Correction: Tag, Not Rewrite.

34. Tarnavskyi, Maksym. «Improving Sequence Tagging for Grammatical Error Correction» (2021).

35. González García, Cristian & Núñez Valdez, Edward & García Díaz, Vicente & Pelayo García-Bustelo, B. & Cueva Lovelle, Juan. (2018). A Review of Artificial Intelligence in the Internet of Things. International Journal of Interactive Multimedia and Artificial Intelligence. 5. 1.

36. Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 1746–1751, Doha, Qatar. Association for Computational Linguistics.

37. Lucy, Li & Gauthier, Jon. (2017). Are Distributional Representations Ready for the Real World? Evaluating Word Vectors for Grounded Perceptual Meaning. 76-85. 10.18653/v1/W17-2810.

38. Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

39. Oleksiy Syvokon and Olena Nahorna. 2021.UA-GEC: Grammatical Error Correction andFluency Corpus for the Ukrainian Language.CoRR, abs/2103.16997v1.

40. Ruder, Sebastian. (2016). An overview of gradient descent optimization algorithms.

41. Kingma, Diederik & Ba, Jimmy. (2014). Adam: A Method for Stochastic Optimization. International Conference on Learning Representations.

42. Andrews, Gerard. «What Is Synthetic Data?» NVIDIA Blog, 8 June 2021, https://blogs.nvidia.com/blog/2021/06/08/what-is-synthetic-data/.

43. Artsrouni, Georges. «History of natural language processing.» Wikipedia, https://en.wikipedia.org/wiki/History_of_natural_language_processing.

44. Brownlee, Jason. «What Is Natural Language Processing?» Machine Learning Mastery, 22 September 2017, https://machinelearningmastery.com/natural-language-processing/.

45. «Building a Large Annotated Corpus of Learner English: The NUS Corpus of Learner English.» ACL Anthology, https://aclanthology.org/W13-1703/.

46. «Developing an Automated Writing Placement System for ESL Learners Helen Yannakoudakis1,2, Øistein E. Andersen1,2, Ardeshir Gera.» Department of Computer Science and Technology |, https://www.cl.cam.ac.uk/~hy260/WI-cefr.pdf.

47. Foote, Keith D. «A Brief History of Natural Language Processing (NLP) – DATAVERSITY.» Dataversity, 22 May 2019, https://www.dataversity.net/a-brief-history-of-natural-language-processing-nlp/#.

48. Lutkevich, Ben. “What is Natural Language Processing? An Introduction to NLP.” TechTarget, https://www.techtarget.com/searchenterpriseai/definition/natural-language-processing-NLP.

49. Menzli, Amal. «Tokenization in NLP: Types, Challenges, Examples, Tools – neptune.ai.» Neptune.ai, 13 December 2021, https://neptune.ai/blog/tokenization-in-nlp.

50. «A New Dataset and Method for Automatically Grading ESOL Texts.» ACL Anthology, https://aclanthology.org/P11-1019/.

51. «NLP – overview.» Stanford Computer Science, https://cs.stanford.edu/people/eroberts/courses/soco/projects/2004-05/nlp/overview_history.html.

52. Searle, John. «Natural language processing.» Wikipedia, https://en.wikipedia.org/wiki/Natural_language_processing.

53. «Synthetic data.» Wikipedia, https://en.wikipedia.org/wiki/Synthetic_data.

54. Syvokon, Oleksiy, and Olena Nahorna. [2103.16997] UA-GEC: «Grammatical Error Correction and Fluency Corpus for the Ukrainian Language.» arXiv, 31 March 2021, https://arxiv.org/abs/2103.16997.

55. «Ukrainian Grammatical Error Correction Dataset.» Grammarly, 30 August 2021, https://www.grammarly.com/blog/engineering/announcing-ua-gec/.

56. «What is Natural Language Processing?» IBM, 2 July 2020, https://www.ibm.com/cloud/learn/natural-language-processing.

57. «What is Tokenization | Tokenization In NLP.» Analytics Vidhya, 26 May 2020, https://www.analyticsvidhya.com/blog/2020/05/what-is-tokenization-nlp/.

58. Abdaoui, Amine, Camille Pradel and Grégoire Sigel. «Load What You Need: Smaller Versions of Mutlilingual BERT.» SUSTAINLP (2020).

59. How to adapt a multilingual T5 model for a single language | by David Dale | Towards Data Science (medium.com), https://towardsdatascience.com/how-to-adapt-a-multilingual-t5-model-for-a-single-language-b9f94f3d9c90.

60. Ground Truth for Grammatical Error Correction Metrics by Courtney Napoles, Keisuke Sakaguchi, Joel Tetreault, and Matt Post.

61. Daniel Dahlmeier and Hwee Tou Ng. 2012. Better Evaluation for Grammatical Error Correction. In Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL 2012).

62. Sascha Rothe, Jonathan Mallinson, Eric Malmi, Sebastian Krause, and Aliaksei Severyn. 2021. A Simple Recipe for Multilingual Grammatical Error Correction.

63. Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. 2021. mT5: A Massively Multilingual Pre-trained Text-to-Text Transformer.

64. Mihir Kale and Abhinav Rastogi. 2020. Text-to-Text Pre-Training for Data-to-Text Tasks.

65. MaxMatch (M^2) Scorer – Evaluation program for grammatical error correction systems, https://github.com/nusnlp/m2scorer.

66. Mathematical Introduction to GloVe Word Embedding, https://becominghuman.ai/mathematical-introduction-to-glove-word-embedding-60f24154e54c.

67. Visualization of Word Embedding Vectors using Gensim and PCA, https://laptrinhx.com/visualization-of-word-embedding-vectors-using-gensim-and-pca-3760734903/.

68. Visualizing A Neural Machine Translation Model, https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/.

69. Analysis of BERT Fine-tuning Mathematical principle, https://www.codestudyblog.com/cs2201ai/40115174706.html.

70. Hyperparameter Tuning Black Magic, https://community.alteryx.com/t5/Data-Science/Hyperparameter-Tuning-Black-Magic/ba-p/449289.

ДОДАТОК А
Вихідний код програми

```
public abstract class ClusteringAlgorithm {
    /**
     * Data set of algorithm.
     */
    private DataSet dataSet;
    /**
     * Data loader of algorithm.
     */
    private DataLoader dataLoader;
    /**
     * Centroids set of algorithm.
     */
    protected CentroidsSet centroidsSet;
    /**
     * Matrix of belongs of algorithm.
     */
    protected BelongingMatrix belongingMatrix;
    /**
     * Epsilon value of algorithm (Condition to stop the algorithm when difference between centroids of two
iterations is lower than epsilon).
     * Default value is 0.01.
     */
    private double epsilon = 0.01;
    /**
     * Centroids number of algorithm.
     */
    private int centroidsNumber;
    /**
     * Amount of data in percents to process (learn).
     * @since 1.2.
     */
    private double processedDataAmount;

    /**
     * Constructor that creates algorithm with specified number of clusters.
     * @param clustersNumber    Clusters number.
     */
    public ClusteringAlgorithm(int clustersNumber){
        centroidsNumber = clustersNumber;
    }

    /**
     * Gets an amount of data in percents to process.
     * @return Returns an amount of data in percents to process.
     * @since 1.2.
     */
    public double getProcessedDataAmount() {
        return processedDataAmount;
    }

    /**
     * Sets an amount of data in percents to process.
     * @param aProcessedDataAmount    Amount of data in percents to process.
     * @since 1.2.
     */
    public void setProcessedDataAmount(double aProcessedDataAmount) {
        processedDataAmount = aProcessedDataAmount;
    }
```

```
    /**
     * Loading the data file with specified file path and splitter.
     * @param aFilePath    Data file path to load.
     * @param aSplitter    Splitter of vector of values.
     */
    public void loadFile(String aFilePath, String aSplitter){
        dataLoader = new DataLoader(aFilePath, aSplitter, processedDataAmount);
        dataLoader.loadFile();
    }

    /**
     * Loading the image file with specified file path.
     * @param aFilePath    Image data file path to load.
     * @since 1.1.
     */
    public void loadImage(String aFilePath){
        dataLoader = new DataLoader(aFilePath);
        dataLoader.setProcessedDataAmount(processedDataAmount);
        dataLoader.loadImageFile();
    }

    /**
     * Fills the DataSet from data file with specified size of matrix of values.
     * @param vectorLength    Length of the DataObject's vector (if vector needed).
     *                        If 0, the length of vector will be determined depending on data file contain (Vector length
will be the same as number of values in a string. In each string must be an equal number of values).
     */
    public void fillDataSet(int vectorLength){
        dataSet = dataLoader.fillDataSet(vectorLength);
        initCentroidsAndBelong();
    }

    /**
     * Fills the DataSet from data file with specified size of matrix of values.
     * @param matrixHeight    Height of the DataObject's matrix. Can not be 0.
     *                        You can use fillDataSet(int vectorLength) in case if matrixHeight is equals to 1 (if vector
needed).
     * @param matrixWidth    Width of the DataObject's matrix or length of the vector (if vector needed).
     *                        If 0, the length of vector will be determined depending on data file contain (Vector length
will be the same as number of values in a string. In each string must be an equal number of values).
     *                        If 0, matrixHeight must be 1.
     */
    public void fillDataSet(int matrixHeight, int matrixWidth){
        dataSet = dataLoader.fillDataSet(matrixHeight, matrixWidth);
        initCentroidsAndBelong();
    }

    /**
     * Fills the DataSet from image file with specified size of matrix of values. Use after loading of image file.
     * @param matrixHeight    Height of the DataObject's matrix. Can not be 0. If vector needed set equals to 1.
     * @param matrixWidth     Width of the DataObject's matrix or length of the vector (if vector needed). Can
not be 0.
     * @since 1.1.
     */
    public void fillDataSetFromImage(int matrixHeight, int matrixWidth){
        dataSet = dataLoader.fillDataSetFromImage(matrixHeight, matrixWidth);
        initCentroidsAndBelong();
    }

    /**
     * Initializes of centroids set and matrix of belonging values.
     */
```

```
private void initCentroidsAndBelong(){
    dataSet.setMatrixSize();
    dataSet.centralizeAll();
    dataSet.normalizeAll();
    centroidsSet = new CentroidsSet(centroidsNumber, dataSet.getMatrixHeight(), dataSet.getMatrixWidth());
    belongingMatrix = new BelongingMatrix(dataSet.getObjectsNumber(), centroidsNumber);
    calculateCentroids();
}

/**
 * Prints the matrix of values of each DataObject in the list with their indexes.
 */
public void printDataSet(){
    dataSet.printDataSet();
}

/**
 * Prints the matrix of values of each centroid (DataObject) in the list with their indexes.
 */
public void printCentroidsSet(){
    centroidsSet.printCentroidsSet();
}

/**
 * Prints the list of vector of belongs with their indexes.
 */
public void printBelongingMatrix(){
    belongingMatrix.printBelongingMatrix();
}

/**
 * Prints the vector of the clusters with the maximum belong value for each object.
 */
public void printBelongClusters(){
    belongingMatrix.printBelongClusters();
}

/**
 * Gets DataSet of algorithm.
 * @return Returns link on DataSet of algorithm.
 */
public DataSet getDataSet() {
    return dataSet;
}

/**
 * Gets DataLoader of algorithm.
 * @return Returns link on DataLoader of algorithm.
 */
public DataLoader getDataLoader() {
    return dataLoader;
}

/**
 * Gets CentroidsSet of algorithm.
 * @return Returns link on CentroidsSet of algorithm.
 */
public CentroidsSet getCentroidsSet() {
    return centroidsSet;
}

/**
 * Gets BelongingMatrix of algorithm.
```

```
 * @return Returns link on BelongingMatrix of algorithm.
 */
public BelongingMatrix getBelongingMatrix() {
   return belongingMatrix;
}

/**
 * Gets epsilon of algorithm.
 * @return Returns epsilon value of algorithm.
 */
public double getEpsilon() {
   return epsilon;
}

/**
 * Sets epsilon of algorithm.
 * @param anEpsilon    An epsilon value (condition) of algorithm.
 */
public void setEpsilon(double anEpsilon) {
   epsilon = anEpsilon;
}

/**
 * Gets centroids number of algorithm.
 * @return Returns centroids number of algorithm.
 */
public int getCentroidsNumber() {
   return centroidsNumber;
}

/**
 * Sets centroids number of algorithm.
 * @param aCentroidsNumber    Centroids number of algorithm.
 */
public void setCentroidsNumber(int aCentroidsNumber) {
   centroidsNumber = aCentroidsNumber;
}

/**
 * Abstract method of calculating of matrix of belong.
 * @param isProcessedToCalculate    Is to calculate belong vector for object processed due data analysis.
 */
public abstract void calculateBelongingMatrix(boolean isProcessedToCalculate);

/**
 * Abstract method of calculating of centroids matrices.
 */
public abstract void calculateCentroids();

/**
 * Iterative method of algorithm execution.
 * Iteratively calculates matrix of belong values and centroids.
 * Prints number of iteration, current value of epsilon, and time elapsed for iteration execution.
 * @param anEpsilonValue    An epsilon value (condition) of algorithm.
 * @since 1.1.
 */
public void doAlgorithm(double anEpsilonValue){
   epsilon = anEpsilonValue;
   doAlgorithm();
}

/**
 * Iterative method of algorithm execution.
```

```
    * Iteratively calculates matrix of belong values and centroids.
    * Prints number of iteration, current value of epsilon, and time elapsed for iteration execution.
    */
   public void doAlgorithm(){
      int iterationNumber = 0;
      while(centroidsSet.calculateDifferenceBetweenIterations() > epsilon){
         long currentTime = System.nanoTime();
         calculateBelongingMatrix(true);
         calculateCentroids();
         iterationNumber++;
         System.out.println(String.format("Iteration: %1$-4d ε: %2$-10.6f Time: %3$.5f s", iterationNumber,
centroidsSet.calculateDifferenceBetweenIterations(), ((double)(System.nanoTime() – currentTime) /
Math.pow(10, 9))));
         if(iterationNumber == 1 || iterationNumber ==
20){calculateBelongingMatrix(false);printBelongingMatrix();}
      }
      calculateBelongingMatrix(false);
   }

   /**
    * Changing of the size of the each matrix of values and centroids.
    * Prints time of transposing.
    * Not works if the matrix of size is zero or the product of the old sizes not equals to the product of the new
sizes.
    * @param newMatrixHeight    The new matrix height of data objects.
    * @param newMatrixWidth     The new matrix width of data objects.
    */
   public void transformAll(int newMatrixHeight, int newMatrixWidth){
      long currentTime = System.nanoTime();
      dataSet.transformAll(newMatrixHeight, newMatrixWidth);
      centroidsSet.transformAll(newMatrixHeight, newMatrixWidth);
      System.out.println(String.format("Data transformation time: %1$.5f s", ((double)(System.nanoTime() –
currentTime) / Math.pow(10, 9))));
   }

   /**
    * Gets membership matrix of patterns of data.
    * @return Membership matrix of patterns.
    * @since 1.1.
    */
   public double[][] getMembershipMatrix(){
      return belongingMatrix.getMembershipMatrix();
   }

   /**
    * Gets the vector of the clusters with the maximum belong value for each object.
    * @return Returns the vector of the clusters with the maximum belong value for each object.
    * @since 1.1.
    */
   public double[][] getMemberClusters(){
      return belongingMatrix.getMemberClusters();
   }

   /**
    * Shows frame with original picture.
    * @since 1.1.
    */
   public void showOriginalImage(){
      ImageFrame i = new ImageFrame(dataLoader.getDataImageFile(), "Original Image");
   }

   /**
    * Get color of each cluster in array to show result of clustering in image.
```

```
     * @return Returns color of each cluster in array to show result of clustering in image.
     * @since 1.1.
     */
    public int[] getClusterColor(){
        double[] clusterColor = new double[centroidsNumber];
        for(int i = 0; i < clusterColor.length; i++){
            for(int j = 0; j < dataSet.getObjectsNumber(); j++){
                if(belongingMatrix.getBelongVector(j).getBelongCluster() == i){
                    clusterColor[i] = dataSet.getDataObject(j).getAverageColorValue();
                    break;
                }
            }
        }
        int[] clusterColorToReturn = new int[centroidsNumber];
        /*for(int i = 0; i < centroidsNumber; i++){
            clusterColorToReturn[i] = (int)(255 * clusterColor[i]);
        }*/
        int minIndex = 0;
        for(int i = 0; i < centroidsNumber; i++){
            minIndex = 0;
            for(int j = 0; j < centroidsNumber; j++){
                if(clusterColor[j] < clusterColor[minIndex])
                    minIndex = j;
            }
            clusterColorToReturn[minIndex] = i * 255 / (centroidsNumber – 1);
            clusterColor[minIndex] = 2;
        }
        return clusterColorToReturn;
    }

    /**
     * Generates image after processing, with cluster colours.
     * @return Returns image after processing, with cluster colours.
     * @since 1.1.
     */
    public BufferedImage generateProcessedImage(){
        BufferedImage imageToReturn = new BufferedImage((dataLoader.getDataImageFile().getWidth() +
dataSet.getMatrixWidth() – 1)/dataSet.getMatrixWidth(), (dataLoader.getDataImageFile().getHeight() +
dataSet.getMatrixHeight() – 1)/dataSet.getMatrixHeight(), BufferedImage.TYPE_INT_RGB);
        int[] clusterColor = getClusterColor();
        for(int i = 0; i < imageToReturn.getHeight(); i++){
            for(int j = 0; j < imageToReturn.getWidth(); j++){
                imageToReturn.setRGB(j, i, clusterColor[belongingMatrix.getBelongVector(i *
imageToReturn.getWidth() + j).getBelongCluster()] * 65793);
            }
        }
        return imageToReturn;
    }

    /**
     * Shows frame with original picture.
     * @since 1.1.
     */
    public void showProcessedImage(){
        ImageFrame i = new ImageFrame(generateProcessedImage(), "Processed Image");
```

ДОДАТОК Б

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	105 с.
	Інші документи	
2.	Презентаційні матеріали	__ плакатів

.	.	Прізвище та ініціали.	Підп.	Дата			
Розробив		Яковлева О.М.			Вирішення задачі виправлення граматичних помилок в текстах з використанням сучасних технологій глибинного навчання	Шифр групи	Код напр./спец.
Перевірив		Шевченко О.Ю.				СШІздм-21-1	122
Н.контр.		Малєєва І.А.				ХНУРЕ кафедра ШІ	
Затв.		Філатов В.О.					