КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Міністерство освіти і науки України

Кваліфікаційна наукова праця

на правах рукопису

ВРУБЛЕВСЬКИЙ ВІТАЛІЙ НАТАНОВИЧ

УДК 004.891

ДИСЕРТАЦІЯ

МОДЕЛІ ПРЕДСТАВЛЕННЯ СЕМАНТИКИ РЕЧЕНЬ ПРИРОДНОЇ МОВИ

122 Комп’ютерні науки

Подається на здобуття наукового ступеня доктора філософії.

Дисертація містить результати власних досліджень.
Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело _____ Врублевський В.Н.

Науковий керівник: д.ф-м.н., професор Марченко О.О.

Київ - 2024

АНОТАЦІЯ

Врублевський В.Н. Моделі представлення семантики речень природної мови. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії за спеціальністю 112 «Комп'ютерні науки». – Київський національний університет імені Тараса Шевченка, Київ, 2024.

Робота присвячена дослідженню побудови моделей представлення семантики речень текстів природної мови із застосуванням синтаксичної структури речення як ключової ознаки.

У **вступі** розкрито сутність і актуальність наукової проблематики, що досліджується. Обґрунтовано вибір теми, визначено мету, об'єкт, предмет, методи дослідження, розкрито наукову новизну дослідження, особистий внесок здобувача, зазначено інформацію про впровадження і апробацію результатів.

Актуальність теми. Дослідження семантичних представлень у моделях обробки природної мови – ключове у галузі комп'ютерної лінгвістики та штучного інтелекту.

Векторні представлення речення відіграють ключову роль у розумінні нюансів текстів. Удосконалення цих представлень допомагає глибше розуміти текстову інформацію та впливає на інші завдання: машинний переклад, класифікацію тексту та інші.

Застосування мовних моделей продовжує розширюватися, що вимагає моделей, які вміло розуміють і створюють текст у різних мовах і контекстах. Таким чином, дослідження синтаксичної структури речення в побудові моделей семантичного представлення має важливе значення для вдосконалення моделей розуміння мови.

Мета дослідження. Метою дисертаційного дослідження є створення методів побудови векторного представлення речення на основі різних моделей; дослідження використання синтаксичних графових структур репрезентації речення та їх застосування у моделях.

Об’єктом дослідження є методи побудови векторних представлень речень природної мови.

Предметом дослідження є процес проєктування різноманітних моделей (машинного навчання, нейронних мереж різної архітектури) для побудови векторних представлень речень природної мови; їх застосування для розв’язання прикладних задач, таких як виправлення граматичних помилок та ідентифікація парафраз.

У першому розділі проведено дослідження наявних моделей представлення семантики речень. Оскільки представлення слів є важливим елементом при побудові представлення речення, проведено короткий огляд та класифікацію методів побудови представлення слів (WordNet, One-hot вектор, Word2Vec та інші). Проаналізовано методи представлення структури речень, такі як синтаксичне дерево розбору, дерево залежностей, AMR граф.

Також оглянуто та досліджено різні наявні методи побудови векторних представлень речень та вивчено методи оцінки якості векторних представлень речень. Однією з практичних задач для цієї оцінки було обрано ідентифікації парафраз. В результаті проведено огляд корпусів даних для неї та класифікацію наявних методів розв’язання.

У другому розділі розглянуто синтаксичну компоненту в моделях представлення речень. Досліджено різні методи модифікації алгоритму Ерлі та запропоновано метод коригуючого парсера з використанням попередньо обробленої вхідної граматики послідовності.

Розроблено модифікацію алгоритму Ерлі для роботи з великими граматиками, використовуючи підхід «повернення назад» (back-tracking) та знаходження декількох дерев виводу послідовності для неоднозначних граматик.

Запропоновано коригуючий парсер на основі алгоритму Ерлі для виправлення помилок вставки, видалення, заміни терміналів та зміни порядку нетерміналів, та досліджено його роботу на малих та великих граматиках.

У якості перевірки на практичній задачі проведено експеримент та зроблено порівняння коригуючого парсера з іншими системами виправлення граматичних помилок.

У третьому розділі проаналізовано використання дерева залежностей для репрезентації структури речення. Головним висновком є експериментальне підтвердження того, що поєднання дерев залежностей та векторного представлення слів можна ефективно використовувати для побудови якісних моделей представлення семантики речень.

В результаті розроблено та реалізовано кілька алгоритмів обходу та агрегування ознак із застосуванням дерева залежностей, зокрема створення метрики для порівняння підграфів та шляхів у деревах. В ході експериментів досліджено ефективність запропонованих ознак.

У четвертому розділі проаналізовано різні моделі з архітектурою Трансформер, а також використання матриці на основі дерева залежностей, як додаткової ознаки для моделі. Експериментальним чином було підтверджено, що дерева залежностей можуть покращити базові моделі з архітектурою Трансформер.

Для цього досліджено та проаналізовано різні моделі з архітектурою Трансформер та ефективність цих моделей для ідентифікації парафраз.

Також досліджено та проаналізовано використання LLM моделей, таких як Llama 2 для класифікації речень, а саме задачі ідентифікації парафраз.

Створено модель, використовуючи ознаки на основі дерева залежностей у self – attention шарі, та в ході експериментів досліджено її ефективність.

У висновках окреслено основний зміст отриманих наукових результатів, а саме:

1. Проведений аналіз синтаксичної компоненти в моделях представлення семантики речення за допомогою синтаксичного дерева розбору свідчить про доцільність її використання для створення повної моделі представлення речення. Експериментальне дослідження ефективності застосування даного представлення для

розв’язання задачі граматичної корекції речення вказало на доцільність використання синтаксичної компоненти.

2. Побудована модель на основі дерева залежностей як головної структури для репрезентації речення. Побудована модель на основі цих ознак для ідентифікації парафраз з використанням методів машинного навчання (таких як SVM). Вона ефективно вирішує поставлену задачу і досягає конкурентних результатів у цьому класі моделей.
3. Досліджено та проаналізовано різні моделі з архітектурою Трансформер та їх модифікації.
4. Запропоновано модифікацію шару self – attention з використанням ознак на основі дерева залежностей для моделей архітектури Трансформер. В результаті експериментів проаналізовано якість моделі. Запропонована модифікація показала кращі показники точності на F1, ніж базова модель. Таким чином продемонстровано доцільність застосування таких ознак для покращення розуміння структури речення.
5. Проаналізовано здатність LLM моделей до класифікації даних. В результаті експериментів розглянуто Llama 2 модель, що змогла показати співмірні результати з базовими моделями на основі архітектури Трансформер. Для такого класу задач можемо повідомити про те, що для використання та тонкого налаштування цих моделей потрібно значно більше обчислювальних ресурсів, ніж для звичайних моделей.

Ключові слова: методи побудови представлення речень, обробка природної мови, машинне навчання, нейронні мережі, моделі на основі архітектури Трансформер, синтаксичне дерево розбору, дерево залежностей.

SUMMARY

Vrublevskyi V. N. Models for representing the semantics of natural language sentences. Qualification scientific work with manuscript rights. – Qualification scientific work on the rights of the manuscript.

The PhD thesis on competition of a scientific degree of the doctor of philosophy on a specialty 122 “Computer Science”. – Taras Shevchenko National University of Kyiv, Kyiv, 2024.

The work is devoted to the study of the construction of models for representing sentence semantics in natural language texts, using the syntactic structure of the sentence as a key feature.

The **introduction** reveals the essence and relevance of the researched scientific issues. The choice of the topic is substantiated, the purpose, object, subject, and research methods are defined, the scientific novelty of the research is discussed, the personal contribution is indicated, and information about the implementation and approval of the results is indicated.

Actuality of theme. The study of semantic representations in natural language processing models is critical in computational linguistics and artificial intelligence.

Vector representations of sentences play a crucial role in understanding the nuances of texts. Improving these representations helps us better understand textual information and affects other tasks, such as machine translation and text classification.

The use of language models continues to expand, requiring models that can adeptly understand and produce text in different languages and contexts. Thus, the study of the syntactic structure of the sentence in the construction of semantic representation models is important for improving language understanding models.

The aim of the study. The dissertation research aims to create methods for constructing a vector representation of a sentence based on various models and research on the use of syntactic graph structures of sentence representation and their application in models.

The object of research is methods of constructing vector representations of natural language sentences.

The **subject of research** is the process of designing various models (machine learning, neural networks of various architectures) for building vector representations of natural language sentences and their application to solving applied problems, such as correcting grammatical errors and identifying paraphrases.

In the first chapter, a study of existing models of sentence semantic representation was conducted. Since word representation is essential in constructing a sentence representation, a brief review and classification of word representation construction methods (WordNet, One-hot vector, Word2Vec, and others) was conducted. The methods of presenting the structure of sentences, such as the parsing syntactic tree, dependency tree, and AMR graph, were analysed.

Various available methods of constructing vector representations of sentences were also reviewed and studied, as were quality assessment methods of vector representations of sentences. One of the practical tasks for this evaluation was the identification of paraphrases. As a result, a review of data corpora for it and a classification of available solution methods were conducted.

The second chapter considers the syntactic component in sentence presentation models. Various methods of modifying the Earley algorithm were investigated and a corrective parser method using a pre-processed input sequence grammar was proposed.

Early's algorithm was modified to work with large grammars, using the “back-tracking” approach and finding several sequence output trees for ambiguous grammars.

A corrective parser based on Early's algorithm is proposed to correct insertion, deletion, terminal substitution, and nonterminal reordering errors. Its performance on small and large grammars is investigated.

An experiment was conducted to test a practical task, and the correcting parser was compared with other systems for correcting grammatical errors.

The third chapter analyses the use of a dependency tree to represent the sentence structure. The main conclusion is the experimental confirmation that the combination of dependency trees and vector representation of words can be effectively used to build qualitative models of sentence semantics.

As a result, several algorithms for traversal and feature aggregation using a dependency tree were developed and implemented, in particular, creating a metric for comparing subgraphs and paths in trees. During the experiments, the effectiveness of the proposed features was investigated.

In the fourth chapter, various models with the Transformer architecture were analysed, and a matrix based on a dependency tree was used as an additional feature for the model. It has been experimentally confirmed that dependency trees can improve basic models with the Transformer architecture.

For this, various models with the Transformer architecture and the effectiveness of these models for identifying paraphrases were investigated and analysed.

LLM models, such as Llama 2, have also been investigated and analysed for sentence classification, specifically the paraphrase identification task.

A model was created using features based on the dependency tree in the self-attention layer, and its effectiveness was investigated through experiments.

In the conclusions, the main content of the obtained scientific results is outlined, namely:

1. Analysing the syntactic component in the sentence semantic representation models using the parsing syntactic tree indicates its feasibility in creating a complete sentence representation model. An experimental study of the effectiveness of using this representation to solve the problem of grammatical sentence correction indicated the feasibility of using the syntactic component.
2. The model is built using the tree of dependencies as the main structure for representing the sentence. It is built based on these features to identify paraphrases using machine learning techniques (such as SVM). It effectively solves the task and achieves competitive results in this class of models.
3. Various models with the Transformer architecture and their modifications have been studied and analysed.
4. A modification of the self-attention layer using features based on the dependency tree for Transformer architecture models is proposed. As a result

of the experiments, the quality of the model was analysed. The proposed modification showed better accuracy on F1 than the base model. In this way, the expediency of using such features to improve the understanding of the sentence structure is demonstrated.

5. The ability of LLM models to classify data was analysed. As a result of the experiments, the Llama 2 model was considered, which could show comparable results with the basic models based on the Transformer architecture. For this class of problems, using and fine-tuning these models requires significantly more computing resources than conventional models.

Keywords: methods of sentence representation, natural language processing, machine learning, neural networks, models based on Transformer architecture, parse tree, dependency tree.

Список публікацій здобувача за темою дисертації:

***Наукові праці, в яких опубліковані основні наукові результати дисертації*:**

1. Vrublevskyi, V., & Marchenko, O. (2022). Development and Analysis of a Sentence Semantics Representation Model. Cybernetics and Systems Analysis, 58(1), 16–23. https://doi.org/10.1007/s10559-022-00430-9

2. Vrublevskyi, V. N., & Marchenko, O. O. (2023). Review of approaches for paraphrase identification. Bulletin of Taras Shevchenko National University of Kyiv. Physics and Mathematics, (1), 71–78. https://doi.org/10.17721/1812-5409.2023/1.10

Наукові праці, які засвідчують апробацію матеріалів дисертації:

1. Vrublevskyi, V., & Marchenko, O. (2019). Grammar Error Correcting by the Means of CFG Parser. In *Proceedings of the 2019 IEEE International Conference On Advanced Trends In Information Theory (ATIT)*, 430–435. https://doi.org/10.1109/ATIT49449.2019.9030458
2. Vrublevskyi, V., & Marchenko, O. (2020). Paraphrase Identification Using Dependency Tree and Word Embeddings. In *Proceedings of the 2020 IEEE 2nd International Conference On Advanced Trends In Information Theory (ATIT)*, 372–375. https://doi.org/10.1109/ATIT50783.2020.9349338
3. Marchenko, O., & Vrublevskyi, V. (2023). Comparison of Transformer-based Deep Learning Methods for the Paraphrase Identification task. In *Proceedings of the 2023 X International Scientific Conference "Information Technology and Implementation" (IT&I-2023)*, 447-455. https://ceur-ws.org/Vol-3624/Short_5.pdf

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AMR – представлення абстрактного значення, мова семантичного представлення (Abstract Meaning Representation)

ANN – штучна нейромережа (Artificial Neural Network)

BiLSTM – модель штучної нейромережі, що працює за принципом двосторонньої довгої короткочасної пам'яті (Bidirectional Long Short-Term Memory)

BoW – модель «Торба слів» (Bag of Words)

CBOW – модель «Неперервна торба слів» (Continuous Bag of Words)

CNN – згорткова нейронна мережа (Convolutional Neural Network)

GEC – виправлення граматичних помилок (Grammar Error Correction)

GPU – графічний процесор (Graphics Processing Unit)

IDF – обернена частота документа (Inverse Dense Frequency)

LLM – велика мовна модель (Large Language Model)

LSTM – модель штучної нейромережі, що працює за принципом довгої короткочасної пам'яті (Long Short-Term Memory)

NLP – обробка природної мови (Natural Language Processing)

POS – частина мови (Part of Speech)

RNN – рекурентна нейронна мережа (Recurrent Neural Networks)

SVD – сингулярний розклад матриці (Singular Value Decomposition)

SVM – метод опорних векторів (Support Vector Machine)

TF-IDF – кількісна оцінка термінів в тексті (Term Frequency-Inverse Document Frequency)

TPU – тензорний блок обробки (Tensor Processing Units)

WPT – токенізація WordPiece (Word Piece Tokeniser)

ВСТУП

Обґрунтування вибору теми дослідження. Побудова моделей представлення семантики слів, речень та текстів природної мови займає сьогодні центральне місце в комп'ютерній лінгвістиці та штучному інтелекті загалом. Такі моделі, як BERT [1], RoBERTa [2], AlBERT [3] та інші змінили обробку природної мови. Вони створені найбільшими світовими IT-компаніями та обчислювалися на надпотужних ресурсах їх data-центрів. Ці багатовимірні векторні моделі демонстрували найкращі результати рівня state-of-the-art для більшості задач комп'ютерної лінгвістики.

Дослідження векторних представлень речень є важливим з багатьох причин. Дані представлення мають на меті вловити значення та семантичні нюанси речень. Їх удосконалення допомагає розуміти текст на глибшому рівні, дозволяючи ефективніше розуміти контекст та розв'язувати інші задачі.

Якісні векторні представлення слів змінили методи обробки та аналізу природної мови, адже слова є фундаментом мови. Речення є наступним етапом в даній ієрархії, тому високоякісні представлення зможуть вплинути на розвиток інших областей та задач, таких, як машинний переклад, класифікація тексту та пошук інформації.

Починаючи з моделей представлення семантики слів, дослідники намагалися закодувати у векторах слова інформацію про його найближчого оточення. Наприклад, k сусідніх слів зліва та k сусідніх слів справа, як в моделях CBOW та Skip-gram [4]. Це є занадто спрощеним уявленням, що не враховує розмаїття різних складних нелінійних зв'язків між словами всередині речення. Структура речення не є лінійною послідовністю слів, а має радше структуру дерев з елементами рекурсії. Ігнорування таких реалій мови може мати значні негативні наслідки, коли дослідники мають справу з аналітичними мовами (наприклад, з англійською мовою), де є чітко фіксований порядок слів у реченні. У випадку з синтетичними мовами, де порядок слів у реченні може вільно

змінюватися, неврахування справжньої синтаксичної структури може призвести до зниження ефективності моделі.

Застосування мовних моделей постійно розширюється, і тому зростає потреба в моделях, які можуть ефективно розуміти та генерувати текст у різних мовах і контекстах. Тому дослідження моделей представлення семантики речень природної є важливим для вдосконалення можливостей розуміння мови.

У даній роботі було проведено експерименти з англійською мовою, проте дані підходи та моделі можна застосовувати до всіх мов, у яких є чітко виражена структура речення (наприклад, аналітичні мови - германські мови, французька та інші).

Мета і завдання дослідження. Метою дисертаційного дослідження є створення методів побудови векторного представлення речення на основі різних моделей; дослідження використання графових структур репрезентації речення та їх застосування у моделях.

Для досягнення поставленої мети сформовано наступні завдання:

1) Провести дослідження наявних методів побудови векторного представлення та виконати експериментальну перевірку їх ефективності на практичних задачах.
2) Перевірити використання різних графових представлень речення для побудови представлення речення.
3) Створити модель для побудови представлення речення з урахуванням його структури.

Об’єктом дослідження є методи побудови векторних представлень речень природної мови.

Предметом дослідження є процес проєктування різноманітних моделей (машинного навчання, нейронних мереж різної архітектури) для побудови векторних представлень речень природної мови; їх застосування для розв’язання

прикладних задач, таких як виправлення граматичних помилок та ідентифікація парафраз.

Методи дослідження. У дисертаційній роботі застосовуються методи машинного навчання, багатошарові нейронні мережі (на основі архітектури Трансформер), теорія графів та контекстно-вільних граматик. Для програмної реалізації методів та відповідних моделей використовується мова програмування Python 3.10.

Наукова новизна отриманих результатів. Під час дослідження отримано наступні результати:

1) Модифіковано алгоритм Ерлі для роботи з граматиками природніх мов, використано підхід «повернення назад» для її роботи алгоритму. Показано можливість розширення граматики для виведення речень з граматичними помилками та механізми корекції помилок. Проаналізовано «синтаксичну близькість» між вхідним реченням та реченнями, які є допустимими у вхідній граматиці.
2) Створено модель з використанням дерева залежностей для репрезентації структури речення. Побудована множина ознак на його основі за допомогою агрегації властивостей на графі. Пропонований метод дозволяє розв’язувати задачу ідентифікації парафраз та доводить доцільність використання структури речення у моделях представлення речень.
3) Створено модель на основі архітектури Трансформер з використанням дерева залежностей. Досліджено її ефективність та порівняно з іншими моделями тієї ж архітектури.

Особистий внесок здобувача. Дисертаційна робота є результатом самостійних розробок автора. В роботах, виконаних у співавторстві, автору належать наступні результати: [5] – модифікація та дослідження застосування

алгоритму Ерлі для виправлення граматичних помилок у текстах; [6] – побудова моделі з використанням дерева залежностей та дослідження різноманітних ознак на його основі; [7] – проєктування моделі з використанням структури речення, її дослідження та експериментальна перевірка ефективності застосування для розв'язання задачі ідентифікації парафраз; [8] – огляд та аналіз різних підходів розв'язання задачі ідентифікації парафраз; [9] – огляд та аналіз існуючих моделей на основі нейронної мережі архітектури Transformer та LLM, дослідження їх використання для ідентифікації парафраз. Лістинг програмного коду доступний за посиланням [10].

Апробація матеріалів дисертації. Основні теоретичні та практичні результати дисертаційної роботи доповідались на науково-технічних конференціях та семінарах:

2019 IEEE International Conference on Advanced Trends in Information Theory (м. Київ, 18-20 грудня 2019 р.).

2020 IEEE 2nd International Conference on Advanced Trends in Information Theory (м. Київ, 25-27 листопада 2020 р.).

2023 International Conference "Information Technology and Implementation" IT&I. (м. Київ, 20-21 листопада 2023 р.).

Публікації. За темою дисертаційної роботи опубліковано 5 наукових праць (2 статті та 3 тези): 2 статті – у фахових наукових виданнях України; 1 стаття та 2 тези проіндексовано в базі даних Scopus.

Структура та обсяг дисертаційної роботи. Дисертаційна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел, додатків. Загальний обсяг роботи складає 144 сторінки, з яких основний зміст викладено на 115 сторінках. Дисертація містить 29 рисунків, 31 таблицю та список використаних джерел зі 93 найменувань.

РОЗДІЛ 1. Огляд існуючих моделей представлення семантики та структури речень та їх використання для прикладних задач

У сфері обробки природної мови (NLP) ключовою є задача розуміння людської мови. Для того, аби розв’язати цю задачу, було розроблено низку методів для побудови репрезентацій слів та речень, як ключових одиниць інформації в мові. Отже, задача розуміння певним чином зводиться до кодування текстової інформації (наприклад, в числові вектори), які будуть містити, або репрезентувати семантичне значення у більш зручній формі для подальшого аналізу та обробки.

У цій роботі ми фокусуємося саме на вивченні представлення семантики речень. Як фундамент у більшості моделей використовуються різні представлення слів. Саме тому ми б хотіли коротко переглянути деякі методи побудови представлення слів, а після цього перейти до моделей представлення семантики та структури речень. В даній роботі було використано задачу ідентифікації парафраз для оцінки якості моделей. Оглянуто також наявні підходи до розв’язання цієї задачі.

1.1. Представлення слів

Перед тим, як почати вирішувати будь – яку задачу обробки природної мови, потрібно вирішити, яким чином будуть представлені слова в майбутній моделі. Методи представлення слів еволюціонували протягом останніх 20-30 років від певного «атомарного» представлення у тезаурусах, де були різні зв’язки між словами, до векторного представлення, яке використовується в більшості сучасних моделей.

1.1.1. WordNet

WordNet – це лексична база даних англійської мови. Іменники, дієслова, прикметники та прислівники згруповані в набори когнітивних синонімів (синсети), кожен з яких виражає окреме поняття. Синсети пов'язані між собою за допомогою понятійно-семантичних і лексичних відношень [11].

Основним зв’язком між словами в WordNet є синонімія. Синоніми — слова, що позначають те саме поняття та взаємозамінні в багатьох контекстах — групуються в невпорядковані набори (синсети). Найбільш часто використовуваним відношенням серед синсетів є відношення підпорядкованості Воно пов’язує загальні синсети, наприклад {furniture, piece_of_furniture}, із конкретнішими, як-от {bed} чи {table}. Таким чином, WordNet стверджує, що категорія меблів включає ліжко та стіл.

Для того, аби визначити, наскільки два слова схожі, були розроблені різні алгоритми [12], які використовують WordNet. Багато з них аналізують шляхи між поняттями і, базуючись на них, визначають ступінь схожості.

Серед недоліків такого представлення слів є власне формат тезауруса. Із появою нових слів він має завжди оновлюватися. Побудова та оновлення такої структури потребує багато часу спеціалістів, зазвичай лінгвістів, які будуть розробляти його. Що своєю чергою робить WordNet досить дорогим там суб’єктивним. Розробка тезауруса для іншої мови теж вимагає чималих зусиль.

1.1.2. One-hot вектор

Одним з найпростіших способів закодувати слово у вигляді вектора є one-hot вектор. Для того, аби це зробити, потрібно визначити словник $V, |V| = d$.

Кожне слово даного словника у такому випадку можна закодувати унітарним кодом: вектор, всі елементи якого нулі, крім однієї одиниці.

Серед недоліків цього методу кодування є те, що вектори будуть мати розмірність словника – d і майже повністю заповнені 0. Сам словник має бути

зафіксований і, таким чином, аби додати нове слово, потрібно змінити вектори всіх слів у словнику.

Іншою проблемою є визначення схожості між словами. Для цього дуже часто використовують косинусну відстань:

$$similarity_{cos}(a, b) = cos\ (a, b) = \frac{a \cdot b}{\|a\| \cdot \|b\|} \qquad (1.1)$$

У випадку з one-hot векторами відстань завжди буде 0, адже вектори є ортогональними. Тому, використовуючи лише one-hot вектори, неможливо зрозуміти чи слова близькі за змістом, чи ні.

1.1.3. Матриця суміжності та SVD метод

Щоб вирішити проблему визначення близькості було запропоновано будувати вектори, які будуть схожими для слів близьких за змістом.

Для цього було запропоновано звернути увагу на контекст слова у реченні. Під контекстом у цьому випадку будемо розуміти слова, які стоять поряд із словом, для якого будується представлення. Для того, аби закодувати ці залежності побудуємо матрицю суміжності (co-occurrence matrix) слів X. Рядок цієї матриці може певною мірою закодовувати інформацію про слова використовуючи їх контекст. Для того, аби зменшити розмірність цього вектору, можна застосувати сингулярний розклад матриці [13] (SVD), вибрати k – розмірність вектора та використати частину розкладу метриці.

Розглянемо побудову матрицю суміжності на прикладі речень:

1. I like to eat.
2. I want to sleep.
3. I want to read a book.

Визначимо розмір контексту c, а саме скільки слів вправо чи вліво ми будемо використовувати для побудови матриці. Нехай у даному прикладі c = 2.

Матриця суміжності будується шляхом підрахунку, як часто слово x та y знаходяться поряд.

Таблиця 1.1. Матриця суміжності

	I	like	to	eat	want	sleep	read	a	book
I	0	1	3	0	0	0	0	0	0
like	1	0	1	1	0	0	0	0	0
to	3	1	0	1	2	1	1	1	0
eat	0	1	1	0	0	0	0	0	0
want	0	0	2	0	0	1	1	0	0
sleep	0	0	1	0	1	0	0	0	0
read	0	0	1	0	1	0	0	1	1
a	0	0	1	0	0	0	1	0	1
book	0	0	0	0	0	0	1	1	0

Після застосування SVD розкладу ми отримаємо 3 матриці:

$$X = USV^T \tag{1.2}$$

Після вибору k – бажаної розмірності вектора слова, перші k стовпчиків у матриці U будуть містити представлення слів.

$$\tilde{X} = \begin{bmatrix} u_{1,1} & \cdots & u_{1,k} \\ \vdots & \ddots & \vdots \\ u_{V,1} & \cdots & u_{V,k} \end{bmatrix} \begin{bmatrix} \sigma_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_k \end{bmatrix} \begin{bmatrix} v_{1,1} & \cdots & v_{1,V} \\ \vdots & \ddots & \vdots \\ v_{k,1} & \cdots & v_{k,V} \end{bmatrix}, \tag{1.3}$$

де матриця $\tilde{X}$ – апроксимація матриці X рангу k.

Серед недоліків цього підходу є те, що для того, аби додати нові слова потрібно змінити матрицю і ще раз знаходити її розклад. Утворена матриця є дуже розрідженою і має великий розмір – $V \times V$. Також деякі слова, такі як “the”, “a” та інші будуть зустрічатися у корпусі дуже часто і з багатьма іншими словами і створювати шум. Тому певні спеціальні модифікації матриці X потрібні, аби збалансувати їхню частоту.

1.1.4. Word2vec модель

Word2vec модель відрізняється від попереднього методи тим, що для того, аби побудувати вектор для слова, не потрібно будувати матрицю суміжності. Натомість дана модель намагається побудувати класифікатор, який відповідає на запитання: «Чи ймовірно, що слово a з'явиться біля слова b»?

Маючи даний класифікатор, ваги, що відповідають кожному слову, можуть бути використанні як представлення слів [13].

Word2vec містить в собі набір алгоритмів (Skip-gram, CBOW) і був описаний Томасом Міколовим та іншими у [4,14].

У Skip-gram класифікатор намагається максимізувати ймовірність того, що слово w і його контекст може зустрітися поряд, на відміну від якихось випадково вибраних слів. А CBOW навпаки має на меті передбачити слово використовуючи його контекст.

Дана модель побудови представлення слів змінила світ обробки природної мови і стала однією з фундаментальних для наступних моделей.

Однією з особливостей векторних представлень стала їх можливість вловлювати відносні зв'язки між словами.

Наприклад, результатом виразу $\vec{w}_{king} - \vec{w}_{man} + \vec{w}_{woman}$ є вектор дуже близький до вектора $\vec{w}_{queen}$.

Серед недоліків Word2vec є його фіксований словник і проблеми зі словами, що не входять у нього, або рідкісними термінами, яких немає в навчальному корпусі. Під час тренування word2vec використовує лише обмежений контекст слова, тому може не вловлювати ширші контекстуальні нюанси.

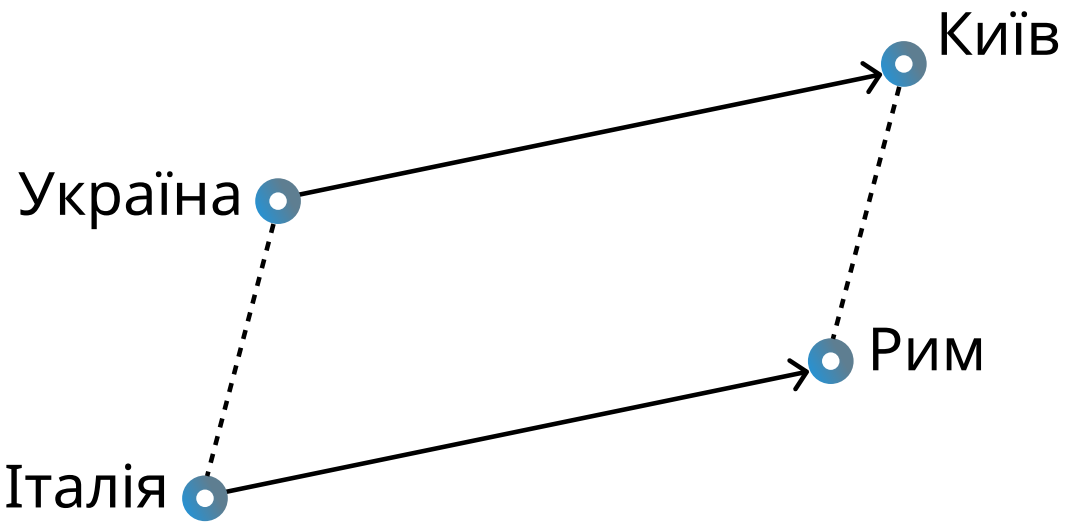


Рис. 1.1. Модель для задачі «аналогій» використовуючи векторні представлення слів

1.2. Представлення структури речення

Існує багато способів презентації формальної структури речення. Більшість з них намагається описати взаємозв'язки між різними словами чи словосполученнями у реченні. Розглянемо деякі з них:

1.2.1. Синтаксичне дерево

Контекстно-вільні граматики є основою багатьох моделей синтаксису природної мови. Синтаксичний аналіз – це процес аналізу граматичної структури речення відповідно до набору правил [15]. Дерева синтаксичного аналізу використовуються для перевірки граматики речення. Синтаксичне дерево є корисним інструментом аналізу речення [13].

Контекстно-вільна граматика складається з набору правил або продукцій, кожне з яких виражає способи, якими елементи мови можуть бути згруповані та впорядковані разом.

Нехай задана граматика з правилами:

$$\begin{aligned} S &\rightarrow NP\ VP \\ NP &\rightarrow Pronoun \mid Det\ Nominal \\ Nominal &\rightarrow Nominal\ Noun \mid Noun \\ VP &\rightarrow Verb\ NP \end{aligned} \tag{1.4}$$

тоді синтаксичне дерево для речення “I want a good party” буде мати вигляд:

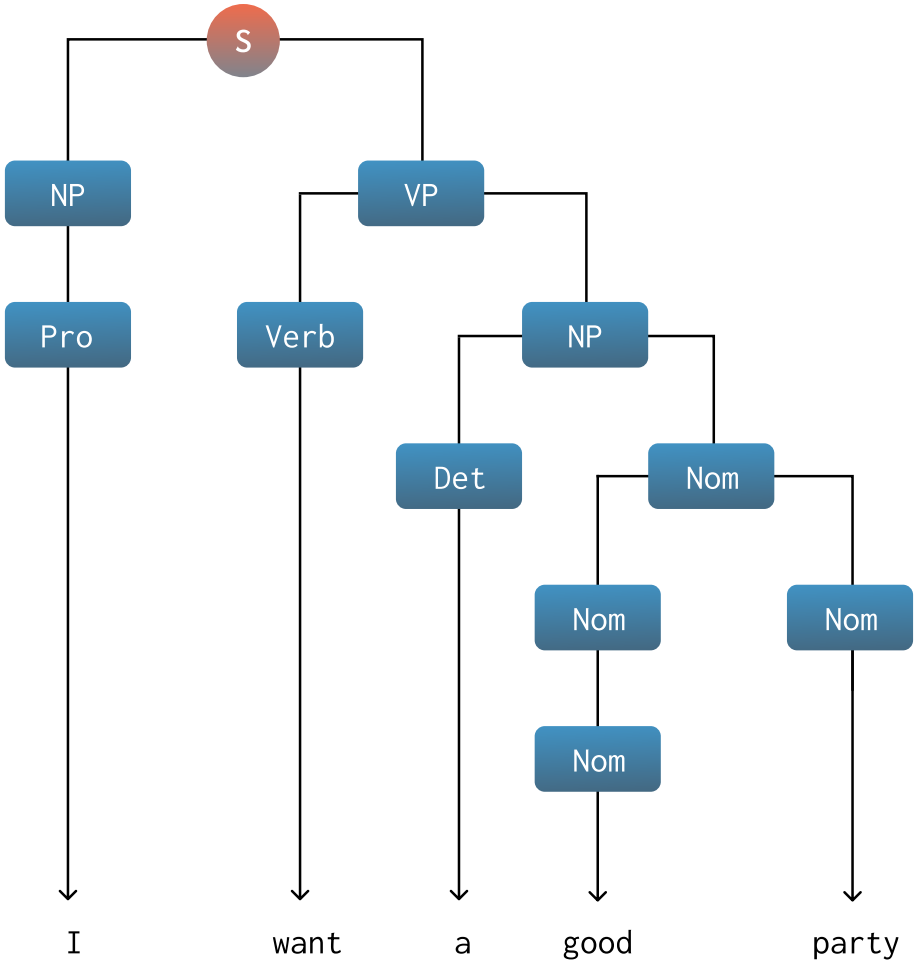


Рис. 1.2. Синтаксичне дерево розбору речення “I want a good party”.

Дане дерево аналізу можна також представити у більш компактному форматі, який називається дужкова нотація [13]:

$$[_S[_{NP}[_{Pro}\ I]]\ [_{VP}\ [_V\ want]\ [_{NP}\ [_{Det}\ a]\ [_{Nom}\ [_N\ good]\ [_{Nom}\ [_N\ party]]]]]]] \quad (1.5)$$

Більш глибоко синтаксичний аналіз та синтаксичні дерева буде проаналізовано у Розділі 2.

1.2.2. Дерево залежностей

Ще одним важливим представленням структури речення є дерево залежностей. У ньому структура речення описується виключно в термінах спрямованих бінарних граматичних зв'язків між словами.

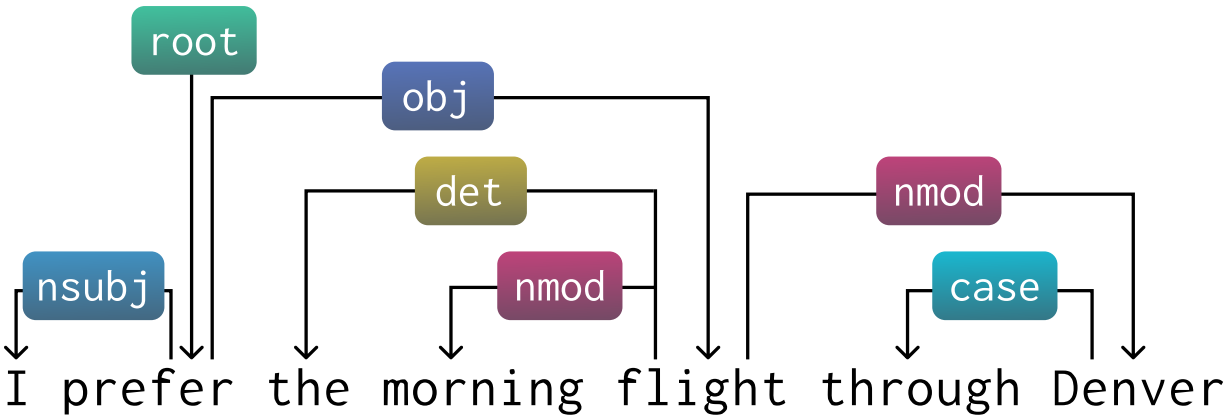


Рис. 1.3. Дерево залежностей речення "I prefer the morning flight through Denver"[13]

Дерево залежностей — це орієнтований граф $G = (V, A)$, який задовольняє наступні обмеження [13]:

1. Є один кореневий вузол, який не має вхідних дуг – $root$.
2. За винятком кореневого вузла кожна вершина має рівно одну вхідну дугу.
3. Існує унікальний шлях від кореневого вузла до кожної вершини в V.

Зв’язки між словами проілюстровані над реченням спрямованими дугами від головного елемента до залежного. Кожна дуга має свій тип і тому дане дерево є типізованим деревом залежностей. Вершина `root` явно позначає корінь дерева.

Проект Universal Dependencies [16] – є спробою анотувати залежності та інші аспекти граматики у понад 100 мов. Поточний перелік містить 37 зв’язків залежностей. Ось деякі з них:

Таблиця 1.2. Типи граматичних залежностей [13]

Тип зв’язку	Опис	Приклад (*Курсивом* виділено головний, **жирним** залежний елемент)
NSUBJ	Nominal subject	**United** *canceled* the flight.
OBJ	Direct object	United *diverted* the **flight** to Reno. We *booked* her the first **flight** to Miami
IOBJ	Indirect object	We *booked* **her** the flight to Miami.
NMOD	Nominal modifier	We took the **morning** *flight*.
AMOD	Adjectival modifier	Book the **cheapest** *flight*.
NUMMOD	Numeric modifier	Before the storm JetBlue canceled **1000** *flights*.
APPOS	Appositional modifier	*United*, a **unit** of UAL, matched the fares.
DET	Determiner	**The** *flight* was canceled. **Which** *flight* was delayed?
CASE	Prepositions, postpositions and other case markers	Book the flight **through** *Houston*
CONJ	Conjunct	We *flew* to Denver and **drove** to Steamboat.
CC	Coordinating conjunction	We flew to Denver **and** *drove* to Steamboat.

1.2.3. AMR граф

AMR (Abstract Meaning Representation) – це спосіб нотації семантичного представлення речення [17]. AMR – орієнтований граф, який має виділений корінь. Для двох речень, які мають однакове значення, AMR граф має бути однаковим, наприклад для (“The boy looked up the answer”, “The boy looked the answer up”).

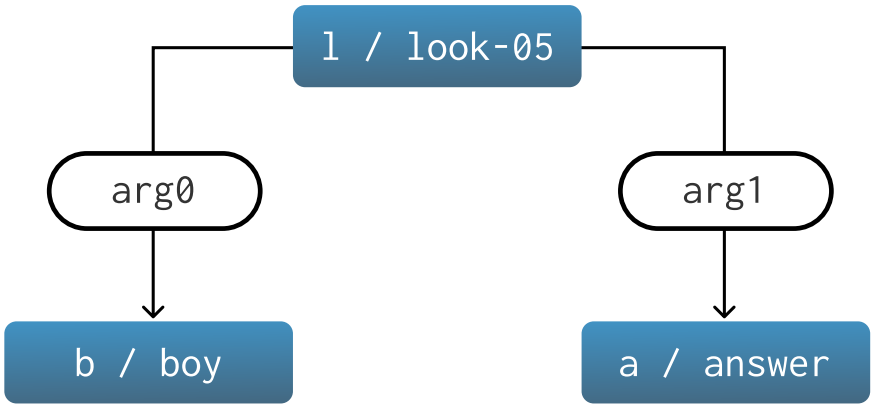


Рис. 1.4. AMR граф речення “The boy looked the answer up”

Вершинами цього графу є сутності, події, властивості та стани. Листки відповідають концептам, так, наприклад **“(b / boy)**” відноситься до екземпляра (під назвою **b**) концепту **boy**. Ребра графу з’єднують його сутності, наприклад **“(d / die-01 :location (p / park))”** означає, що сталася смерть **(d)** в парку **(p)**. AMR використовує приблизно 100 відношень.

AMR граф також можна описувати використовуючи PENMAN нотацію [18]. Речення “The boy wants to go” буде занотовано наступним чином.

$$\begin{aligned}(w\ /\ want-01\\ :arg0\ (b\ /\ boy)\\ :arg1\ (g\ /\ go-01\\ :arg0\ b))\end{aligned} \tag{1.6}$$

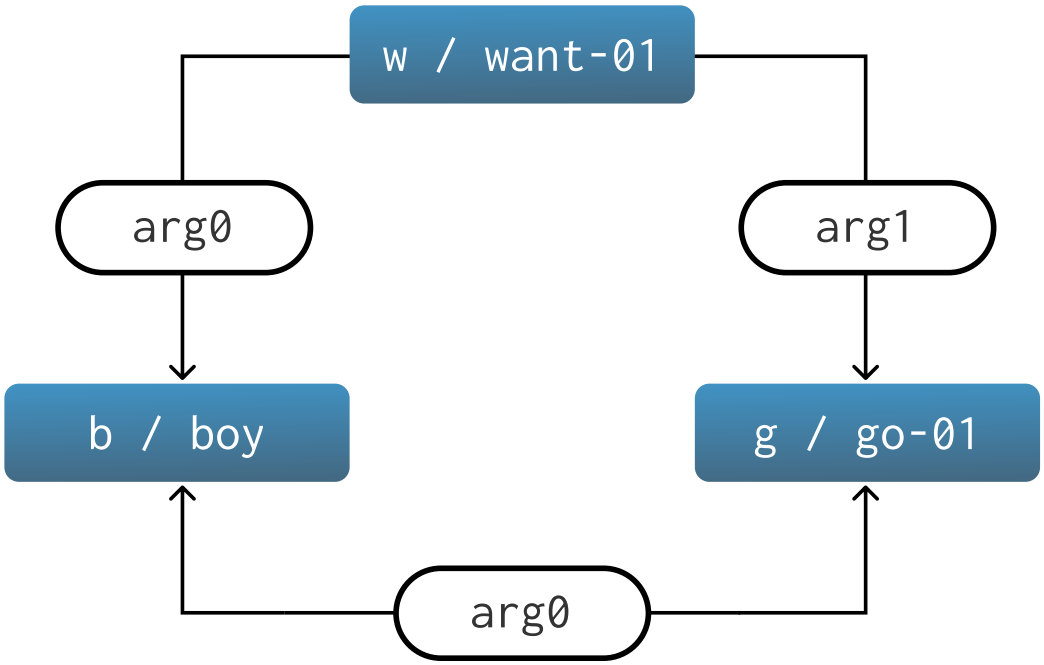


Рис. 1.5. AMR граф речення “The boy wants to go”[17].

1.3. Методи представлення речення

В даній секції оглянемо фундаметальні моделі представлення речень, починаючи з найпростіших – «мішка слів» (bag-of-words) до нейронних мереж.

1.3.1. Bag-of-Words

Однією з найочевидніших моделей презентації речення є Bag-of-Words (BoW). Маючи вхідне речення і вектори кожного зі слів – їх можна, наприклад, усереднити і отримати вектор речення [19].

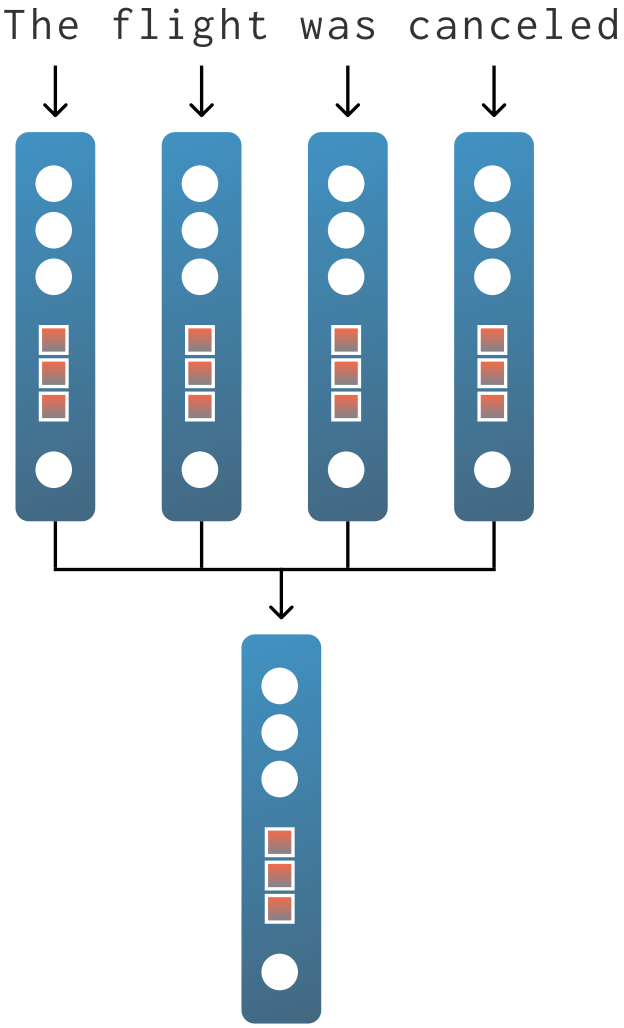


Рис. 1.6. Модель BoW для речення “The flight was canceled”

Це досить простий метод, який дає можливість дуже просто представляти речення, але він не враховує ні порядок слів у реченні, ні будь-яку структуру речення.

1.3.2. Рекурентні нейронні мережі

Рекурентна нейронна мережа (RNN) — це мережа, яка містить цикл у своїх з’єднаннях. Це означає, що значення деякої одиниці мережі прямо чи опосередковано залежить від її власних попередніх виходів [13].

Даний клас мереж виявився досить ефективним для розв'язання задач обробки мови, через їхню можливість обробляти інформацію послідовно, як, наприклад, слова у реченні.

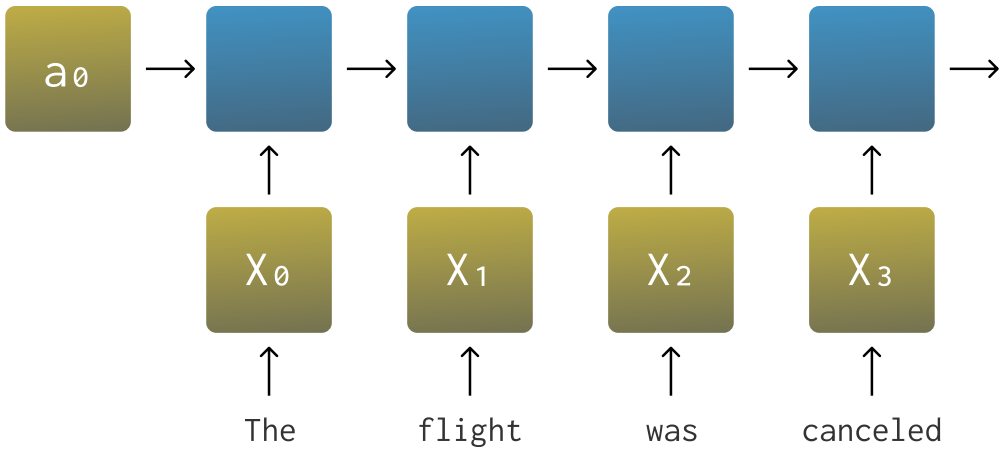


Рис. 1.7. Застосування RNN для речення "The flight was canceled" [20]

Таким чином, в результаті обробки всієї послідовності буде утворено представлення речення. Крім простої послідовності можна використовувати і інші структури, які базуються на графах, наприклад у [21].

Розглянемо блок RNN більш детальніше:

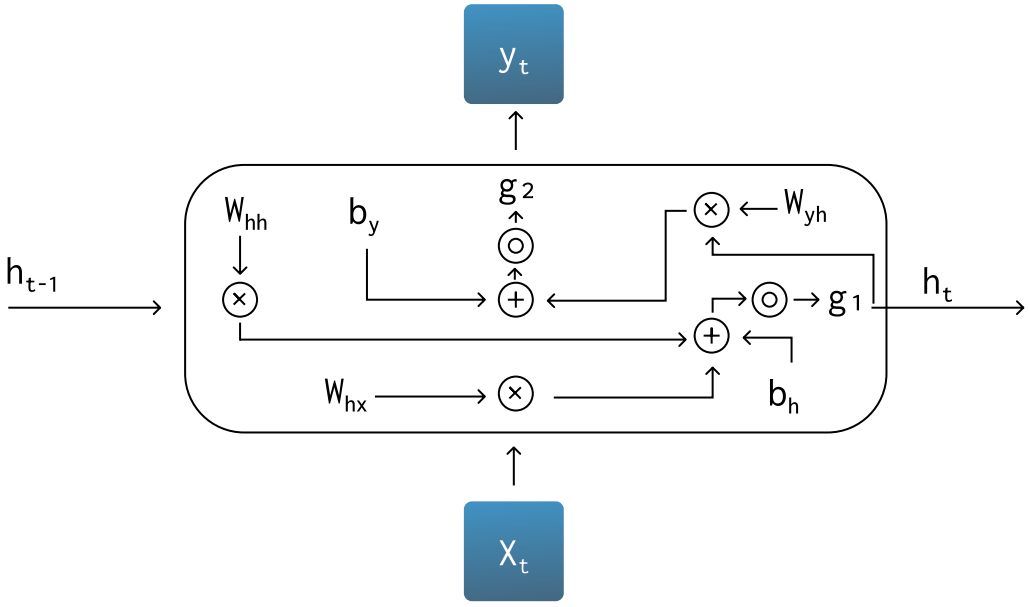


Рис. 1.8. Схема блоку RNN [20]

Для кожного часу t, вектор прихованого шару h_t та вихідний вектор y_t підраховуються наступним чином:

$$h_t = g_1(W_{hh}h_{t-1} + W_{hx}x_t + b_a), \quad (1.7)$$

$$y_t = g_2(W_{yh}h_t + b_y), \quad (1.8)$$

де $W_{aa}, W_{ax}, W_{ya}, b_a, b_y -$ коефіцієнти, які не залежать від часу t, а $g_1, g_2 -$ функції активації.

Дана мережа зручна тим, що розмір моделі не збільшується зі збільшенням вхідної послідовності, до уваги береться послідовність елементів. Але також саме це і обмежує модель, адже всі розрахунки повинні бути виконані теж послідовно. На практиці також було виявлено, що інформація про елементи на початку послідовності втрачається для довгих речень через проблеми зникаючих та зростаючих градієнтів (vanishing and exploading gradients), які були описані в [22]. Причина, чому це відбувається, полягає в тому, що важко вловити довгострокові залежності через мультиплікативний градієнт, який може експоненціально зменшуватися/збільшуватися щодо кількості шарів/вхідних даних. Аби вирішити цю проблему, були розробленні RNN зі спеціальними шлюзами (gates), кожен з яких має своє призначення, а також окремий стан, через який передається інформація між шарами. Дана модель RNN має назву LSTM (Long-short term memory).

Таблиця 1.3. Типи вентилів у RNN LSTM [20]

Тип шлюзу	Роль
Forget gate Γ_f	Даний шлюз використовується для того, аби забувати/видаляти інформацію з вектора стану.
Output gate Γ_o	Даний шлюз використовується для того, аби формувати наступний прихований шар.
Input gate Γ_i	Даний шлюз допомагає моделі визначити, яку інформацію з поточного стану передавати в майбутнє.

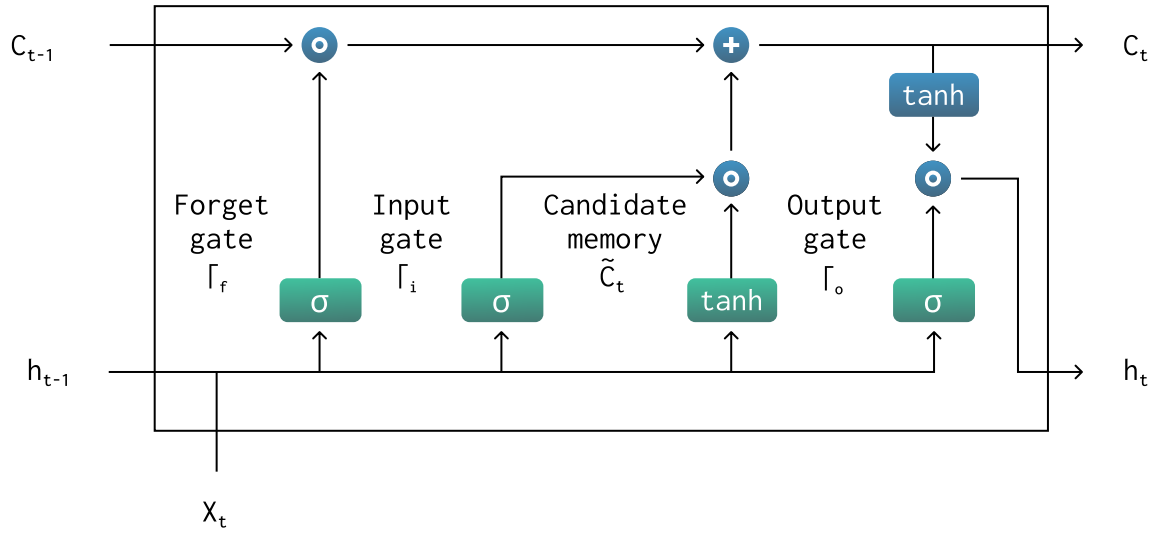


Рис. 1.9. Схема блоку LSTM [20]

Для кожного часу t, вектор прихованого шару h_t та вектор стану c_t підраховуються наступним чином [23]:

$$f_t = g(U_f h_{t-1} + W_f x_t + b_f), \quad (1.9)$$

$$i_t = g(U_i h_{t-1} + W_i x_t + b_i), \quad (1.10)$$

$$o_t = g(U_t h_{t-1} + W_o x_t + b_o), \quad (1.11)$$

$$\tilde{c}_t = c(U_c h_{t-1} + W_c x_t + b_c), \quad (1.12)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \quad (1.13)$$

$$h = o_t \odot h(c_t). \quad (1.14)$$

Серед недоліків LSTM є їх досить висока складність тренування, що своєю чергою може потребувати більшої кількості даних для навчання. LSTM – це підтип RNN, тому всі розрахунки будуть виконуватися послідовно, що створює обмеження для паралельного тренування.

1.3.3. Модель нейронної мережі Трансформер

Архітектура моделі Трансформер [24] була створена, аби розв'язати проблеми RNN описані вище. Більш детально дана архітектура буде розглянута у Розділі 4, але для швидкого ознайомлення звернемо увагу на ключові ідеї.

Дана архітектура базується на механізмі уваги, які дозволяють ефективно знаходити залежності між вхідною і вихідною послідовністю.

Трансформер складається з енкодера та декодера і був вперше описаний для розв'язання задачі перекладу. Для того, аби побудувати представлення речення, достатньо використати лише енкодер частину.

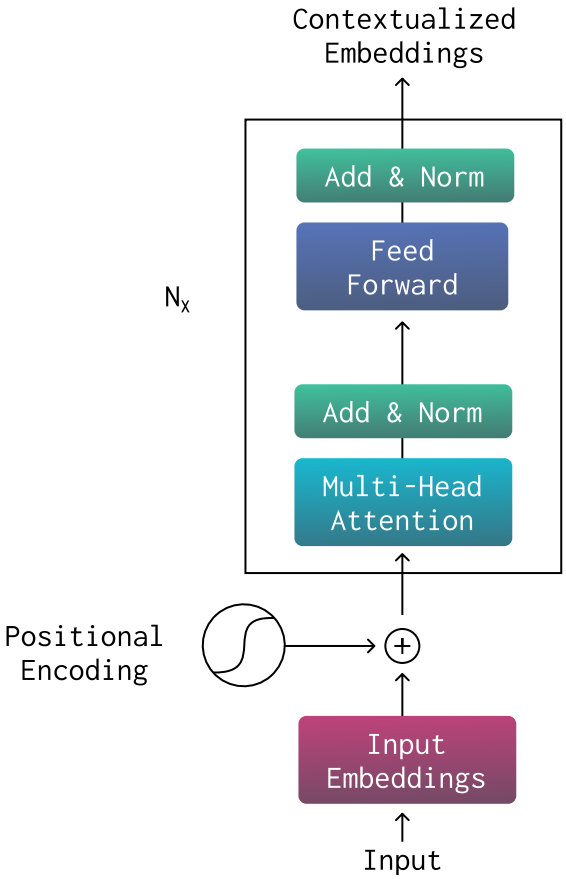


Рис. 1.10. Архітектура енкодера моделі Трансформер

Серед переваг Трансформера є можливість проводити тренування паралельно відносно елементів вхідної послідовності.

1.4. Оцінка якості векторних представлень речення

Оцінка векторних представлень речень не є простою задачею. Не існує «правильних» представлень, тому не можна зробити порівняння з певним ідеалом.

Тому для оцінки векторних представлень та самої мовної моделі дослідники покладаються на непрямі методи оцінки.

Ці методи, в основному, базуються на перевірці можливостей застосування утворених векторів для розв'язанні різних прикладних лінгвістичних задачах.

У даній роботі ми будемо використовувати задачу ідентифікації парафраз.

1.5. Задача ідентифікації парафраз

Ідентифікація парафраз — це завдання визначення того, чи два тексти мають однакове, чи схоже значення, навіть, якщо вони сформульовані по-різному.

Прикладом парафраз можуть слугувати такі речення:

Він розлютив мене, коли проявив свою невихованість за вечерею.

Його неввічливість, плітки та загальна відсутність поваги за вечерею розлютили мене.

У контексті обробки природної мови ідентифікація парафразу зазвичай розглядається як завдання бінарної класифікації, де вхідними даними є пара речень, а виходом є булеве значення, що вказує, чи є два речення парафразами, чи ні.

Ефективність моделей зазвичай оцінюється за допомогою стандартних показників, таких як точність, повнота, F1.

Загалом ідентифікація парафразу є складним завданням, яке вимагає глибокого розуміння семантики природної мови. Проте, за останні роки було досягнуто значного прогресу, і найсучасніші моделі можуть досягти високого рівня точності на різних контрольних наборах даних.

1.6. Корпуси даних для задачі ідентифікації парафраз

Існує досить багато різних корпусів, які зазвичай використовуються для задачі ідентифікації парафразу. Наведемо деякі з них:

1) Microsoft Research Paraphrase Corpus (MRPC) [25] – це корпус який містить пари речень, для яких вручну було зазначено чи являються вони парафразами. Він був створений Доланом і Брокеттом у Microsoft Research і став одним із найпоширеніших датасетів для даної задачі. Він містить 5801 пару речень, які взяті з різних джерел, включаючи статті новин, статті в енциклопедіях і вебсторінки. Щоб забезпечити якість міток, для кожної пари речень використовували кілька анотаторів, а розбіжності вирішувалися більшістю голосів.
2) PPDB [26] – це масштабна база даних, яка містить понад 220 мільйонів пар парафраз. Вона містить багато фразових та лексичних парафраз.
3) SICK (Sentences Involving Compositional Knowledge) [27] – це корпус, який використовуються для задач семантичної близькості. Він містить 10,000 пар речень, що охоплюють широкий спектр тем, включаючи політику, науку та розваги. Пари були підібрані таким чином, щоб вони мали різний ступінь семантичної спорідненості, починаючи від дуже схожих до зовсім не пов’язаних. Семантичний зв’язок між парами речень оцінюється за безперервною шкалою від 1 (зовсім не пов’язані) до 5 (семантично еквівалентні).
4) Пари запитань Quora [28] – це набір даних пар запитань із вебсайту Quora. Мета набору даних – визначити, чи є два запитання семантично еквівалентними, чи ні. Питання охоплюють широкий спектр тем, зокрема науку, технології, політику та розваги. Він містить понад 400 000 пар запитань і є одним із найбільших корпусів, доступних для ідентифікації парафраз.
5) ParaNMT-50M [29]: це великий корпус паралельних речень із різних джерел. Він містить понад 50 мільйонів пар речень і є одним із

найбільших доступних наборів даних. Пари речень у ParaNMT-50M узяті з різних джерел, у тому числі з новинних статей, вебсторінок і державних документів. Цей корпус був згенерований автоматично за допомогою перекладу частини декількох чесько-англійських корпусів.

Вище наведено лише кілька найчастіше використовуваних корпусів, які застосовують для розв'язання даної задачі. Вибір набору даних часто залежить від конкретного завдання та контексту, у якому використовуватиметься модель, а також від обчислювальних потужностей, які є в дослідників.

Таблиця 1.4. Корпуси даних

Назва корпусу	Рік створення	Джерела данних	Кількість пар речень
Microsoft Research Paraphrase Corpus	2005	Статті новин, енциклопедій і веб-сторінки	5,801
PPDB	2013	Статті новин, веб-сторінки і державні документи	220,000,000
Sentences Involving Compositional Knowledge	2014	Статті про політику, науку та розваги.	10,000
Пари запитань Quora	2017	Пари запитань з веб-ресурсу	404,289
ParaNMT-50M	2017	Статті новин, веб-сторінки і державні документи	51,409,585

1.7. Огляд підходів до розв'язання задач ідентифікації парафраз

Мета цього огляду – не стільки намагання охопити всі роботи, присвячені ідентифікації парафраз, скільки продемонструвати основні підходи та методи, які дослідники використовували в своїх розробках.

Методи виявлення парафраз сильно вдосконалилися завдяки прогресу в обробці природної мови і машинному навчанні. Розглядаючи цю еволюцію, можна виділити такі етапи або класи методів.

- Підходи, засновані на ручних правилах: на початку досліджень виявлення парафраз багато систем базувалися на створених вручну правилах та евристиках. Ці методи спиралися на синтаксичні та семантичні шаблони для ідентифікації парафраз і часто були обмежені їхньою залежністю від конкретних мовних особливостей.
- Використання лексичної подібності: дослідники почали використовувати лексичну та семантичну подібність використовуючи тезауруси, такі як WordNet [30] та латентний семантичний аналіз (LSA)[31], для ідентифікації парафраз. Ці методи базувалися на ідеї, що слова зі схожими значеннями, як правило, зустрічаються в подібних контекстах.
- Машинне навчання: оскільки методи машинного навчання стали більш популярними в обробці природної мови, дослідники почали розробляти контрольовані моделі навчання для виявлення парафраз. Ці моделі зазвичай використовують розмічені дані, щоб вивчати шаблони та особливості, які вказують на парафрази.
- Глибоке навчання: в останнє десятиліття глибоке навчання стало домінуючим підходом у машинному навчанні і застосовувалося для широкого кола завдань, включаючи виявлення парафраз. Моделі глибокого навчання, такі як нейронні мережі та Трансформери, показали високу продуктивність у різних тестах.

Для того, аби краще зрозуміти різні ідеї та алгоритми, розгляньмо декілька підходів з кожного класу.

Досить складно знайти чи виділити підходи, які будуть використовувати лише один з вище наведених класів, тому ми при виборі алгоритмів класифікували їх за основною ідеєю.

1.7.1. Методи, побудовані на ручних правилах

Одним з прикладів підходу, який базується на ручних правилах слів є система iSTART [32].

Вона використовує різноманітні лексичні, синтаксичні та семантичні особливості для ідентифікації парафраз.

Для того, щоб визначити структуру парафразу, розглядається декілька підходів їх побудови:

1) Синоніми: заміна слів їх синонімами.
2) Тип речення: зміна типу речення з активного на пасивний або навпаки.
3) Словоформа/частина мови: зміна форми слова в іншу, наприклад, змінити іменник на дієслово, прислівник або прикметник.
4) Розбиття речення: довге речення розбивається на маленькі.
5) Визначення: замінити слово на його визначення.
6) Структура речення: зміна структури речення.

В основі алгоритму є поняття графів концептів [33]. Для кожного речення будується свій граф і після цього відбувається порівняння графів двох кандидатів.

Процес порівняння полягає в тому, щоб знайти якомога більше збігів між трійками «поняття-відношення-поняття».

Речення перетворюється на граф концептів за допомогою аналізатора Link Grammar [34].

Даний алгоритм було реалізовано і частина з описаних вище структур була використана для визначення парафраз, проте автори використовували власний корпус даних для аналізу і не опублікували своїх результатів.

1.7.2. Методи, побудовані на лексичній подібності

Прикладом методу, де поєднана лексична та семантична подібність, є система розроблена Раду Міхалчем та Андраш Чомай [35], яка була опубліковано у 2006 році.

У даному підході скомбіновано два різних підходи до вимірювання семантичної подібності між двома частинами тексту: вимірювання на основі корпусу та вимірювання на основі знань. Властивості, засновані на корпусі, покладаються на статистичний аналіз великих текстових корпусів, тоді як властивості, засновані на знаннях, використовують зовнішні ресурси, такі як лексичні бази даних і онтології.

Автори презентують дві характеристики семантичної подібності на основі корпусу, засновані на дистрибутивній подібності та латентному семантичному аналізі.

Однією з них є характеристика, $PMI - IR$ яка базується на семантичній подібності слів у великому корпусі, була запропонована Терні [36].

$$PMI - IR(w_1, w_2) = log_2 \frac{p(w_1 \& w_2)}{p(w_1) * p(w_2)} \quad (1.15)$$

Для того, аби слова вважалися близькими, у корпусі було використано вікно довжиною десять слів, як баланс між точністю та продуктивністю.

Також вони вводять шість характеристик, засновані на базі знань WordNet [30]. Наведемо декілька з них:

- Подібність за Лікокам і Ходоровом [37] базується на нормалізованій довжині шляху з урахуванням глибини загальної ієрархії

  $$LH(w_1, w_2) = log_2 \frac{len(w_1, w_2)}{2D}, \quad (1.16)$$

  де D – максимальна глибина ієрархії в базі знань.
- Подібність за Ву і Палмером [38] враховує глибину найменшого спільного предка концептів в ієрархії:

$$WP(w_1, w_2) = -\log_2 \frac{depth(LSO(w_1,w_2))}{depth(w_1)+ depth(w_2)}, \quad (1.17)$$

де LSO – найменший спільний предок концептів пари слів.

Результати експериментів показують, що властивості як на основі корпусу, так і на основі знань можуть бути ефективними для ідентифікації парафраз.

Для оцінки системи автори використовують Microsoft Research Paraphrase Corpus. Результати показують, що даний метод досягає точності 70.3% та F1 81.3%.

1.7.3. Методи, побудовані на машинному навчанні

Варто зазначити, що більшість підходів так чи інакше використовують різний інструментарій машинного навчання, проте хотілося б розглянути методи, де саме це є в основі алгоритму.

У своїй статті Нітін Маднані та Джоель Тетро [39] пропонують новий підхід до оцінки моделей ідентифікації парафраз шляхом перепрофілювання існуючих метрик машинного перекладу. Автори стверджують, що існуючі метрики машинного перекладу, такі як BLEU, METEOR і TER, добре підходять для побудови моделей ідентифікації парафраз.

Розглянемо декілька метрик, які використовуються в даному алгоритмі:

- BLEU [40] – одна з найбільш часто використовуваних метрик для оцінки машинного перекладу. Вона базується на кількості однакових n-грам різних довжин у двох реченнях.

$$BLEU(s_1, s_2) = BP(s_1, s_2) exp\left[\sum_{n=1}^{N} \frac{1}{N} \log(p_n)\right], \quad (1.18)$$

$$BP(s_1, s_2) = exp\left[min\left[1 - \frac{|s_1|}{|s_2|}\right], 1\right], \quad (1.19)$$

$$p_n = \frac{\sum_{x \in NGram(s_1,n)} count(x, NGram(s_1, n) \cap NGram(s_2, n))}{\sum_{x \in NGram(s_1,n)} count(x, NGram(s_1, n))}, \quad (1.20)$$

$$count(x, S) = |\{el \mid el \in S \,\&\, el = x\}|, \quad (1.21)$$

де N – максимальна довжина n-грами, BP – коефіцієнт стислості, призначений для штрафування речень, якщо одне коротше за інше.

- NIST [41] є варіантом BLEU, яка використовує середнє арифметичне кількості однакових n-грам, а не середнє геометричне. Дана метрика також зважує кожен n-грам відповідно до його інформативності, використовуючи її частоту.
- TER [42] визначається як кількість редагувань, необхідних для того, аби з одного речення зробити інше використовуючи операції вставки, видалення та заміни та зсуву.

У якості класифікатора використовувався ансамбль декількох класифікаторів. На верхньому рівні був простий мета класифікатор, який використовував результати класифікації декількох інших класифікаторів нижнього рівня: логістичну регресію, метод опорних векторів та класифікатор на основі алгоритму найближчого сусіда [43].

Автори оцінюють систему, використовуючи Microsoft Research Paraphrase Corpus. Результати показують, що даний метод досягає точності 77.4% та F1 84.1%.

1.7.4. Методи, побудовані на глибокому навчанні

Більшість сучасних підходів використовують переваги наявності великих розмічених корпусів даних та нейронні мережі і Трансформери в якості алгоритмічного фундаменту.

Одним з яскравих представників таких алгоритмів є система Sentence – BERT [44].

У цій статті пропонується новий метод створення високоякісних представлень речень за допомогою сіамської мережі та BERT.

Представлення речення – це вектор фіксованої довжини, який використовується для того, аби описати семантичне значення речення у багатовимірному векторному просторі. Вони застосовуються в різних завданнях обробки природної мови, включаючи ідентифікацію парафраз, класифікацію тексту та пошук інформації.

Автори починають із зауваження, що існуючі методи генерації вбудованих речень зазвичай використовують неконтрольовані підходи, такі як усереднення представлення слів та інші. Однак ці методи часто призводять до низькоякісних представлень через природну складність захоплення семантичного значення речення з простого набору слів.

Мережа приймає як вхідні дані два речення та виводить оцінку схожості між ними, яку можна використовувати для створення високоякісних векторів, що будуть описувати речення.

Автори спочатку навчають сіамську мережу BERT на великому наборі даних пар речень, позначених бінарними мітками, використовуючи функцію втрат. Функція втрати мінімізує відстань між парафразами та максимізує відстань між не-парафразами у просторі.

Після навчання автори використовують мережу BERT для створення представлення речень для різноманітних завдань, включаючи ідентифікацію парафраз. Результати показують, що модель Sentence – BERT перевершує існуючі найсучасніші методи на більшості контрольних наборів даних на момент публікації.

1.8. Висновки до розділу 1

В результаті проведеного дослідження наявних моделей представлення семантики речень було розглянуто різноманітні структури та моделі представлення речення. В результаті було отримано наступні результати:

1. Проведено короткий огляд та класифікацію методів побудови представлення слів.

2. Проведено огляд та класифікацію методів представлення структури речень.
3. Проведено огляд та класифікацію методів побудови векторних представлень речень.
4. Вивчено методи оцінки якості векторних представлень речень та було обрано задачу ідентифікації парафраз для подальшого аналізу та порівняння моделей.
5. Проведено огляд корпусів даних для задачі ідентифікації парафраз.
6. Проведено огляд та класифікацію методів розв’язання задачі ідентифікації парафраз.

РОЗДІЛ 2. Синтаксична компонента в моделях представлення семантики речення

Обробка природної мови – це міждисциплінарна галузь лінгвістики та інформатики, що породила суміжну область під назвою «Обчислювальна лінгвістика». Вона фокусується на обробці природних мов на обчислювальних пристроях. Природна мова складається з речень, саме вони є значущими лінгвістичними одиницями, що включають одне або кілька слів, пов'язаних одне з одним.

Сучасні дослідження у галузі комп'ютерної лінгвістики звертають увагу на моделюванні людських мов. Обчислювальна лінгвістика є міждисциплінарною галуззю, що поєднує інформатику та лінгвістику, співпрацюючи з областю штучного інтелекту й вивчаючи обчислювальні аспекти людської мови [45]. Люди обмінюються знаннями, ідеями, думками та інформацією, використовуючи природну мову. Тому головним завданням комп'ютерної лінгвістики є розуміння цієї мови.

Кожна природна мова складається з нескінченної кількості речень, що відповідають певній базовій структурі. Структура речення може сприйматися ієрархічно на різних рівнях абстракції, таких як слово, словосполучення. Утворення речення залежить від синтаксично допустимих структур, які описані у граматиці мови. Основні структури речень значною мірою залежать від позицій підмета, присудка, додатка та їх взаємозв'язку.

Синтаксичний аналіз базується на з’ясуванні структури об’єктів. Кожен об’єкт формується з простих атомарних елементів, що задають алфавіт формальної мови [46]. У мові такими атомарними елементами можна вважати слова. Дані прості елементи можуть утворювати фрази або речення мови, що і є об’єктом даного дослідження.

Мета даного розділу – дослідити використання синтаксичного аналізу для побудови семантичного представлення. У пошуках способу оцінки та порівняння

двох дерев розбору, було побудовано метрику відстані. Для того, аби перевірити її ефективність, було обрано задачу виправлення граматичних помилок.

2.1. Граматика мови

Розглянемо деякі поняття та визначення, якими будемо далі оперувати в роботі [46,47].

Визначення 2.1. Граматика – це четвірка:

$G =< N, \Sigma, P, S >,$ де:

N – скінченна множина – допоміжний алфавіт (нетермінали);

Σ – скінченна множина – основний алфавіт (термінали);

P – скінченна множина правил виду:

$$\alpha \rightarrow \beta, \alpha \in (N \cup \Sigma)^* * N * (N \cup \Sigma)^*, \beta \in (N \cup \Sigma)^*$$

S – виділений нетермінал (аксіома) з N, називається початковим символом.

В залежності від структури правил граматики діляться на чотири типи (класифікація граматик по Хомському):

- ***Тип 0:*** граматики загального виду, правила яких не мають обмежень, тобто:

$$\alpha \rightarrow \beta, \alpha \in (N \cup \Sigma)^* * N * (N \cup \Sigma)^*, \beta \in (N \cup \Sigma)^* \quad (2.1)$$

- ***Тип 1:*** граматики, що не укорочуються, коли обмеження на правила мінімальні, а саме:

$$\alpha \rightarrow \beta, \alpha \in (N \cup \Sigma)^* * N * (N \cup \Sigma)^*, \beta \in (N \cup \Sigma)^*, |\alpha| \leq |\beta| \quad (2.2)$$

- ***Тип 2:*** контекстно-вільні граматики, коли правила в схемі Р мають вигляд:

$$\alpha \rightarrow \beta, \alpha \in N, \beta \in (N \cup \Sigma)^* \quad (2.3)$$

- ***Тип 3:*** скінчено-автоматні граматики, коли правила в схемі Р мають вигляд:

$$\begin{gathered} A_i \rightarrow wA_j, \qquad A_j, A_i \in N, w \in \Sigma^* \\ A_i \rightarrow w, \quad A_i \in N, w \in \Sigma^* \end{gathered} \quad (2.4)$$

$$A_i \rightarrow A_j w, \qquad A_j, A_i \in N, w \in \Sigma^*$$

Визначення 2.2. Ланцюжок w_1 безпосередньо виводиться з ланцюжка w (позначається $w \Rightarrow w_1$), якщо $w = x\alpha y,\ w_1 = x\beta y$ та в схемі Р граматики G є правило виду $\alpha \rightarrow \beta$. Оскільки поняття «безпосередньо виводиться» розглядається на парах ланцюжків, то надалі символ $\Rightarrow$ буде трактуватися як бінарне відношення.

Визначення 2.3. Ланцюжок w_1 виводиться з ланцюжка w (позначається $w \Rightarrow^* w_1$), якщо існує скінчена послідовність виду $w \Rightarrow w^1, w^1 \Rightarrow w^2, \dots w^{n-1} \Rightarrow w_1$. Або кажуть, що бінарне відношення $\Rightarrow^*$ - це рефлексивно-транзитивне замикання бінарного відношення $\Rightarrow$.

Визначення 2.4. Граматика G називається неоднозначною, якщо існує декілька варіантів виводу w в G ($w \in L(G)$).

Приклад 2.1. Розглянемо таку граматику $G(N, \Sigma, P, S)$ зі схемою P

$$S \rightarrow S - S \mid S * S \mid t \qquad (2.5)$$

Покажемо, що для ланцюжка $w = t - t - t$ існує щонайменше два варіанти виводу:

1. $S \Rightarrow S - S \Rightarrow t - S \Rightarrow t - S - S \Rightarrow t - t - S \Rightarrow t - t - t$
2. $S \Rightarrow S - S \Rightarrow S - S - S \Rightarrow S - S - t \Rightarrow S - t - t \Rightarrow t - t - t$

Визначення 2.5. Мова, породжена граматикою G, що позначається $L(G)$, являє собою набір термінальних ланцюжків, породжених G. Таким чином,

$$L(G) = \{w \mid w \in \Sigma^*, S \Rightarrow^* w\} \qquad (2.6)$$

Набір мов, які можуть бути згенеровані з контекстно-вільних граматик, називається контекстно-вільними мовами.

2.2. Синтаксичний аналіз та синтаксичне дерево

Синтаксичний аналіз – це процес аналізу граматичної структури речення. Метою є аналіз структури речення відносно набору граматичних правил [15].

Щоб виконати синтаксичний аналіз, потрібно мати граматику і послідовність, яка аналізується. Маючи такі вхідні дані, можна провести

граматичний розбір, аби з'ясувати, чи може дана послідовність бути породженою граматикою.

Граматики, що описують природні мови, зазвичай належать до контекстно-вільних граматик. Для їхнього аналізу традиційно застосовують дерева граматичного або синтаксичного розбору, або дерева виведення [46].

Визначення 2.6. Впорядковане синтаксичне дерево розбору фрази $F = x_1 x_2 \dots x_n$ на основі контекстно-вільної граматики $G(N, \Sigma, P, S)$ повинно задовольняти такі вимоги [46].

1) Кожна вершина має за мітку деякий символ з алфавіту
$$V = \Sigma \cup N \cup \{\varepsilon\}. \quad (2.7)$$
2) Коренем дерева є початковий символ S.
3) Листки дерева, якщо читати зліва направо утворюють фразу F (порядок листків має значення).
4) Будь-яка нелистова вершина має за мітку нетермінальний символ, листова – термінальний.
5) Якщо вершина A має синів $B_1 B_2 \dots B_n$ (враховуючи порядок зліва направо), то множині P повинно належати правило $A \to B_1 B_2 \dots B_n$

Побудова синтаксичного дерева виведення F в G виконується покроково з урахуванням стратегії виводу.

Візьмемо для прикладу речення:

Mariia likes apples. (2.8)

та базову граматику з такими правилами виводу:

$$\begin{gathered} S \to Noun\ Phrase + Verb\ Phrase \\ Noun\ Phrase \to Noun \\ Verb\ Phrase \to Verb + Noun\ Phrase \\ Noun \to Mariia \\ Noun \to apples \\ Verb \to likes \end{gathered} \quad (2.9)$$

Дане речення є вивідним у цій граматиці і має дерево граматичного розбору.

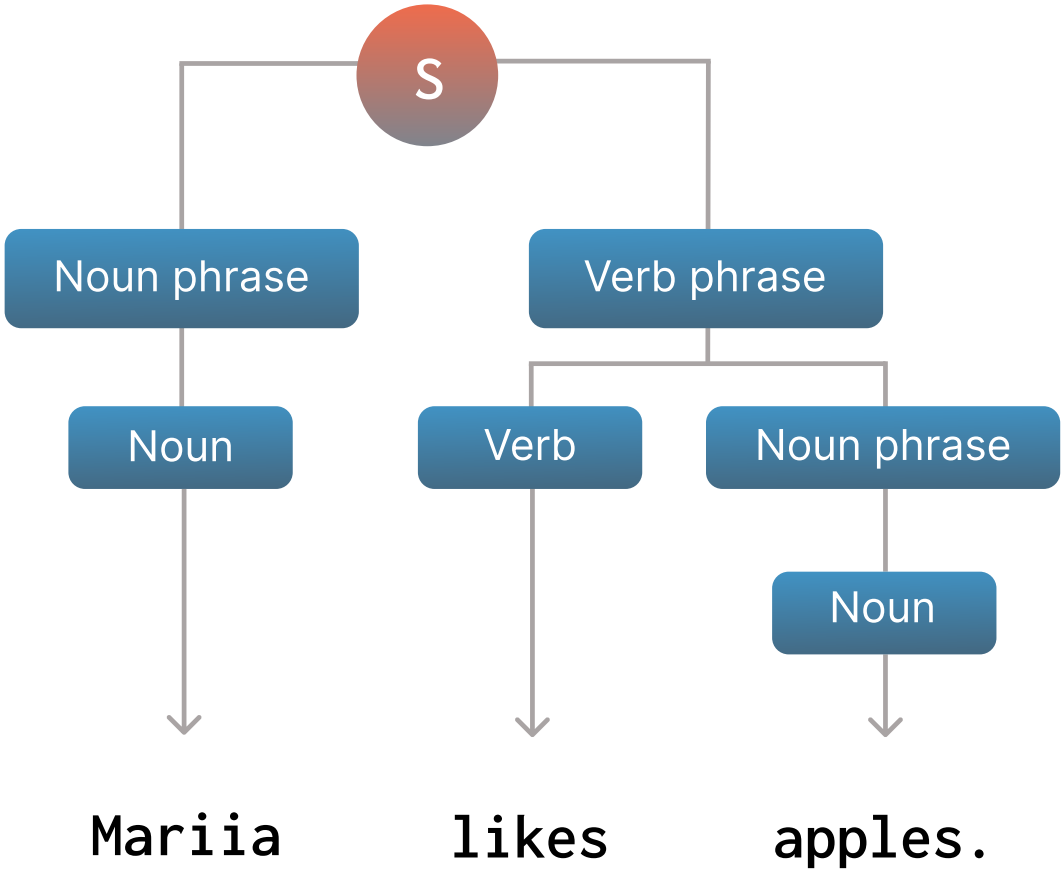


Рис. 2.1. Синтаксичне дерево розбору речення “*Mariia likes apples*”.

Існують дві основні стратегії граматичного розбору: згори донизу і знизу нагору.

Аналіз згори донизу починається з кореня дерева – *S*, до якого послідовно застосовуються правила підстановки, поки не буде отримана фраза, яка аналізується.

Аналіз знизу нагору починається із фрази, що розбирається. Знайдені підланцюжки замінюються на ліві частини відповідних правил підстановки, поки не буде отриманий початковий символ.

2.3. Алгоритм Ерлі

Алгоритм Ерлі [48] – це загальний алгоритм парсингу, здатний аналізувати контекстно-вільну граматику. Загальні алгоритми синтаксичного аналізу, такі як Ерлі, дозволяють безперешкодно аналізувати неоднозначні граматичні конструкції, що використовуються, наприклад в C, C++ [49] та обробці природної мови.

Парсер Ерлі працює, створюючи послідовність множин. Ці множини іноді називають множинами Ерлі. З огляду на вхід $x_1, x_2 \dots x_n$ синтаксичний аналізатор будує $n + 1$ множини: початкова множина S_0 та по одній множині S_i для кожного вхідного символу x_i. Елементи цих множин називаються елементами, що складаються з трьох частин:

1) Правила виводу в граматиці.
2) Позиції в правій частині правила, що вказує, яка частина цього правила була помічена.
3) Вказівника на попередню множину Ерлі.

Зазвичай елементи Ерлі записуються у такому вигляді:

$$[A \rightarrow \alpha \bullet \beta, j], \tag{2.10}$$

де позиція в правій частині правила позначається крапкою (•), а j – вказівник на множину S_j.

Маючи множину S_i, множина S_{i+1} ініціалізується шляхом застосування наступних трьох кроків до елементів в S_i доки S_{i+1}не стабілізується (не можна буде додати до S_{i+1}нових елементів).

СКАНЕР. Якщо $[A \rightarrow \dots \bullet a \dots, j] \in S_i$ і $a = x_{i+1}$, додайте $[A \rightarrow \dots\ a \bullet \dots, j]$ до S_{i+1}.

ПРЕДИКТОР. Якщо $[A \rightarrow \ldots \bullet B \ldots, j] \in S_i$, додайте $[B \rightarrow \bullet \alpha, i]$ до S_i для всіх правил $B \rightarrow \alpha$.

ЗАВЕРШУВАЧ. Якщо $[A \rightarrow \ldots \bullet, j] \in S_i$, додайте $[B \rightarrow \ldots A \bullet \ldots, k]$ до S_i для всіх елементів виду правил $[B \rightarrow \ldots \bullet A \ldots, k] \in S_j$.

Елемент буде додано до множини тільки в тому випадку, якщо його ще немає в множині (всі елементи у множинах унікальні). Початкова множина містить елемент $[S' \rightarrow \bullet S, 0]$ – ми припускаємо, що граматику доповнено новим початковим правилом $S' \rightarrow S$ – і остаточна множина повинна містити $[S' \rightarrow S \bullet, 0]$ щоб речення вважалося коректно виведеним. На рисунку 2.1 показаний приклад роботи аналізатора Ерлі.

З точки зору реалізації множини Ерлі будуються в порядку зростання у міру читання елементів введення. Крім того, кожна множина зазвичай представлена у вигляді списку елементів, як це було запропоновано Ерлі [48].

Таке представлення множини у вигляді списку дуже зручне, оскільки список елементів діє як «робоча черга» при побудові набору: елементи перевіряються послідовно, застосовуючи СКАНЕР, ПРЕДИКТОР, ЗАВЕРШУВАЧ за потреби; елементи, додані в набір, додаються в кінець списку.

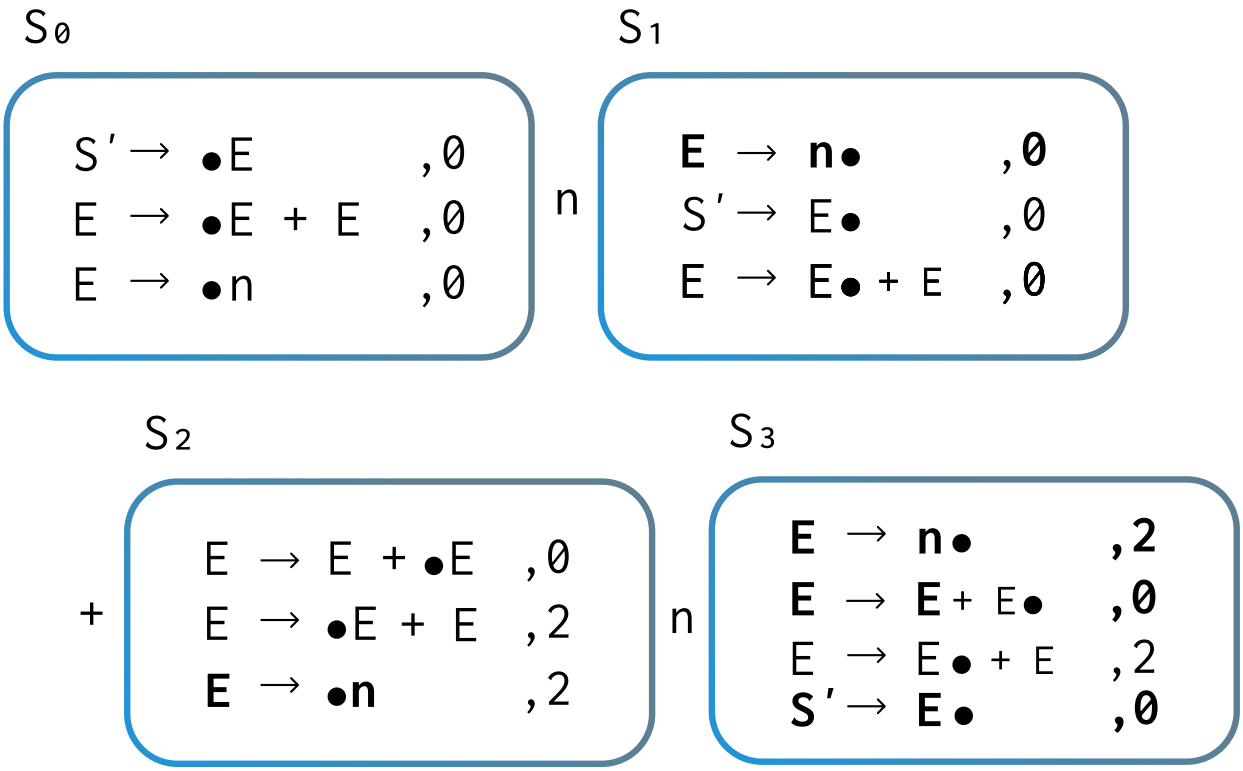


Рис. 2.2. Множини Ерлі для граматики $E \rightarrow E + E \mid n$ і входу $n + n$

2.3.1. Обробка ε – правил

При обробці граматик парсери мають деякі складнощі з ε – правилами, тобто правила виду

$$A \rightarrow \varepsilon, \tag{2.11}$$

які часто виникають в граматиках, що представляють практичний інтерес.

У будь-який момент часу протягом парсингу ми маємо дві частково створені множини. СКАНЕР може додавати елементи в S_{i+1}, а в S_i можуть знаходитися елементи, додані до неї ПРЕДИКТОРОМ і ЗАВЕРШУВАЧЕМ.

Коли ЗАВЕРШУВАЧ обробляє елемент $[A \rightarrow \bullet, j]$, який відповідає правилу $A \rightarrow \varepsilon$, він повинен переглядати елементи в S_j з крапкою перед A. На жаль, для ε – елементів, j завжди дорівнює i, тому ЗАВЕРШУВАЧ буде проглядати лише

частково побудовану множину S_i. Реалізація алгоритму обробляє елементи S_i послідовно, тому якщо елемент $[B \rightarrow \ldots \bullet A \ \ldots, k]$ буде додано в S_i після того, як завершено обробку $[A \rightarrow \bullet, j]$, ЗАВЕРШУВАЧ ніколи не додасть $[B \rightarrow \ldots A \bullet \ldots, k]$ до S_i.

Своєю чергою елементи, які мали б бути отримані з $[B \rightarrow \ldots A \bullet \ldots, k]$, теж буде опущено. Це ефективно скорочує потенційні шляхи виведення, що може призвести до відхилення правильного входу.

Приклад 2.1: На рисунку 2.2 наведено приклад такої граматики та входу a.

$$\begin{aligned} S' &\rightarrow S \\ S &\rightarrow AAAA \\ A &\rightarrow a \\ A &\rightarrow E \\ E &\rightarrow \varepsilon \end{aligned} \tag{2.12}$$

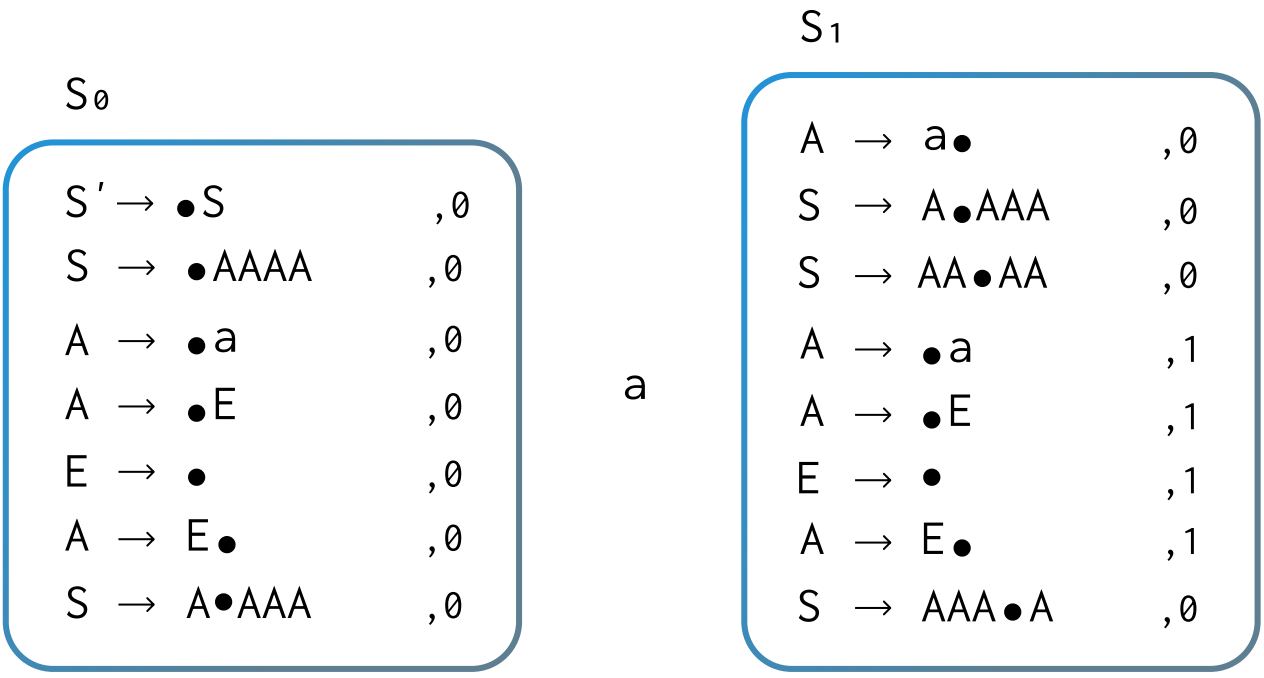


Рис. 2.3. Парсер Ерлі відхиляє коректний вхід

Для того, аби коректно обробляти ε – правила, ми скористалися рішенням запропонованим в [50]. Дане рішення містить модифікацію ПРЕДИКТОРА, яка базується на ідеї перевірки нетермінала, на вивідність ε.

Нетермінал A виводить ε, якщо $A \Rightarrow^* \varepsilon$. Термінальні символи, звичайно, ніколи не можуть виводити ε.

Дана властивість нетерміналів у граматиці може бути легко попередньо підрахована з використанням добре відомих методів [51].

Використовуючи це поняття, ПРЕДИКТОР може бути сформульований таким чином:

Якщо $[A \rightarrow \ldots \bullet B \ldots, j\,] \in S_i$, потрібно додати $[B \rightarrow \bullet\ \alpha, i]$ до S_i для всіх правил виду $B \rightarrow \alpha$. Якщо $B \Rightarrow^* \varepsilon$, також потрібно додати $[A \rightarrow \ldots B \bullet \ldots, j\,]$ до S_i. Іншими словами, ми одразу перемістили крапку за нетерміналом, якщо цей нетермінал $B \Rightarrow^* \varepsilon$.

Приклад 2.2: Використовуючи даний підхід зникають труднощі з ε правилами – рисунок 2.4.

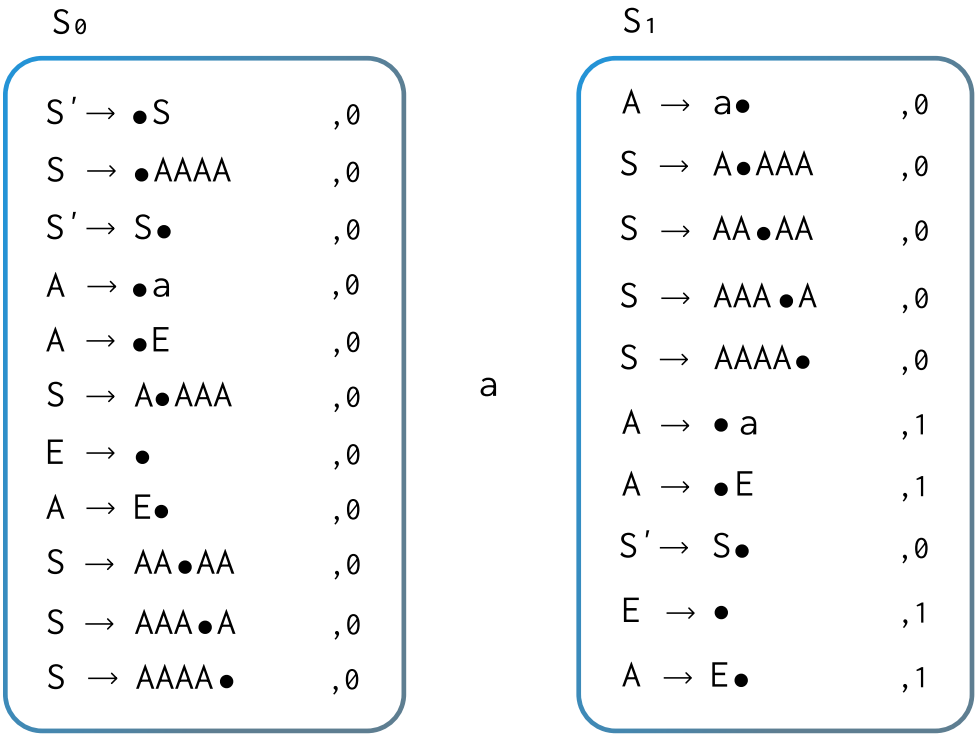


Рис. 2.4 – Парсер Ерлі приймає вхід a, використовуючи модифікацію в ПРЕДИКТОРІ

2.3.2. Модифікація алгоритму Ерлі для граматик великої потужності

У роботі з граматиками природної мови ми зіштовхнулися з тим, що потужність даних граматик дуже велика і класичний алгоритм Ерлі у випадку коректного речення на практиці працює досить довго. Це відбувається внаслідок того, що ми виконуємо функції ПРЕДИКТОР та ЗАВЕРШУВАЧ до тих пір, поки на конкретному кроці не досягнемо всіх можливих виведень під час побудови i-тої множини Ерлі. Таким чином кількість елементів у множинах Ерлі росте дуже швидко.

Ми пропонуємо модифікацію класичного алгоритму Ерлі з використанням підходу «повернення назад». Було розроблено евристику, яка зменшує час розбору коректної послідовності на порядок (базуючись на результатах тестування).

Основна ідея підходу полягає у тому, що на відміну від алгоритму Ерлі, під час розбору i-тої лексеми ми не пробуємо додавати до розгляду всі можливі продукції на цьому кроці, а робимо це доти, доки не з’явиться продукція з нашою наступною лексемою. У такому випадку ми переходимо до наступного етапу алгоритму. Якщо дана гілка виводу була не продуктивною, ми рекурсивно повертаємося на попередні етапи та продовжуємо послідовний аналіз продукцій.

Зосередимося на описі та модифікації самого алгоритму та його функцій.

Для кожної множини Ерлі S_i введемо поняття $predictElementIndex_i$ та $completeElementIndex_i$ які позначатимуть індекси у впорядкованій множині S_i елементів, до яких ще не було застосована функція ПРЕДИКТОР або ЗАВЕРШУВАЧ.

Також позначимо через $minMatchedElements$ кількість елементів, які повинні на наступному кроці алгоритму вивести наступний термінал у нашій вхідній послідовності.

Алгоритм 2.1. Модифікація алгоритму Ерлі використовуючи підхід «повернення назад».

Вхід: граматика $G =< N, \Sigma, P, S >, w \in \Sigma^*, w = w_1 w_2 \dots w_m$

Вихід: дерево розбору для w у випадку $w \in L(G)$.

Метод:

Крок 1. Нехай $j = 0$. Додамо $[Root \rightarrow \bullet\ S, 0]$ до I_0.

Крок 2. Ініціалізуємо змінні $predictElementIndex_j = 0$, $completeElementIndex_j = 0, matchedElements = 0$.

Крок 3: Виконаємо функцію СКАНЕР. Для кожного елемента в I_j такого виду $[A \rightarrow \alpha \bullet \gamma B\beta, i]$ де $\gamma = w_{j+1}$ додамо $[A \rightarrow \alpha\ \gamma \bullet B\beta, i]$ до I_{j+1}, при кожному такому додаванні оновимо $matchedElements \mathrel{+}= 1$. Якщо $matchedElements = minMatchedElements$, тоді оновимо $j = j + 1$ та перейдемо до Кроку 2.

- Очевидно, що якщо $j = 0$ та лише один елемент $[Root \rightarrow \bullet\ S, 0]$

знаходиться у цій множині, то ми пропускаємо цей крок.

Крок 4. Виконаємо функцію ПРЕДИКТОР. Будемо обробляти всі елементи з множини I_j індекс яких $x \leq predictElementIndex_j$. Якщо елемент з порядковим індексом x $[A \rightarrow \alpha \bullet B\beta, i]$ знаходиться в I_j і є $B \rightarrow \gamma$ правило в P, тоді додаймо елемент $[B \rightarrow \bullet\ \gamma, j]$ до I_j. Після того, як обробимо всі правила виду $B \rightarrow \gamma$ для елемента з індексом x, оновимо $predictElementIndex_j = x + 1$.

Якщо правило мало вигляд $B \rightarrow \alpha\gamma, \alpha \in \Sigma, \gamma \in (\mathrm{N} \cup \Sigma)^*$ та $\alpha = w_{j+1}$ тоді оновимо $matchedElements \mathrel{+}= 1$ та виконаймо функцію СКАНЕР: додамо $[B \rightarrow \alpha\ \bullet \gamma, j]$ до I_{j+1}.

Якщо $matchedElements = minMatchedElements$, тоді оновимо $j = j + 1$ та перейдемо до Кроку 2.

Крок 5. Виконаємо функцію ЗАВЕРШУВАЧ. Будемо поступово обробляти всі елементи з множини I_j, індекс яких $x \leq completeElementIndex_j$. Якщо елемент з порядковим індексом x $[A \rightarrow \alpha \bullet, i] \in I_j$ та $[B \rightarrow \beta \bullet A\gamma, k] \in I_i$, то додаймо елемент $[B \rightarrow \beta A \bullet \gamma, k]$ до I_j. Після того, як обробимо всі елементи виду $[B \rightarrow \beta \bullet A\gamma, k] \in I_i$ для елемента з індексом x, оновимо $completeElementIndex_j = x + 1$.

Якщо елемент мав вигляд $[B \rightarrow \beta A \bullet \gamma\delta, k], \gamma \in \Sigma, \delta \in (\mathrm{N} \cup \Sigma)^*$ та $\gamma = w_{j+1}$, тоді оновимо $matchedElements \mathrel{+}= 1$ та виконаймо функцію СКАНЕР: додамо $[B \rightarrow \beta A\gamma \bullet \delta, k]$ до I_{j+1}.

Якщо $matchedElements = minMatchedElements$, тоді оновимо $j = j + 1$ та перейдемо до Кроку 2.

Крок 6. У випадку, якщо I_j пуста множина, або ми обробили всі елементи цієї множини, але не можемо переходити до наступної – цьому є дві можливі причини. Перша – це те, що вхідний рядок є недопустимим для цієї граматики, інша – ми не розкрили потрібну гілку в одній з попередніх кроків за рахунок так званої «лінивої обробки». В такому випадку потрібно повернутися на крок назад $j = j - 1$ та продовжити обробку елементів починаючи з $predictElementIndex_{j-1}$, $completeElementIndex_{j-1}$. Тобто перейти до Кроку 3. У випадку успішного знаходження нових шляхів виводу важливо зауважити, що при переході до наступного кроку ми повинні завжди оновлювати змінну $completeElementIndex_j = 0$, адже попередні множити $I_{j-1}, I_{j-2}, \ldots I_0$ могли оновитися, а функція ЗАВЕРШУВАЧ якраз використовує їх.

Крок 7. Якщо елемент $[Root \rightarrow S \bullet, 0]$ знаходиться в I_m, $|w| = m$, дерево розбору можна отримати послідовно аналізуючи множини I_j. Завершуємо роботу алгоритму.

Якщо ж ми обробили всі множини та такий елемент не знайдено – значить даний вхід є некоректним.

Приклад 2.3: Спробуємо розглянути приклад роботи алгоритму на таких вхідних даних:

Граматика:

$$\begin{gathered} S \rightarrow B \\ B \rightarrow ( ) \\ B \rightarrow (B) \\ B \rightarrow B\ B \end{gathered} \tag{2.13}$$

Вхідна послідовність: (())

Додаткові параметри: $minMatchedElements = 1$

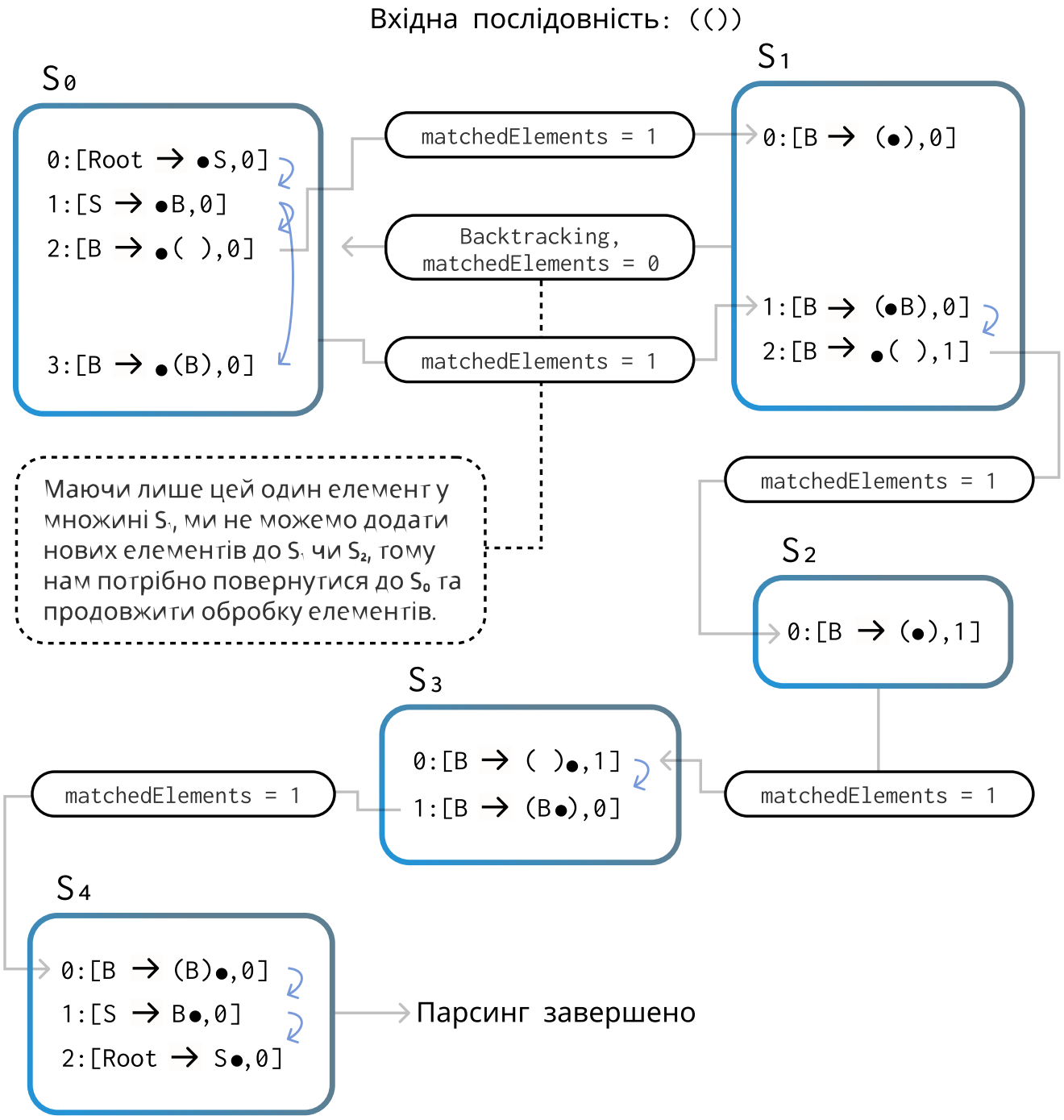


Рис. 2.5. – Приклад роботи модифікованого парсера Ерлі з «поверненням назад»

Аналізуючи даний приклад, можна помітити, що після того, як ми вперше перейшли до множини S_1, парсер був змушений повернутися назад до S_0. Проте після цього повернення і доопрацювання множини S_0 він зміг коректно завершити парсинг вхідної послідовності.

Даний евристичний алгоритм користується такими припущеннями, що дуже часто на практиці ми працюємо з неоднозначними граматиками, де немає єдиного варіанту виведення. Таким чином навіть при опущенні деяких гілок виводу ми не втратимо можливість вивести вхідну послідовність. Даний алгоритм також буде ефективним лише у випадку коректності входу, або високої ймовірності його коректності. Це можна пояснити тим, що у випадку некоректного входу алгоритм буде дуже часто повертатися рекурсивно назад і таким чином ми втратимо всі його переваги.

Ефективність даної евристики залежить від послідовності правил виводу. Якщо для кожного нетерміналу A відсортувати всі його правила у порядку спадання ймовірності його використання, ми отримаємо більш ефективний алгоритм. Адже спочатку будемо пробувати розкривати найбільш ймовірні гілки розбору. Таким чином ми будемо максимізувати ймовірність того, що парсер зможе перейти до наступного крок алгоритму (побудови наступної множини Ерлі).

2.3.3. Модифікація алгоритму Ерлі для знаходження декількох дерев виведення

Класичний алгоритм Ерлі дозволяє отримати лише одне виведення для вхідного рядка w. Це пов’язано з тим, що при роботі з елементами множин Ерлі унікальність елемента $[B \rightarrow \beta \bullet \gamma,\ k]$ визначається лише по правилу виведення, місці знаходження крапки та k – індексу множини, в якій знаходиться батьківський елемент. Тому, якщо у випадку неоднозначної граматики ми отримаємо два однакових елементи, які мають фактично різні шляхи виведення, ми при додаванні у множину I_j не зможемо їх розрізнити.

Для знаходження виправлень дуже важливою є можливість на виході отримати не одне дерево розбору, а декілька, адже це дасть змогу більш гнучко оцінити речення та обрати правильне виправлення.

Таким чином додаймо до елементу множини Ерлі ще одне поле – вказівник на елемент, який ми щойно завершили виводити до нього.

Розглянемо новий модифікований алгоритм Ерлі:

Алгоритм 2.2. Модифікація алгоритму Ерлі для знаходження декількох дерев виведення

Вхід: граматика $G =< N, \Sigma, P, S >, w \in \Sigma^*$.

Вихід: множину дерев розбору для w у випадку $w \in L(G)$.

Метод:

Крок 1. Нехай $j = 0$. Додаймо $[Root \rightarrow \bullet\ S, 0, null]$ до I_j.

Крок 2. Виконаємо функцію СКАНЕР. Для кожного елемента в I_{j-1} виду $[A \rightarrow \alpha \bullet \gamma B\beta,\ i,\ p]$, де $\gamma = w_{j-1}$ додамо $[A \rightarrow \alpha\ \gamma \bullet B\beta,\ i,\ p]$ до I_j.

Крок 3. Виконаємо функцію ПРЕДИКТОР. Якщо елемент $[A \rightarrow \alpha \bullet B\beta,\ i,\ p]$ знаходиться в I_j і є $B \rightarrow \gamma$ правило в P, тоді додамо елемент $[B \rightarrow \bullet \gamma,\ j, null]$ до I_j.

Крок 4. Виконаємо функцію ЗАВЕРШУВАЧ. Якщо елемент $[A \rightarrow \alpha \bullet,\ i,\ p] \in I_j$ та $[B \rightarrow \beta \bullet A\gamma,\ k,\ g] \in I_i$, то додамо елемент $[B \rightarrow \beta A \bullet \gamma,\ k,\ p']$ до I_j, де p' вказівник на $[A \rightarrow \alpha \bullet,\ i,\ p]$.

Крок 5. Якщо елемент $[Root \rightarrow S \bullet, 0, p]$ знаходиться в $I_m, |w| = m$, дерева розбору можна отримати послідовно аналізуючи множини I_j, інакше – даний вхід є некоректним. Завершуємо роботу алгоритму.

Даний алгоритм описує механізм визначення вказівника на останній повністю розкритий елемент. Цього достатньо, щоб отримати не одне, а декілька дерев.

Для покращення роботи алгоритму по пам’яті можна не створювати надлишкові елементи, які відрізняються лише вказівником. Потрібно визначити множину вказівників та при знаходженні однакових елементів оновлювати лише її. Це дозволить зменшити кількість ресурсів, які використовує алгоритм.

Варто зазначити, що дана модифікація знайде всі можливі дерева розбору, що в свою чергу теж не зовсім те, що потрібно, адже кількість таких дерев росте

дуже швидко. Отже, потрібно ввести обмеження на кількість «майже» однакових елементів у одній множині і у випадку перевищення цієї кількості – відкидати елемент. Оскільки цей елемент буде присутній у множині, проте трохи з іншою «історією» виводу, можна стверджувати, що ми не втратимо потенційні шляхи для пошуку коректного дерева розбору.

2.4. Алгоритм граматичної корекції

Граматична корекція помилок (GEC) – це задача виправлення різних видів помилок у тексті, таких як орфографічні, пунктуаційні, граматичні та помилки вибору слова.

GEC, зазвичай, формулюється як завдання виправлення речення. Система GEC приймає потенційно помилкове речення в якості вхідних даних і, як очікується, перетворить його на свою виправлену версію.

Нам було цікаво дослідити можливість використання синтаксичного дерева для виконання граматичної корекції, а саме помилок у структурі речення. Структура речення буде визначатися через синтаксичне дерево речення, яке буде побудоване з різних частин мови (POS тегів слів).

2.4.1. Поняття відстані між рядками

Відстань між двома рядками будемо визначати в термінах мінімального числа помилкових перетворень, які використовуються для виведення одного з іншого, як було зазначено Ахо і Петерсоном [52]. Коли перетворення помилок визначається в термінах помилок заміщення, видалення і вставки, вимір відстані збігається з визначенням метрики Левенштейна [53].

Для заданого вхідного речення y і заданої граматики G аналізатор з виправлення помилок являє собою алгоритм, який шукає речення z в $L(G)$ таке, щоб відстань між z і y, $d(z,y)$ була мінімальною серед відстаней між усіма

реченнями в $L(G)$ і y. Алгоритм також генерує значення $d(z, y)$. Ми просто визначаємо це значення, як відстань між $L(G)$ і y та позначаємо її як $d(L(G), y)$.

Коли дана граматика є контекстно-вільно граматикою, даний підхід може бути реалізований шляхом модифікації алгоритму Ерлі.

Також можна розширити визначення відстані між $L(G)$ і y, $d(L\ (G), y)$ до визначення $d^k(L(G), y)$, середньої відстані між y і k пропозиціями в $L(G)$, які є найближчими до y.

2.4.2. Міра подібності рядків

Для початку розглянемо помилки трьох типів [52]: заміна, видалення і вставка та будемо обробляти їх, як помилки, які визначають перетворення з Σ^* в підмножину Σ^*.

Визначення 2.7. Для двох рядків $x, y \in \Sigma^*$, ми можемо визначити перетворення T: $\Sigma^* \rightarrow \Sigma^*$ таке, що $y \in T(\text{x})$.

Введено наступні три перетворення:

1. Перетворення помилок заміщення:

$$w_1 a w_2 \overset{T_S}{\Rightarrow} w_1 b w_2, \qquad \forall\, a, b \in \Sigma, \qquad a \neq b \tag{2.14}$$

2. Перетворення помилок видалення:

$$w_1 a w_2 \overset{T_D}{\Rightarrow} w_1 w_2, \qquad \forall\, a \in \Sigma \tag{2.15}$$

3. Перетворення помилок вставки:

$$w_1 w_2 \overset{T_I}{\Rightarrow} w_1 a w_2, \qquad \forall\, a \in \Sigma, \qquad w_1, w_2 \in \Sigma^* \tag{2.16}$$

Визначення 2.8. Відстань між двома рядками $x, y \in \Sigma^*$ – $d(x, y)$ визначається, як найменша кількість необхідних перетворень для виведення y з x.

Приклад 2.4. Дано рядок $x = cbabdbb$ і рядок $y = cbbabbdb$, тоді

$$x = cbabdbb \overset{T_S}{\Rightarrow} cbabbbb \overset{T_S}{\Rightarrow} cbabbdb \overset{T_I}{\Rightarrow} cbbabbdb = y\,.$$

Мінімальна кількість перетворень, необхідних для перетворення x в y три, таким чином, $d(x, y) = 3$.

Метрика, визначена вище, дає в точності відстань Левенштейна для двох рядків [53]. Зважена відстань Левенштейна може бути визначена присвоєнням невід'ємних чисел σ, γ та δ перетворенням T_S, T_D та T_I відповідно.

Визначення 2.9. Нехай $x, y \in \Sigma^*$ два рядки, а J – послідовність перетворень, які використовуються для отримання y з x, тоді зважену відстань Левенштейна між x і y позначимо як $d^w(x, y)$:

$$d^w(x, y) = \min_J\{\sigma * k_j + \gamma * m_j + \delta * n_j\} \tag{2.17}$$

де k_j, m_j, n_j відповідно кількість перетворень підстановок, видалення і вставки в J.

Також ми пропонуємо використовувати зважену метрику, що буде відображати різницю одного і того ж типу помилки, зроблену на різних терміналах.

Визначимо її таким чином:

1. Перетворення помилок заміщення:

$$w_1 a w_2 \xrightarrow{T_S, S(a,b)} w_1 b w_2, \qquad \forall\, a, b \in \Sigma, \qquad a \neq b, \tag{2.18}$$

$$S(a, b) - \text{ ціна заміни терміналу } a \text{ на } b, \text{де } S(a, a) = 0 \tag{2.19}$$

2. Перетворення помилок видалення:

$$w_1 a w_2 \xrightarrow{T_S, D(a)} w_1 w_2, \qquad \forall\, a \in \Sigma, \tag{2.20}$$

$$D(a) - \text{ціна видалення терміналу } a \tag{2.21}$$

3. Перетворення помилок вставки:

$$w_1 a w_2 \xrightarrow{T_S, I(a,b)} w_1 b a w_2, \qquad \forall\, a, b \in \Sigma, \tag{2.22}$$

$$I(a, b) - \text{ ціна вставки терміналу } a \text{ перед } b \tag{2.23}$$

Визначення 2.10. Нехай $x, y \in \Sigma^*$ – два рядки, а J – послідовність перетворень для отримання y з x. Через $|J|$ позначимо як суму ваг, пов'язаних з перетвореннями в J, тоді зважена відстань між x і y:

$$d^W(x, y) = \min_J\{|J|\} \tag{2.24}$$

2.4.3. Коригуючий парсер

Нехай $L(G)$ – задана мова, а y – речення. Сутність розбору, що виправляє помилки використовуючи мінімальну відстань, полягає в пошуку речення x в $L(G)$, яке задовольняє наступний критерій мінімальної відстані:

$$d(x, y) = \min_z\{d(z, y) \mid z \in L(G)\} \tag{2.25}$$

Зауважимо, що корекція мінімальної відстані для y – це саме y, якщо $y \in L(G)$.

Процедура побудови парсеру починається з модифікації даної граматики G за допомогою додавання різних типів перетворення помилок у формі правил, так званих «помилкових» правил. Тепер граматика G розширена до G', так що $L(G')$ включає в себе не тільки $L(G)$, але і всі можливі речення з різними типами помилок.

Аналізатор, побудований відповідно до G' з додаванням методу для підрахунку кількості помилок, використаних в розкритті дерева, є аналізатором з виправленням помилок для G.

Для даного речення y буде згенеровано розбір P, яке містить найменшу кількість помилкових продукцій. Речення $x \in L(G)$, яке відповідає критерію мінімальної відстані, може бути створене з P шляхом усунення помилок.

Структура нашого синтаксичного аналізатора з виправленням помилок базується на ідеях Марченка [54] та Фу [55]. Ними було запропонувано ідею модифікації оригінальної граматики мови таким чином, щоб вона могла створювати не лише правильні речення, але й неправильні. На основі дерева розбору ми можемо скоригувати наше вхідне речення.

До цього часу ми розглядали лише типи на найнижчому рівні – рівні терміналів, проте досить часто можна зустріти помилки, де фактично речення складається з правильних «частин», проте вони переплутані місцями. Якщо використовувати попередній підхід для того, аби повністю скоригувати даний вхід, потрібно, зазвичай, застосувати велику кількість локальних виправлень на рівні терміналів. Для того, аби перейти на вищий рівень абстракції, ми пропонуємо розглянути методику створення коригуючої граматики, яка може виправляти помилки такого типу: перестановка місцями двох послідовних нетерміналів.

Введемо наступне перетворення:

1. Перетворення помилки перестановки нетерміналів:

$$w_1abw_2 \overset{T_P}{\Rightarrow} w_1baw_2, \qquad \forall\, a,b \in \Sigma^*, \qquad \exists\, A \Rightarrow^* a, \exists\, B \Rightarrow^* b, A, B \in N \quad (2.26)$$

Також ми пропонуємо використовувати зважену метрику, що буде відображати різницю одного і того ж типу помилки, зроблену на різних нетерміналах.

Визначимо її таким чином:

1. Перетворення помилки перестановки нетерміналів:

$$w_1abw_2 \overset{T_P,\ R(A,B)}{\Longrightarrow} w_1baw_2, \qquad \forall\, a,b \in \Sigma^*, \quad \exists\, A \Rightarrow^* a, \exists\, B \Rightarrow^* b, A, B \in N \quad (2.27)$$

$$R(A,B) - \text{ ціна перестановки нетерміналу } A \text{ з } B \quad (2.28)$$

2.4.4. Алгоритм побудови розширеної граматики

Розглянемо алгоритм побудови розширеної граматики, в якій невід'ємні числа, пов'язані з «помилковими» правилами, є вагами, пов'язаними з відповідними помилками.

Алгоритм 2.3. Побудова розширеної граматики

Вхід: Контекстно-вільна граматика $G - (N, E, P, S)$

Вихід: Контекстно-вільна граматика $G' - (N', E', P', S')$, де P' - множина зважених правил.

Метод:

Крок 1. $N' = N \cup \{S'\} \cup \{E_a \mid a \in \Sigma\} \cup \{C_{\alpha_1\alpha_2\ldots\alpha_m} \mid \alpha_i \in \mathrm{N}\}$, $\Sigma' \supseteq \Sigma$.

Крок 2. Якщо $A \to \gamma\,\alpha_1\alpha_2 \ldots \alpha_m\beta, m \geq 1$ – правило в P, де $\alpha_i \in N$, $\gamma \in (\Sigma \cup N)^*\Sigma^*, \beta \in \Sigma^*(\Sigma \cup N)^*$ то потрібно додати наступне правило $A \to \gamma\, C_{\alpha_1\alpha_2\ldots\alpha_m}\beta$, 0 до P', де $C_{\alpha_1\alpha_2\ldots\alpha_m}$ – новий нетермінал, а 0 – вага даного правила.

Крок 3. Якщо $A \to \alpha_0 b_1 \alpha_1 b_2 \ldots b_m \alpha_m, m \geq 0$ – правило в P', де $\alpha \in N'^*$, $b \in \Sigma$, замінити його на наступне правило $A \to \alpha_0 E_{b_1}\alpha_1 E_{b_2} \ldots b_m E_{b_m}$, 0 до P', де E_{b_i} – новий нетермінал, $E_{b_i} \in N'$, а 0 – вага даного правила.

Крок 4. Також додамо наступні правила до P':

Таблиця 2.1. Правила для виправлення помилок

Правило	Вага
$S \to S'$	0
$E_a \to a, \forall a \in \Sigma$	0
$E_a \to b, \forall a \in \Sigma, \mathrm{b} \in \Sigma', a \neq b$	S(a, b)
$E_a \to \varepsilon, \forall a \in \Sigma$	D(a)
$E_a \to bE_a, \forall a \in \Sigma, \mathrm{b} \in \Sigma'$	I(a, b)
$C_{\alpha_1\alpha_2\ldots\alpha_m} \to \alpha_1\alpha_2 \ldots \alpha_m, \forall\alpha_i \in \mathrm{N}$	0
$C_{\alpha_1\ldots\alpha_i\alpha_{i+1}\ldots\alpha_m} \to \alpha_1 \ldots \alpha_{i+1}\alpha_i \ldots \alpha_m, \forall\alpha_i \in \mathrm{N}$	$\mathrm{R}(\alpha_i, \alpha_{i+1})$

В алгоритмі 2.3 продукції, додані на четвертому кроці називаються правилами помилок. Кожна з цих продукцій відповідає одному типу помилкового перетворення в конкретному символі. Отже, відстань, виміряна в термінах помилкових перетворень, може бути виміряна за допомогою помилкових продукцій, які будуть використовуватися при виведенні.

2.4.5. Алгоритм розбору з виправленням помилок

Парсер – це модифікований алгоритм Ерлі з додаванням функціоналу для накопичення ваг, пов'язаних з продукціями, які використовуються у виведенні дерева. Алгоритм полягає в наступному.

Алгоритм 2.4. Алгоритм розбору з виправленням помилок, використовуючи мінімальну відстань.

Вхід: розширена граматика $G'(N', E', P', S')$ і вхідний рядок

$y = b_1 b_2 \dots b_m \in \Sigma^*$

Вихід: список множин $I_1, I_2 \dots I_m$ для у та $d(x, y)$, де x – це корекція речення y.

Метод:

Крок 1. Нехай $j = 0$. Додамо $[Root \rightarrow \bullet\ S', j, \xi]$ до I_j, де ξ вага, пов'язана з правилом. Для даного початкового правила $\xi = 0$.

Крок 2. Якщо $[A \rightarrow \alpha \bullet B\beta,\ i,\ \xi]$ знаходиться в I_j і є $B \rightarrow \gamma, \eta$ правило в P', тоді додамо елемент $[B \rightarrow \bullet\, \gamma,\ j,\ \eta]$ до I_j.

Крок 3. Якщо $[A \rightarrow \alpha \bullet,\ i,\ \xi] \in I_j$ та $[B \rightarrow \beta \bullet A\gamma,\ k,\ \eta] \in I_i$ та в I_j немає елемента виду $[B \rightarrow \beta A \bullet \gamma,\ k,\ \phi]$, то додамо елемент

$[B \rightarrow \beta A \bullet \gamma,\ k,\ \xi + \eta]$ до I_j.

Якщо $[B \rightarrow \beta A \bullet \gamma,\ k,\ \phi]$ вже знаходиться в I_j, тоді

$$\phi = \min(\phi,\quad \xi + \eta).$$

Крок 4. Якщо $j = m$, перейти до Кроку 6, в іншому випадку $j = j + 1$.

Крок 5. Для кожного елемента в I_{j-1} виду $[A \rightarrow \alpha \bullet b_j\beta,\ i,\ \xi]$ додати елемент $[A \rightarrow \alpha\, b_j \bullet \beta,\ i,\ \xi]$ до I_j та перейти до Кроку 2.

Крок 6. Якщо елемент $[Root \rightarrow S' \bullet, 0, \xi]$ знаходиться в I_m, тоді $d(x, y) = \xi$, де x – виправлення речення y.

Рядок x, який є корекцією мінімальної відстані y, може бути отриманий з розбору y шляхом виключення всіх помилок, що виникають.

2.5. Експерименти та аналіз роботи алгоритму виправлення структури речень із простими граматиками

Спочатку для перевірки концепції спробуємо перевірити роботу на простій граматиці коректних дужкових послідовностей. Для кожного входу будемо вказувати до двох можливих модифікацій, знайдених алгоритмом.

Для кожного типу помилки, не залежно від терміналів та нетерміналів, поставимо вагу рівну 1.

Розглянемо таку граматику $G(N, \Sigma, P, S)$ зі схемою P:

$$S \to (S) \mid S\ S \mid () \tag{2.29}$$

Приклад 2.5. Вхід:)(

Виправлення - ()().

Вага корекції – 0.

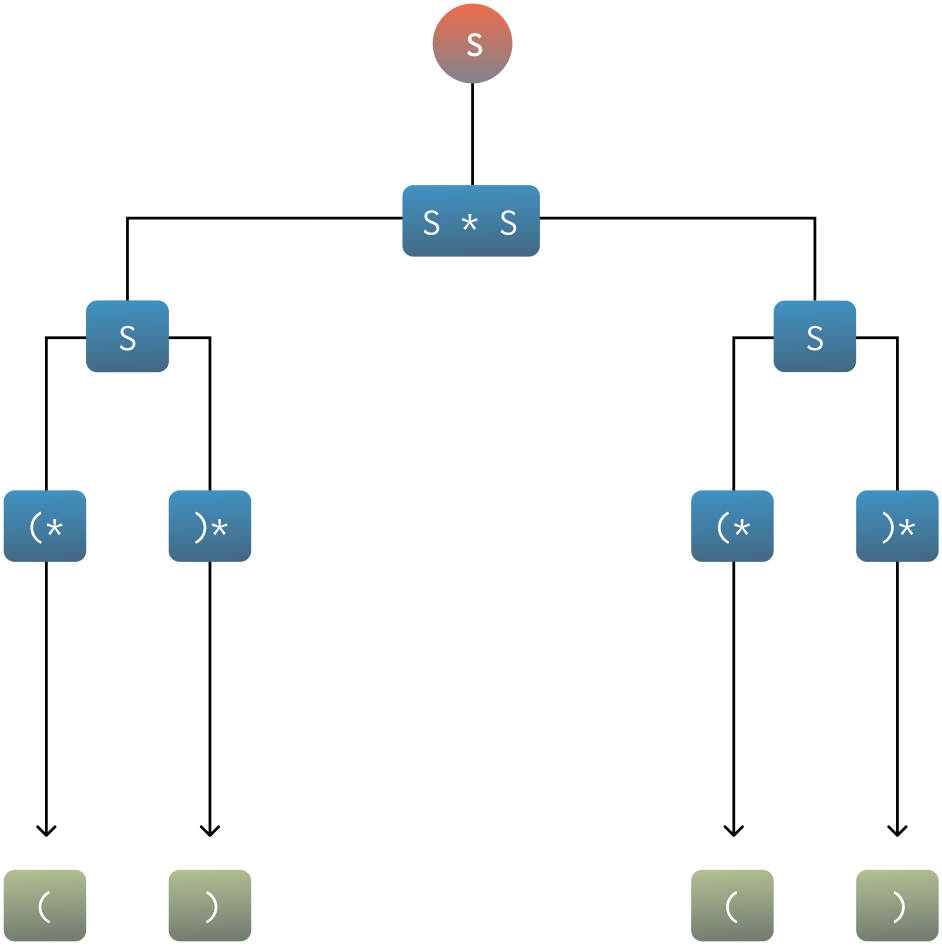


Рис. 2.6. – Дерево розбору послідовності ()()

S*S позначає нетермінал, утворений групуванням нетерміналів у правилі $S \to S\,S$, а (* та)* представляють собою нетермінали, створені при побудові розширеної корегуючої граматики.

Приклад 2.6. Вхід: ()))

Виправлення – (())

Вага корекції – 2.

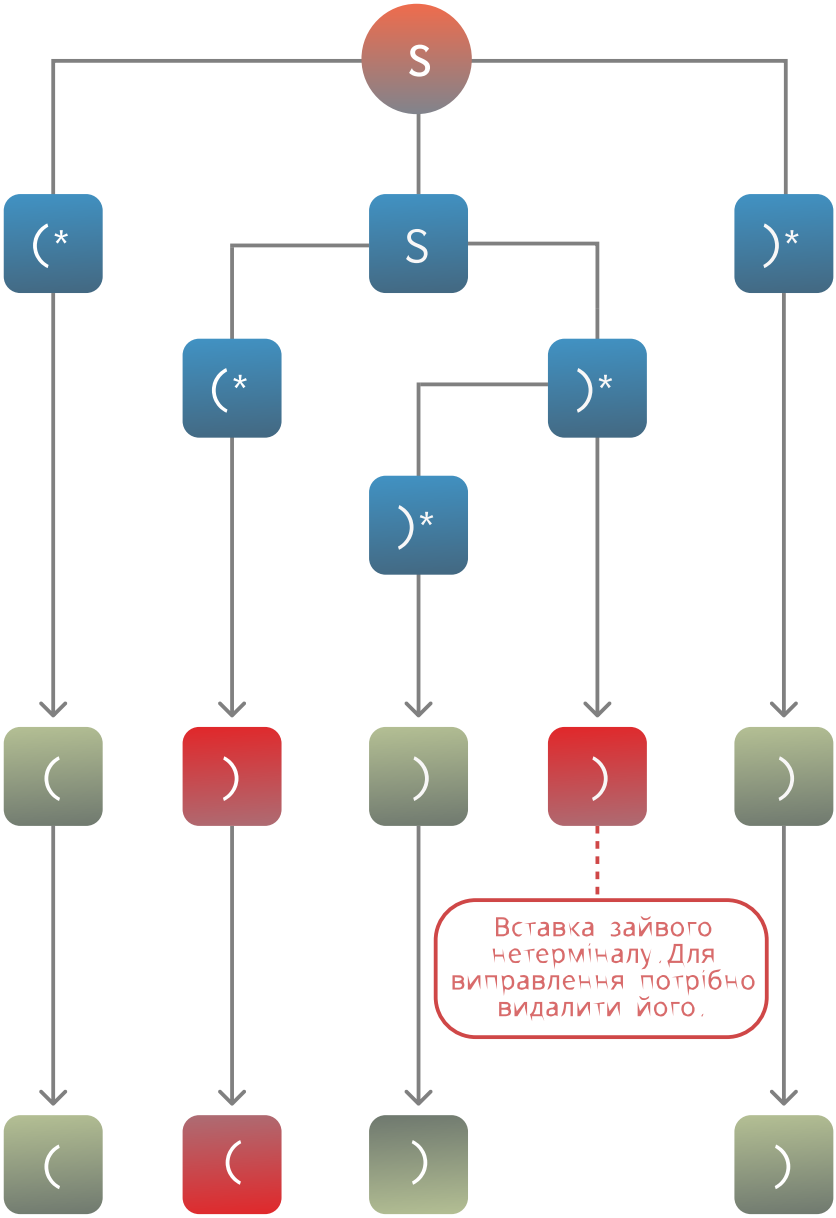


Рис. 2.7. – Дерево розбору послідовності ()))

Спробуємо ускладнити задачу та додамо інший тип дужок. Маємо граматику $G(N, \Sigma, P, S)$ зі схемою P:

$$S \to B\ M, B\ \to (B)|\ B\ B\ |\ (), M\ \to \{M\}|\ M\ M\ |\ \{\} \qquad (2.30)$$

Приклад 2.7. Вхід: {}}(()

Виправлення – (){}{}

Вага корекції – 3.

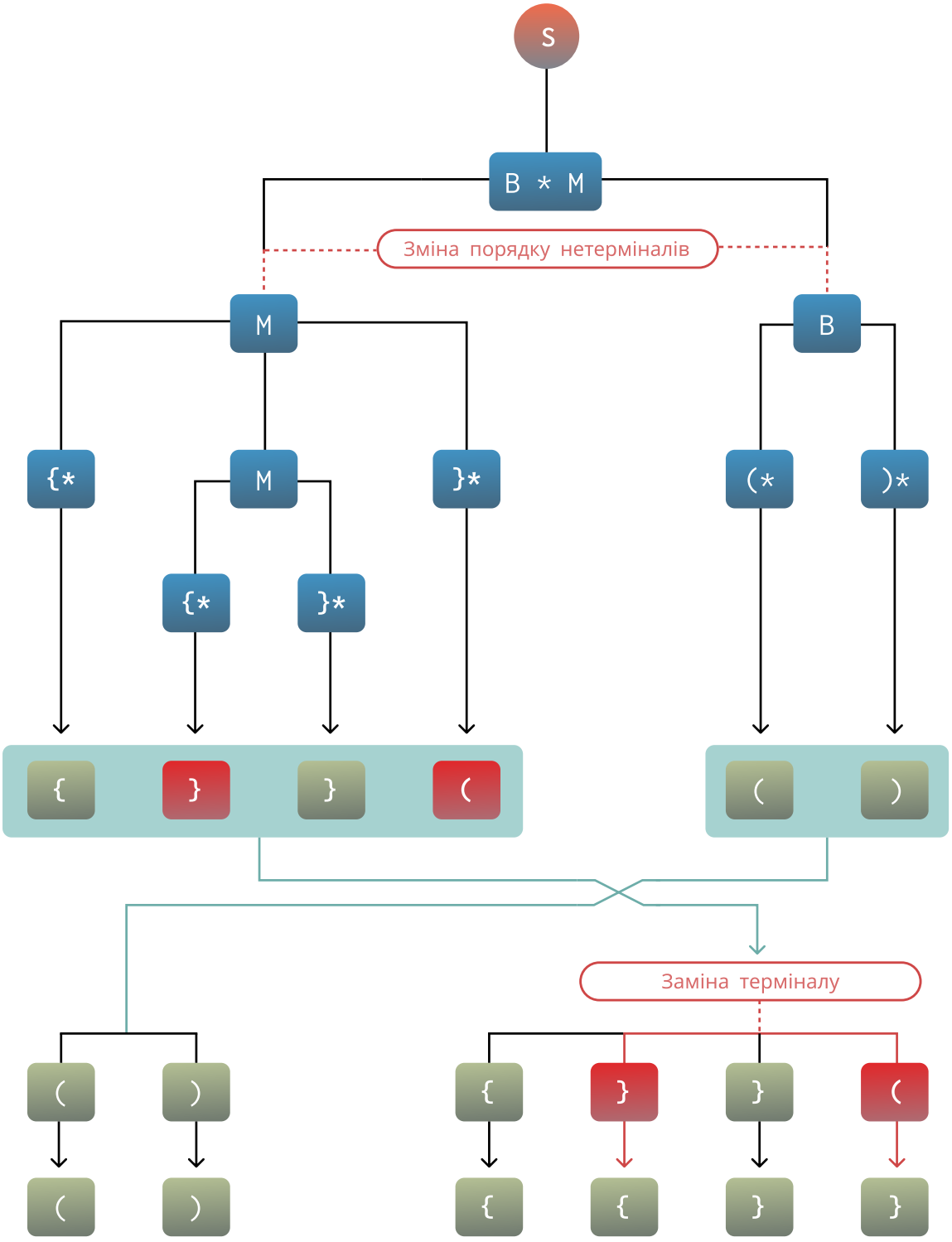


Рис. 2.8. – Дерево розбору і корекція послідовності {}}(()

Таким чином ми переконалися, що даний алгоритм та реалізація успішно виправляє помилки в реченнях для досить простих граматик. Спробуємо протестувати на частковій граматиці англійської мови.

2.6. Експерименти та аналіз роботи алгоритму виправлення структури речень з частковою граматикою англійської мови

Поки нам не вдалося створити чи зібрати повну граматику англійської мови, тому для тестування ми будемо брати невеликий коректний текст англійської мови та, базуючись на ньому, виділяти граматичні правила використовуючи POS теги. На даний момент ми побудували граматику потужністю близько 2000 правил.

Після цього ми штучним чином зробимо декілька помилок в оригінальному тексті та спробуємо перевірити наш алгоритм на ньому.

Приклад 2.8. Для цього прикладу для простоти розглянемо під граматику нашої збудованої граматики $G(N, \Sigma, P, S)$ зі схемою P:

$$S \rightarrow VP\ . \tag{2.31}$$

$$PP \rightarrow IN\ NP$$

$$VP \rightarrow VB\ PP\ ADVP$$

$$NP \rightarrow NP\ PP \mid DT\ NNS \mid NNS$$

$$ADVP \rightarrow RB$$

Вхід: Look the crowds of water-gazers there.

Виправлення: Look <IN> the crowds of water-gazers there.

Вага корекції – 1.

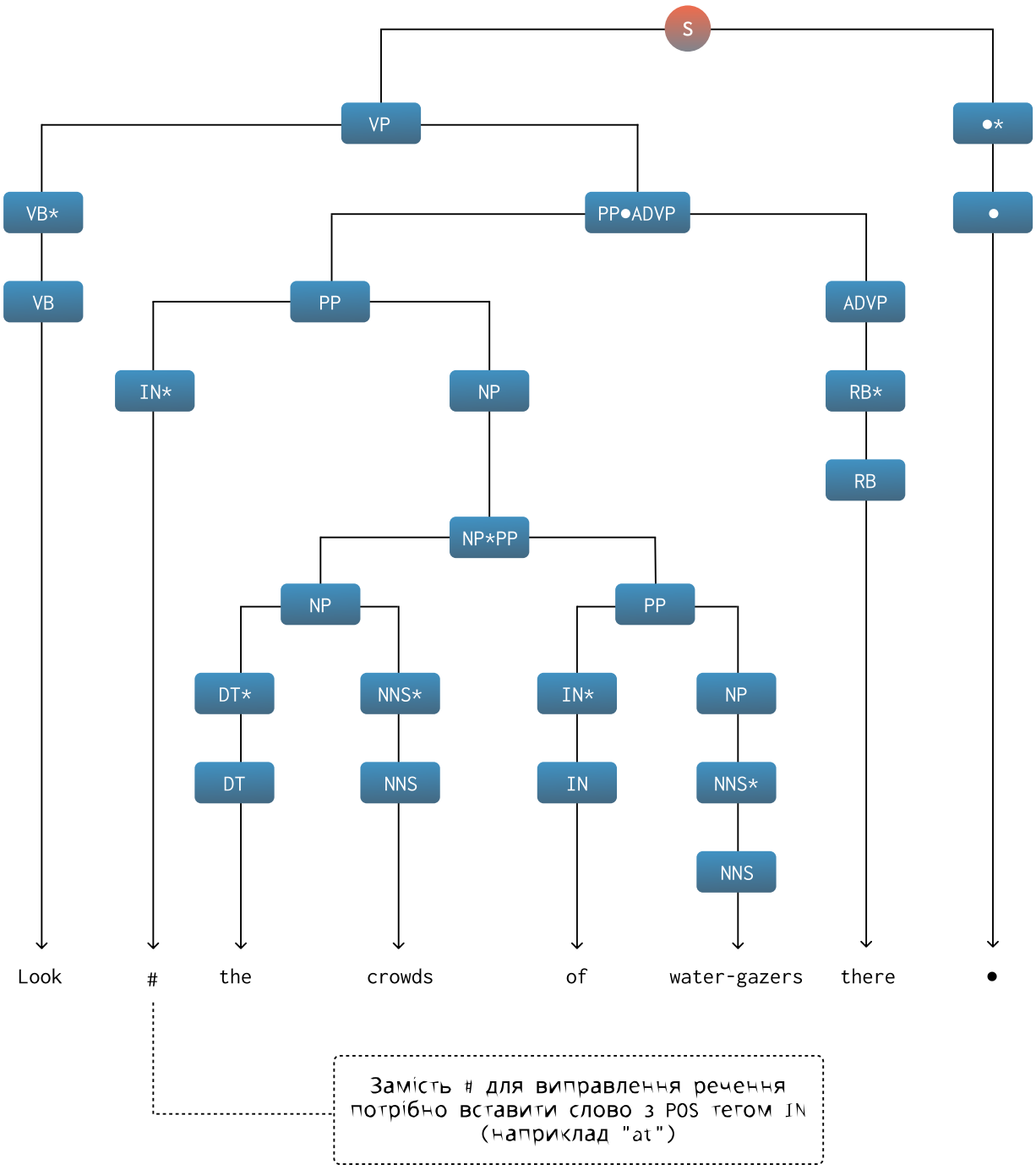


Рис. 2.9. – Дерево розбору речення “Look the crowds of water-gazers there.”

2.7. Порівняння коригуючого парсера з іншими системами для виправлення граматичних помилок

Є кілька добре відомих корпусів GEC для оцінювання, наприклад, корпус Сінгапурського національного університету англійської мови (NUCLE) — це

корпус, створений NLP Group Національного університету Сінгапуру (NUS) у співпраці з Центром англійської комунікації NUS [56] для завдання CoNLL-2013 [57].

На даний момент ми не маємо повного опису синтаксичної структури англійської мови в нашій моделі. Таким чином, вищезазначені документи не відповідають вимогам, які забезпечують точну оцінку запропонованої системи - а саме наш парсер може не розпізнати коректне речення у даному корпусі. Тому однією з умов є те, що текст має використовувати лише синтаксичні структури, які вже існують у нашій моделі. Інший полягає в тому, що в текстах повинні бути присутніми лише ті типи помилок, які вже підтримуються моделлю. Щоб перевірити ефективність розробленої моделі пошуку та виправлення граматичних помилок, необхідно було створити власний текстовий корпус із анотованими помилками.

За основу було взято JFLEG Corpus [58]. Речення були послідовно оброблені нашим аналізатором. Ми включили лише речення, які були успішно оброблені без помилок. В інших випадках ми їх відхиляли. Всього було обрано 286 речень з 747 речень тестової вибірки JFLEG.

Для перевірки нашої моделі ми обрали декілька моделей – одну MT Baseline — модель машинного перекладу, навчену на корпусі даних Lang-8. Також ми порівнюємо нашу ефективність з однією з найкращих моделей на момент проведення експерименту – SMEC [59]. Для того, аби порівняти наш парсер з цими моделями, були використані результати SMEC та MT Baseline на корпусі JFLEG, які можна знайти за цим посиланням [60].

Варто зазначити, що дані експерименти були проведені у 2019 році. З того часу з’явилося багато інших кращий моделей, проте фокус дослідження перемістився до побудови векторного представлення речення (див. наступні розділи), тому ми не наводимо порівняння з іншими сучаснішими моделями.

Таблиця 2.2. Порівняння моделей виправлення граматичних помилок

Система	Точність	Повнота	F0.5
MT Baseline	28.1%	4.4%	13.5%
CFG Parser	21.1%	9.6%	17.1%
SMEC	**44.9%**	**34.9%**	**42.47%**

Наша система перевершує базову модель, але дає гірші результати, ніж SMEC. Це можна пояснити тим фактом, що для нашої системи ми використовуємо лише POS-теги, тоді як SMEC використовує багато інших параметрів, як-от контекст та інші властивості на основі слів.

Нашою основною метою було спробувати лише структурний підхід POS-тегів для тестів на виправлення граматичних помилок. Часто важко інтерпретувати результати моделей, які базуються на машинному чи глибокому навчанні. Ми хотіли використати правила граматики як основне джерело, щоб природньо пояснити результат кінцевому користувачеві. Цей експеримент доводить, що цієї інформації недостатньо, щоб конкурувати з сучасними моделями, але було показано, що дану структуру можна використовувати як важливу характеристику при побудові моделей.

2.8. Висновки до розділу 2

В результаті проведеного дослідження синтаксичного аналізу речення на основі алгоритму Ерлі було розглянуто різні методи модифікації даного алгоритму. Було запропоновано метод коригуючого парсера, використовуючи модифікацію до вхідної граматики послідовності. В результаті було отримано наступні результати:

1. Запропоновано модифікацію алгоритму Ерлі для роботи з великими граматиками, використовуючи підхід «повернення назад» (back-tracking).
2. Запропоновано модифікацію алгоритму Ерлі для знаходження декількох дерев виводу послідовності для неоднозначних граматик.

3. Запропоновано коригуючий парсер на основі алгоритму Ерлі для виправлення помилок вставки, видалення, заміни терміналів та зміни порядку нетерміналів.
4. Експериментально досліджено роботу коригуючого парсера на простих та складних граматиках.
5. Зроблено порівняння коригуючого парсера з іншими системами виправлення граматичних помилок.

РОЗДІЛ 3. Модель на основі дерева залежностей

Головним завданням у цьому розділі є дослідження ефективності використання синтаксичної структури речення, як ключової ознаки при побудові моделей представлення семантики речень природної мови. Для експериментального дослідження ефективності побудованих моделей семантики речень була використана задача – визначення парафраз.

Парафразування (парафраз) − це процес, коли речення перефразовується або переписується з метою формування лексично іншого речення, яке має таке саме значення та зміст. Задача ідентифікації парафразу − це класична проблема класифікації у комп'ютерній лінгвістиці. Зазвичай, система отримує на вхід два речення і має визначити, чи мають вони однакове значення та зміст. Є ще одна пов'язана з нею задача, відома як визначення «семантичної подібності». У цьому випадку системі потрібно оцінити ступінь подібності двох речень.

Два речення, наведені нижче, є прикладом речень, які не є парафразами:

Yucaipa owned Dominick's before selling the chain to Safeway in 1998 for $2.5 billion.

Yucaipa bought Dominick's in 1995 for $693 million and sold it to Safeway for $1.8 billion in 1998.

Як видно з прикладу, просто розглядати набір слів, що містяться в обох реченнях, не є правильним підходом.

Дерево залежностей речення містить важливу інформацію про структуру речення. Дерево можна використовувати для розкладання великого речення на фрази. Можна спробувати зіставляти ці фрази між двома реченнями та використовувати нормалізовану кількість збігів, як важливу ознаку.

У роботі використано заздалегідь навчені векторні моделі слів, щоб відобразити семантику слів та їх подібність. Ці моделі можна розглядати, як основу для сучасних систем з оброблення природної мови. Кожне слово

представлено вектором фіксованої довжини і обчислення подібності слів зводиться до обчислення відстані між цими векторами.

3.1. Основна ідея моделі

В цьому розділі основна увага приділена підбору оптимального набору ознак, який би максимізував ефективність роботи моделі при класифікації парафраз. При цьому ключова роль відводилася саме синтаксичним ознакам.

Головним завданням моделі була спроба поєднати в моделі ознаки на рівні слів та дерева залежностей для представлення речення, подібно до [61,62]. Дерево характеризує структуру речення з точки зору слів та відповідних граматичних зв'язків між ними. Відношення між словами чітко типізовано.

Наведемо набір ознак, застосованих в експериментах для різних моделей.

Перше та друге речення позначатимуться відповідно s_1 та s_2.

3.1.1. Угорська вузлова збіжність

Речення розділяють на слова та обчислюють косинусну відстань для кожної пари векторів слів.

$$similarity(\vec{w}_1, \vec{w}_2) = \frac{\vec{w}_1 \vec{w}_2}{|\vec{w}_1||\vec{w}_2|}, \tag{3.1}$$

де $\vec{w}_1, \vec{w}_2$ це вектори слів з речень s_1 та s_2 відповідно.

На наступному кроці будуємо таблицю H таким чином:

$$H(i,j) = 1 - similarity(\vec{w}_i, \vec{w}_j), w_i \in s_1, w_j \in s_2 \tag{3.2}$$

Потім застосовуємо угорський алгоритм [63] до отриманої матриці і отримуємо набір вузлів, що співпали. Угорський алгоритм намагається поєднати слова з першого речення зі словами з другого, мінімізуючи вартість операції співставлення. В даному випадку ми використовуємо функцію подібності двох слів, щоб виразити штраф. Після цього відфільтровуємо неякісні збіги,

порівнюючи схожість слів із заданим пороговим значенням. Відсоток якісних збігів використовуємо надалі, як ознаку.

Розглянемо цю ознаку на прикладі двох речень:

$$Mariia\ enjoys\ swimming.$$
$$Mariia\ likes\ to\ swim.$$

Таблиця 3.1 Матриця схожості речень.

	ROOT	**Mariia**	**enjoys**	**swimming**	**.**
ROOT	1.0	0.00	0.00	0.00	0.00
Mariia	0.0	1.00	0.08	0.00	0.00
likes	0.0	0.00	0.57	0.21	0.17
to	0.0	0.00	0.00	0.15	0.27
swim	0.0	0.00	0.18	0.72	0.08
.	0.0	0.00	0.17	0.29	1.00

Таблиця 3.2 Матриця штрафів для Угорського алгоритму.

	ROOT	**Mariia**	**enjoys**	**swimming**	**.**
ROOT	0.0	1.0	1.00	1.00	1.00
Mariia	1.0	0.0	0.92	1.00	1.00
likes	1.0	1.0	0.43	0.79	0.83
to	1.0	1.0	1.00	0.85	0.73
swim	1.0	1.0	0.82	0.28	0.92
.	1.0	1.0	0.83	0.71	0.00

Таблиця 3.3 Таблиця співставлення елементів, базуючись на Угорському алгоритмі.

Елемент першого речення	Елемент другого речення	Схожість
ROOT	**ROOT**	1.0
Mariia	**Mariia**	1.0
likes	**enjoys**	0.57
swim	**swimming**	0.72
.	**.**	1.0

Якщо ми використаємо порогове значення $HungarianGraphNodesMatcherThreshold = 0.5$, можемо підрахувати відсоток якісних збігів:

$$\begin{gathered} HungarianAlgorithmMatches(H) = \{(w_i^1, w_j^2) \mid w_i^1 \in s_1, w_j^2 \in s_2\} \\ QualityHungarianAlgorithmMatches(H) = \{ \\ (w_i^1, w_j^2) \mid w_i^1 \in s_1, w_j^2 \in s_2, \\ similarity(w_i^1, w_j^2) > \\ HungarianGraphNodesMatcherThreshold \} \end{gathered} \tag{3.3}$$

$$QualityHungarianAlgorithmMatchesPercentage(s_1, s_2) = \frac{len(QualityHungarianAlgorithmMatches(H))}{len(s_1) + len(s_2)} \tag{3.4}$$

3.1.2. Коригуюча відстань між графами на основі угорського алгоритму

Як і на першому кроці використаємо вже розрахований збіг елементів речення , щоб позначити деякі слова однаковими у двох реченнях. Далі будуємо дерево залежностей та використовуємо алгоритм, запропонований у роботі [64] для обчислення відстані між графами. Даний алгоритм також базується на

угорському алгоритмі для обчислення мінімальної вартості редагування і використовує такі операції, як вставка, видалення та редагування.

Розглянемо цю ознаку на прикладі двох речень вказаних вище.

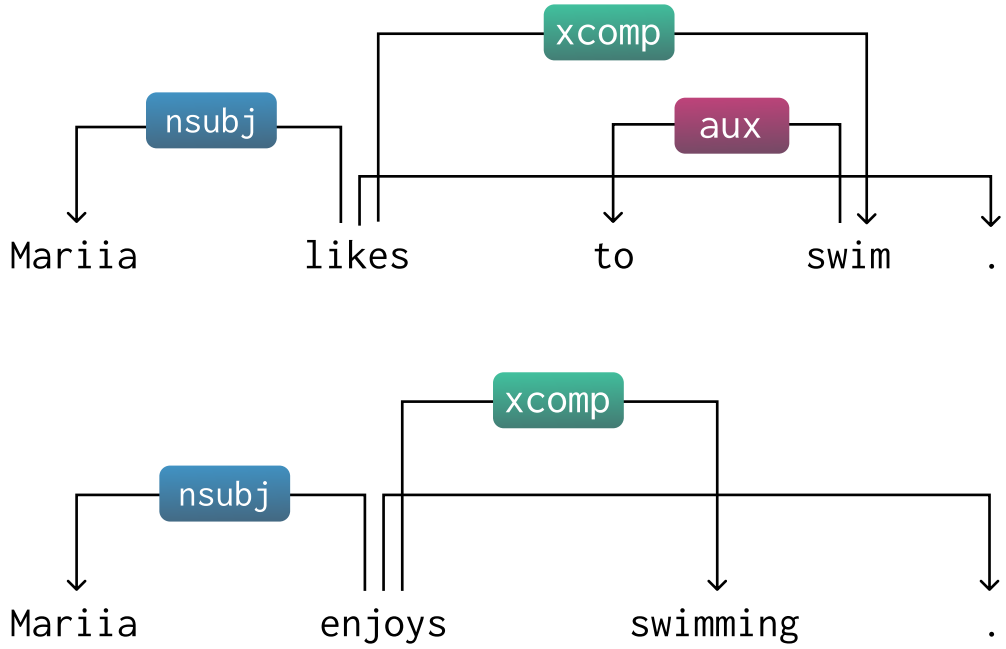


Рис. 3.1. Дерева залежностей для речень “Mariia enjoys swimming.” “Mariia likes to swim.”

Після того, як деякі вершини були визначені, як однакові, отримаємо такі графи:

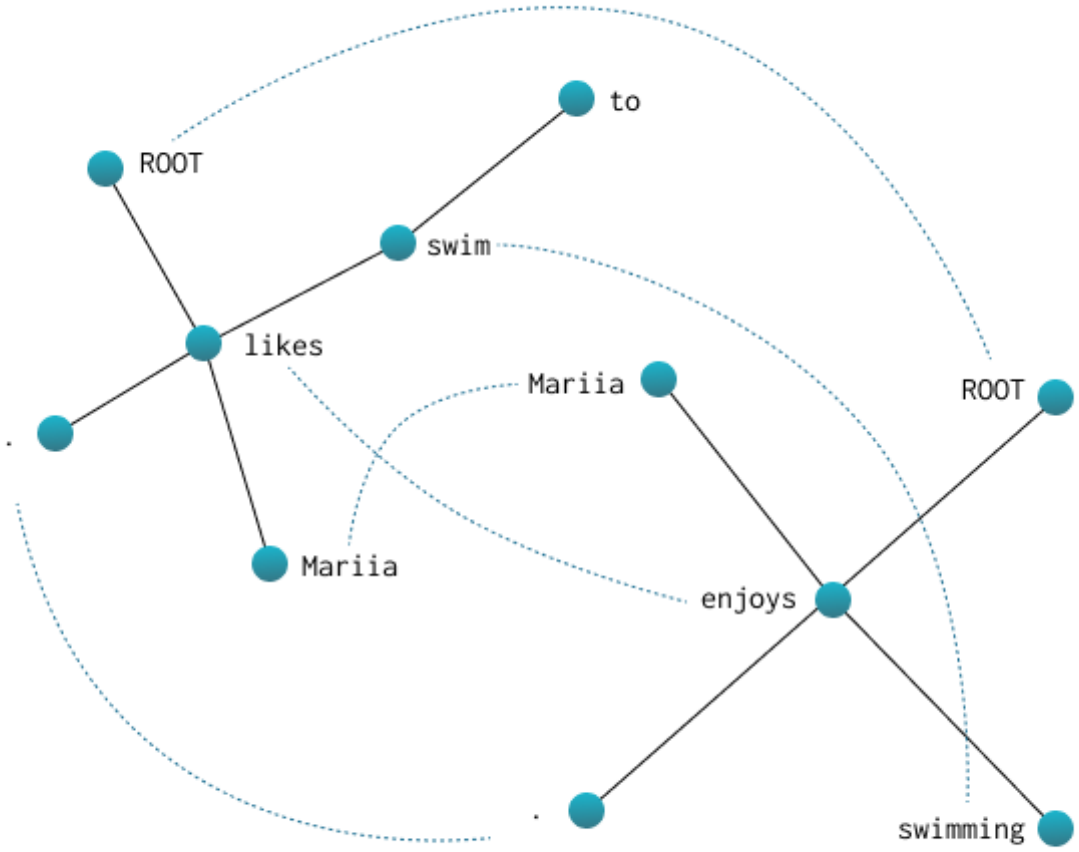


Рис. 3.2. Дерева залежностей із зв’язками схожості вершин

Далі утворимо таблицю штрафів для угорського алгоритму, використовуючи дані графи. Дана таблиця буде мати таку структуру:

Таблиця 3.4 Структура таблиці вартості для коригуючої відстані для графів

Таблиця вартості заміщення елементів.	Таблиця вартості вставки елементів.
Таблиця вартості видалення елементів.	Пуста таблиця.

Таблиця 3.5 Таблиці вартості для заміщення елементів.

	ROOT	**Mariia**	**enjoys**	**swimming**	**.**
ROOT	0.0	1.8	1.8	1.8	1.8
Mariia	1.8	0.0	1.5	1.5	1.5
likes	1.8	1.0	0.0	1.5	1.5
to	1.8	1.5	1.5	1.5	1.5
swim	1.67	1.67	1.67	0.33	1.67
.	1.8	1.5	1.5	1.5	0.00

Діагональ таблиці вставки та видалення заповнені 1 (штрафом) та всі інші елементи - ∞.

При підрахунку таблиці вартості для заміщення елементів береться до уваги значення вершини (чи однакові слова) та структура ребер у графі.

Коригуюча відстань даних графів складається з:

- Штрафу на заміну вершин ***swim*** та ***swimming*** (значення вершин вважаються однаковими, проте їх ребра відрізняються, ***swim*** має ребро до ***to***)
- Штраф вставки вершини **to.**

Отож, коригуюча відстань: $1 + 0.33 = 1.33$

3.1.3. Шлях у дереві залежностей

Для того, щоб обчислити цю ознаку, потрібно обійти дерево залежностей від кореня до листків і побудувати всі можливі шляхи заданої довжини.

$$getAllPaths(s, len) = \{[w_1, w_2, \dots w_{len}]\},$$
$$w_i \in s, [w_i, w_{i+1}] \in dependancyTreeEdges(s), \quad (3.5)$$
$$i \in \{1, \dots len - 1\}$$

Для всіх шляхів обчислюємо агреговане векторне представлення, використовуючи векторні моделі слів, присутніх у шляху.

$$pathEmbedding(path) = \frac{1}{len(path)} \sum_{w_i \in path} \vec{w}_i \tag{3.6}$$

Після того, як для кожного шляху у дереві було побудоване векторне представлення, їх можна легко порівняти між собою. Для побудови ознаки підрахуємо кількість шляхів, значення метрики схожості яких перевищує порогове значення.

$$\begin{gathered} DTPF(s_1, s_2, len) = \frac{2\left|\{(p_1, p_2)\}\right|}{PC_1 + PC_2}, \\ p_1 \in getAllPaths(s_1, len), \\ p_2 \in getAllPaths(s_2, len), \\ similarity\big(pathEmbedding(p_1), \quad pathEmbedding(p_2)\big) \geq \sigma. \\ PC_1 = |getAllPaths(s_1, len)|, \\ PC_2 = |getAllPaths(s_2, len)|, \end{gathered} \tag{3.7}$$

де σ – порогове значення схожості шляхів.

Для нормалізації цієї ознаки використовуємо кількість усіх шляхів в обох реченнях.

Таблиця 3.6 Всі шляхи у дереві залежностей довжини 2 для речень “Mariia enjoys swimming.” та “ Mariia likes to swim. ”.

Mariia enjoys swimming.	**Mariia likes to swim.**
enjoys Mariia	likes Mariia
enjoys swimming	likes swim
enjoys .	swim to
	likes .

3.1.4. Підграфи дерева залежностей

Для того, щоб обчислити цю ознаку, потрібно побудувати всі підграфи з фіксованою глибиною, використовуючи початкове дерево залежностей.

$$dependencyTree(s) = < V_s, E_s >$$
$$getAllSubgraphs(s, d) = \{g \mid g = < V_g, E_g >, V_g \subset V_s \,\&\, E_g \subset E_s \,\&\, depth(g) = d\} \quad (3.8)$$

Для всіх підграфів з цієї множини потрібно обчислити векторне представлення, використовуючи векторні моделі слів наявних у графі.

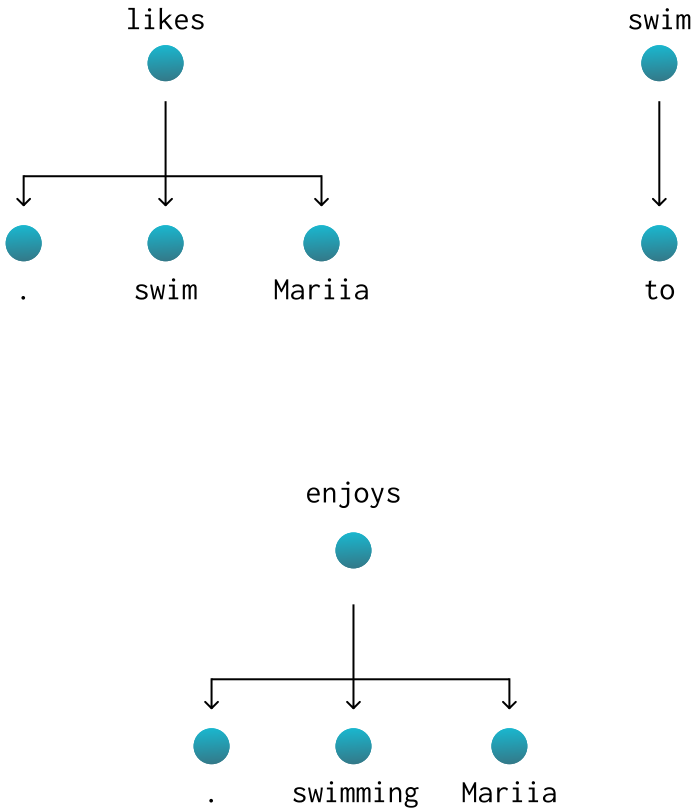
$$graphEmbedding(graph) = \frac{1}{|nodes(graph)|} \sum_{w_i \in nodes(graph)} \vec{w}_i \quad (3.9)$$

За допомогою векторного представлення графи можна порівнювати між собою. Знову використовується поріг схожості та враховується лише якісні збіги.

$$DTGF(s_1, s_2, depth) = \frac{2|\{g_1, g_2\}|}{GC_1 + GC_2},$$
$$g_1 \in getAllSubgraphs(s_1, depth),$$
$$g_2 \in getAllSubgraphs(s_2, depth),$$
$$similarity\big(graphEmbedding(g_1), \quad pathEmbedding(g_2)\big) \geq \zeta. \quad (3.10)$$
$$GC_1 = |getAllSubgraphs(s_1, len)|,$$
$$GC_2 = |getAllSubgraphs(s_2, len)|,$$

де ζ – порогове значення схожості графів.

Для того, щоб нормалізувати цю ознаку, використовується кількість усіх підграфів в обох реченнях.



.

Рис. 3.3. Всі підграфи глибини 2 для речень “Mariia enjoys swimming.” та “Mariia likes to swim.”

3.1.5. Підграфи дерева залежностей із зважуванням Idf

Idf означає обернену частоту документа. У нашому випадку будемо її обчислювати таким чином:

$$idf(w, D) = \log\left(\frac{N}{1 + |\{s \in D \mid w \in s\}|} + 1\right), \quad (3.11)$$

де D – множина речень всього корпусу, N – кількість речень в корпусі, $N = |D|$.

Ознака будується за алгоритмом описаним у попередньому пункті, проте при побудові векторного представлення графа idf слова використовується, як ваговий коефіцієнт.

3.1.6. Деякі інші ознаки

Основні ознаки довжини речення:

$$|s_1|, |s_2|, \big||s_1| - |s_2|\big| \tag{3.12}$$

Перетин n-грам:

$$NGO(s_1, s_2, n) = \frac{|NGram(s_1,n) \cap NGram(s_2,n)|}{|NGram(s_1,n)|}, \tag{3.13}$$

де $NGram(s, n)$ – множина n-грам довжини n речення s.

Перетин залежностей − порівняння ребер дерев залежностей.

$$DO(s_1, s_2) = \frac{|dependencyEdges(s_1) \cap dependencyEdges(s_2)|}{|dependencyEdges(s_1)|} \tag{3.14}$$

де $dependencyEdges(s) = \{(w_1, w_2)\}$, де w_1, w_2 зв'язані ребром в дереві залежностей.

Синтаксичний перетин n-грам [65].

Основна різниця між синтаксичними n-грамами та звичайними полягає в тому, які елементи вважаються сусідами. В звичайних n-грамах використовується порядок слів у речення, тоді як для синтаксичних n-грам використовується дерево залежностей. Слова вважаються сусідами, якщо вони зв'язані в дереві.

BLEU.

$$BLEU(s_1, s_2) = BP(s_1, s_2) exp\left[\sum_{n=1}^{N} \frac{1}{N} \log(p_n)\right], \tag{3.15}$$

$$BP(s_1, s_2) = exp\left[min\left[1 - \frac{|s_1|}{|s_2|}\right], 1\right], \tag{3.16}$$

$$p_n = \frac{\sum_{x \in NGram(s_1,n)} count\big(x, NGram(s_1, n) \cap NGram(s_2, n)\big)}{\sum_{x \in NGram(s_1,n)} count(x, NGram(s_1, n))}, \tag{3.17}$$

$$count(x, S) = |\{el \mid el \in S \,\&\, el = x\}|,$$

де N – максимальна довжина n-грами.

BP – коефіцієнт стислості, призначений для штрафування речень, якщо одне коротше за інше.

3.2. Експеримент

Для тестування було використано корпус MRPC. Цей корпус складається з 5800 пар речень, отриманих із новин в Інтернеті, а також анотацій, що вказують, чи є пари речень парафразами..

Для експериментів було вибрано набір класичних моделей, таких як SVM, логістична регресія, дерево рішень, Random Forest та інші, доступні в бібліотеці scikit-learn Python [66].

Використано попередньо навчену статистичну модель для англійської мови, розроблену Spacy [67] для синтаксичного аналізу речень та побудови дерева залежностей та векторних представлень слів. Дана модель була згенерована із застосуванням згорткової нейронної мережі, використовуючи корпус OntoNotes [68].

Для того, щоб оцінити ефективність застосування різних ознак, були проведені тести з різними класифікаторами. Результати цих тестів доступні в наведеній нижче таблиці.

Методика експерименту полягала в тому, що декілька класифікаторів тренувалися на навчальній вибірці з використанням різних наборів ознак та оцінювалися на тестовій вибірці. Основними метриками були обрані точність P та F_1.

$$F_1 = \frac{2 * (Recall * Precision)}{Recall + Precision} \quad (3.17)$$

В таблиці представлено класифікатори з найкращими F_1 показниками для кожної групи ознак.

Таблиця 3.7 Ефективність ознак

Ознака	**Класифікатор**	**Точність, P**	**F_1**
Угорська вузлова збіжність	SVM	73 %	82,3 %
Коригуюча відстань між графами на основі угорського алгоритму	SVM	72,4 %	81,9 %
Шлях у дереві залежностей	SVM	73,4 %	82,5 %
Підграфи дерева залежностей	SVM	73,6%	82,7 %
Підграфи дерева залежностей із зважуванням idf	SVM	73,0 %	82,4 %
Основні ознаки довжини речення + ознаки на основі n -грам	Ridge	74,3 %	82,4%
BLEU	SVM	73,0 %	82,3 %
Всі	SVM	75,4 %	83,6 %

Базуючись на проведених експериментах, найкращим класифікатором в переважній більшості випадків став метод опорних векторів SVM.

Ознаки на основі підграфів дерева залежностей виявилися найбільш ефективними серед розглянутих. Це доводить широкі можливості використання дерев підпорядкування в моделюванні семантики речень для задач визначення семантичної подібності.

Таблиця 3.8 Порівняння зі іншими моделями цього типу

Алгоритм	**Точність, P**	**F_1**
KM [69]	76,6 %	79,6 %
MCS [35]	70,3 %	81,3 %
ParaDetect [70]	74,7 %	81,8 %
Схожість на основі векторного представлення [61]	73,0 %	82,0 %
SDS [71]	73,0 %	82,3 %
SAMS-RecNN [72]	78,6 %	85,3 %
TF-KLD [73]	80,4 %	85,9 %
Наша система	75,4 %	83,6 %

Результати розробленої системи не перевершують рівень кращих сучасних моделей, проте впевнено входять у п’ятірку лідерів моделей свого класу [74]. Сама ж модель є дуже простою в реалізації.

3.3. Висновки до розділу 3

В результаті проведеного дослідження було проаналізовано використання дерева залежностей для ідентифікації парафраз. Головним висновком є експериментальне підтвердження того, що поєднання дерев залежностей та векторного представлення слів можна ефективно використовувати для побудови якісних моделей представлення семантики речень. В результаті було отримано наступні результати:

1. Розроблено та реалізовано кілька алгоритмів обходу та агрегування ознак із застосуванням дерева залежностей, зокрема обчислення підграфів та схожість шляхів у дереві.
2. В ході експериментів досліджено ефективність запропонованих ознак.
3. Створено модель, яку можна використовувати у прикладних системах.

РОЗДІЛ 4. Аналіз та використання моделей Трансформерів та LLM

4.1. Модель Трансформер

Архітектура моделі Трансформер була представлена у 2017 року у відомій статті – “Attention is all you need” [24]. У ній автори запропонували нову мережеву архітектуру, засновану виключно на механізмах уваги. Ці механізми дозволяють ефективно знаходити залежності між вхідною і вихідною послідовністю. Ще однією перевагою даної архітектури є можливість її паралелізації.

До появи Трансформерів більшість послідовність-до-послідовності (sequence-to-sequence) моделей базувалися на рекурсивних нейронних мережах в структурі енкодер-декодер (encoder-decoder). Ці моделі обробляють слова послідовно один за одним. Прихований шар моделі на кожній ітерації буде містити інформацію про останнє слово і коли на вхід подається довга послідовність, модель забуває інформацію про початок послідовності. Для того, аби розв’язати цю проблему був запропонований механізм уваги.

На етапі декодування декодер має можливість використовувати інформацію з кожного етапу енкодування. Механізм уваги надає можливість отримувати інформацію про усю послідовність у вигляді зваженої суми станів енкодера. Таким чином декодер намагається знайти, які слова з початкової послідовності є важливими при генерації вихідного слова. Цей підхід несе за собою важливе обмеження – вся вхідна послідовність має бути опрацьована від початку до кінця і немає можливості для паралельного виконання різних етапів.

Трансформер також має енкодер-декодер структуру, проте у ній не використовуються рекурсивні нейронні мережі для того, аби розпаралелити обрахунки й ефективніше використовувати наявні обчислювальні пристрої.

Перед тим, як описувати повну архітектуру, потрібно розглянути елементи, з якої вона складається.

4.1.1. Механізм Multi Head Self – Attention

Само-увага (self-attention) є механізмом уваги, що зв'язує різні елементи однієї послідовності, щоб обчислити представлення цієї послідовності в контексті певного слова.

Розглянемо, як це побудовано в Трансформері.

Кожний вектор x_i використовується в трьох ролях

1) Query – для того, аби отримати ваги уваги - w_{ij} для себе самого.

$$q_i = W_q\, x_i \tag{4.1}$$

2) Key - для того, аби отримати ваги уваги для всіх інших слів.

$$k_i = W_k\, x_i \tag{4.2}$$

3) Value – для того, аби вже маючи ваги уваги, підрахувати середнє зважене значення.

$$v_i = W_v\, x_i \tag{4.3}$$

Ваги уваги розраховуються за формулою

$$w'_{ij} = \frac{q_i^T k_j}{\sqrt{d_k}}, \tag{4.4}$$

$$w_{ij} = softmax\big(w'_{ij}\big), \tag{4.5}$$

$$y_i = \sum_j w_{ij} v_j \tag{4.6}$$

де d_k - це розмірність вектору ключів, y_i – контекст, який використовується для репрезентації послідовності відповідноx_i, W_q, W_k, W_v – це матриці, що дозволяють підрахувати для кожного вектору його значення query, key, value. Значення цих матриць шукаються під час тренування моделі.

$Softmax$ може бути досить чутливою до великих значень, а скалярний добуток збільшується зі збільшенням розмірності вектора. Для компенсації цього ефекту використовується нормування за допомогою $\sqrt{d_k}$. Даний механізм уваги називається авторами Scaled Dot-Product Attention.

Якщо розглядати q_i, k_i, v_i як частини матриць Q, K, V отримаємо таку формулу:

$$Attention\ (Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{4.7}$$

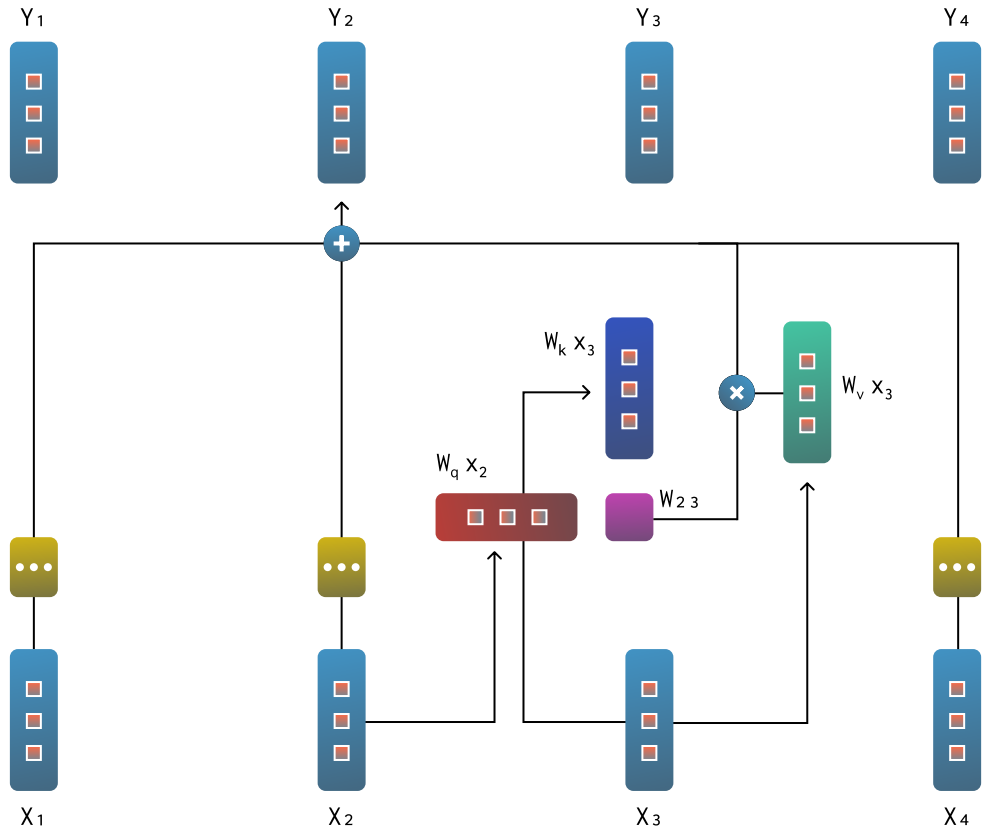


Рис. 4.1. Механізм уваги Scaled Dot-Product Attention (нормалізація не зображена) [75]

Замість того, аби використати лише Scaled Dot-Product Attention, автори запропонували спочатку лінійно спроектувати кожен q_i, k_i, v_i h разів у менші простори відповідно. В кожному просторі застосувати механізм уваги Scaled Dot-Product Attention, отримати h y_i, та використовуючи ще одне лінійне перетворення отримати фінальне значення y_i. Ці маніпуляції можна трактувати,

як надання можливості різним словам давати різну інформацію в механізмі уваги.

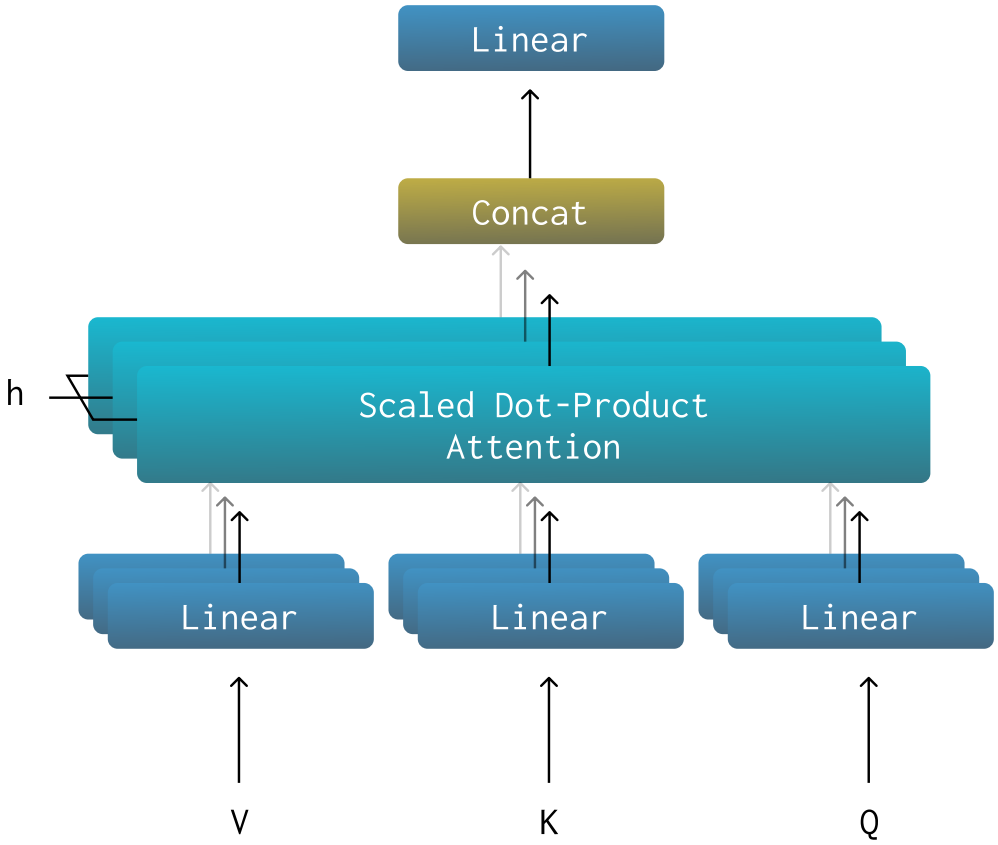


Рис. 4.2. Механізм уваги Multi Head Attention [24]

Таким чином фінальний механізм уваги можна описати:

$$Multi\ Head\ Attention\ (Q,K,V) = Concat(head_1, \dots head_h)W^O, \quad (4.8)$$

$$head_i = Attention\left(QW_i^Q, KW_i^K, VW_i^V\right), \quad (4.9)$$

де $W_i^Q \in R^{d_{model} \times d_k}, W_i^K \in R^{d_{model} \times d_k}, W_i^V \in R^{d_{model} \times d_v}, W_i^O \in R^{hd_v \times d_{model}}$.

4.1.2. Позиційне кодування

Трансформер не використовує рекурентних чи згорткових елементів, тому, якщо використовувати лише механізм уваги, результат роботи моделі не зміниться, навіть після перестановки слів. Це не є бажаним і для того, аби зафіксувати порядок, до кожного векторного представлення слова було додане його “позиційне представлення». Воно має таку ж розмірність як і вектор слова, таким чином дані вектори просто додаються. »

Для того, аби закодувати порядок, використано функції синус та косинус.

$$PositionalEncoding(pos, 2i) = PE(pos, 2i) = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right) \quad (4.10)$$

$$PositionalEncoding(pos, 2i+1) = PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right), \quad (4.11)$$

де pos – це позиція слова, а i – це позиція у векторному представленні.

4.1.3. Опис архітектури Трансформера

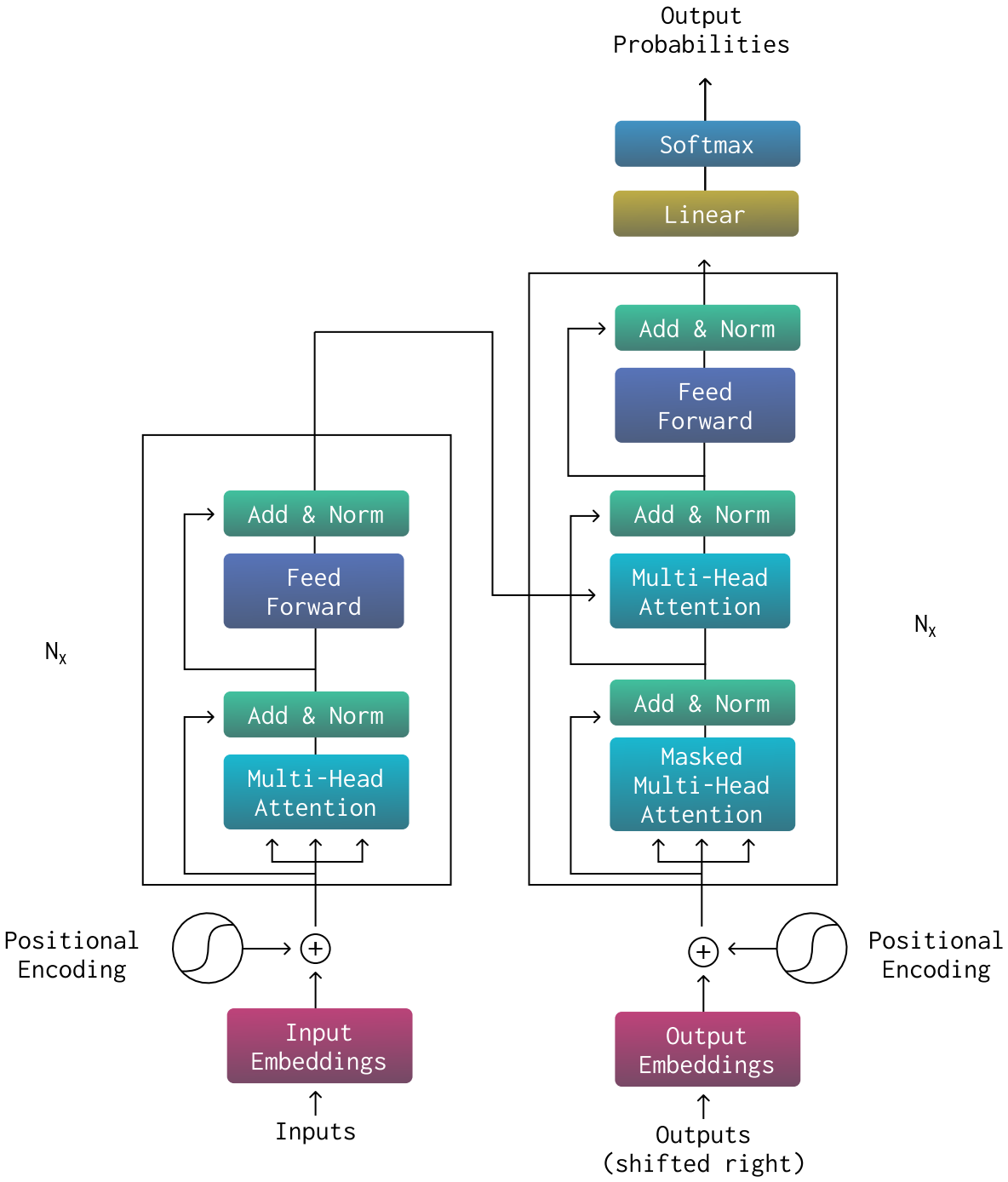


Рис. 4.3 Архітектура моделі Трансформер

Як вже було зазначено вище, архітектура Трансформера складається з двох логічних компонентів - енкодер та декодер. Кожен з них складається з N ідентичних внутрішніх шарів. В оригінальній статті $N = 6$.

Енкодер містить два підшари.

1) Multi-Head Self Attention
2) Повно зв'язна мережа прямого зв'язку, яка застосовується незалежно до кожного вектора.

Для того, щоб мати можливість тренувати моделі, які мають достатньо велику кількість параметрів, автори використали достатньо відомі на той час методи, такі як Residual Connections (покращують потік градієтів під час forward propagation) та Layer Normalization (покращує силу сигналу між шарами).

Декодер містить три підшари.

1) Масковану Multi-Head Self Attention.
2) Multi-Head Attention.
3) Повнозв'язна мережа прямого зв'язку, яка застосовується незалежно до кожного вектора.

У декодері теж додаються залишкові зв'язки та нормалізація.

В Трансформері механізми уваги застосовуються в декількох місцях:

- В шарі уваги енкодер-декодер value та keys надходять з енкодеру, а values з попереднього шару декодеру. Це дозволяє декодеру застосувати механізм уваги до кожної позиції енкодера.
- Енкодер містить шари з механізмом уваги Self - Attention. В цьому шарі всі query, keys, values надходять з попереднього шару енкодера. Кожна позиція в енкодері може застосувати механізм уваги до кожної позиції з попереднього шару.
- Декодер містить шари з механізм уваги Self - Attention. В цьому шарі всі query, keys, values надходять з попереднього шару декодера. Відмінність полягає в тому, що декодер не має можливості дивитися у «майбутнє» і

може використовувати для механізму уваги всі позиції до поточної. Це досягається за допомогою діагональних масок.

4.2. Огляд та аналіз різноманітних моделей, що базуються на Трансформерах

Для аналізу обрано кілька різних моделей, які базуються на ендкодері Трансформера. Саме енкодер намагається побудувати векторне представлення вхідної послідовності.

У цьому розділі розглянуто і представлено найпопулярніші ідеї, що лежать в основі кожної моделі. Ми почнемо з найвідомішої моделі Трансформера, а всі інші розташуємо за алфавітом.

4.2.1. Модель BERT

BERT [1] скорочено від «Bidirectional Encoder Representations from Transformers». BERT зосереджується на попередньому навчанні глибоких двонаправлених представлень речення**,** враховуючи як лівий, так і правий контекст. Його можна точно налаштувати, використовуючи один додатковий шар, відповідальний за вирішення конкретної задачі. Таким чином можна вирішувати багато задач, наприклад, класифікацію речень на декілька класів та інші без будь-яких змін до архітектури, пов’язаних із завданням.

Деталі: під час навчання цієї моделі її задачею були моделювання мови (визначення прихованого слова в реченні) та передбачення наступного речення. У зв’язку з цим додано сегментне векторне представлення (segment embeddings), яке використовується для того, аби відрізняти одне речення від іншого. Позиційне кодування здійснюється автоматично протягом навчання, на відміну від наперед заданого. Коли модель отримує на вхід два речення, механізм уваги використовує дані обох послідовностей.

4.2.2. Модель ALBERT

ALBERT [3] – скорочено від «A lite BERT». Її було створено, щоб подолати проблеми, які виникають під час збільшення розміру моделі: час навчання, обмеження пам’яті GPU/TPU та інші. Щоб розв’язати ці проблеми, автори запропонували методики зменшення кількості параметрів, щоб зменшити використання пам’яті та прискорити навчання BERT. Деталі: були використані повторювані шари, що призводить до зменшення обсягу пам’яті для збереження моделі (але витрати на обчислення залишаються незмінними).

4.2.3. Модель DistilBERT

DistilBERT [76] скорочено від «A distilled version of BERT». Ця модель намагається мінімізувати параметри великих попередньо навчених моделей, таких як BERT. Їхній підхід відрізняється від попередніх досліджень, оскільки вони намагалися застосувати методи дистиляції [77] під час фази попереднього навчання. Даний метод дозволяє створити меншу модель, за рахунок стиснення «знань» великої моделі. У результаті вони зменшили розмір на 40%, зберігши 97% можливостей розуміння мови в порівнянні з початковою моделлю. Деталі: автори вводять функцію потрійних втрат, яка поєднує мовне моделювання (language modeling), дистиляцію та косинусну відстань; його навчили передбачати точні ймовірності більшої моделі; під час навчання цієї моделі її задачею було передбачення точної ймовірності більшої оригінальної моделі.

4.2.4. Модель BART

BART [78] скорочено від «Bidirectional and Auto-Regressive Transformer». Це модель «послідовність-до-послідовності» (sequence-to-sequence), яка використовує для кодування архітектуру на основі Transformer, як-от BERT, але також включає декодер зліва направо, як-от GPT. Під час навчання цієї моделі її

задачею було виправлення штучно зіпсованого тексту. BART здатний як генерувати текст, так і розв’язувати задачі, що вимагають розуміння тексту, як от задача сумаризації (згенерувати скорочену версію вхідної текстової послідовності).

4.2.5. Модель ELECTRA

ELECTRA [79] скорочено від «Efficiently Learning an Encoder that Classifies Token Replacements Accurately». У цій роботі використовується новий підхід до тренування, а саме виявлення замінених токенів замість знаходження замаскованого слова, що використовується в BERT. Це нове завдання попереднього навчання є ефективним, оскільки воно працює над усіма вхідними токенами, а не лише з маскованою підмножиною. Підвищення ефективності надзвичайно помітне для менших моделей, наприклад, модель, яка навчалася на одному графічному процесорі протягом 4 днів, перевершує попередню на датасеті GLUE [80]. Деталі: в базову модель BERT не внесено жодних змін (ELECTRA — це підхід до навчання).

4.2.6. Модель MobileBERT

MobileBERT [81] — модель, заснована на моделі BERT, але стиснена і прискорена. Автори намагалися зменшити розмір великих моделей, аби зробити їх можливими для використання на мобільних пристроях. Отримана модель є універсальною, як і оригінальна BERT, і може бути застосована для розв’язання різних завдань шляхом тонкого налаштування (fine-tuning). Деталі: на основі моделі BERT_LARGE створено модель викладача та застосовано передачу знань за рахунок методу дистиляції.

4.2.7. Модель RoBERTa

RoBERTa [2] скорочено від «Robustly optimised BERT approach». Ця модель досліджує, як продуктивність оригінального BERT можна оптимізувати за допомогою більшої кількості даних і довшого часу навчання. Також пропонується змінити мету навчання – відкинути частину, яка намагається передбачити наступне речення. Їхня вдосконалена процедура попереднього навчання дозволила досягти найсучасніших результатів на момент публікації навіть без тонкого налаштування на різних задачах. Цей підхід також підкреслює важливість вибору гіперпараметрів і їх значний вплив на продуктивність. Деталі: RoBERTa має ту саму архітектуру, що й BERT, але використовує іншу схему попереднього навчання.

4.2.8. Модель I-BERT

I-BERT [82] скорочено від «Integer BERT». Ця модель намагається покращити моделі на основі Transformer, такі як BERT і RoBERTa, запроваджуючи новий підхід квантування, який допоможе зменшити використання таких ресурсів як пам’ять. Він заснований на зменшенні обсягу пам’яті за допомогою представлення нижчої бітової точності [83]. Автори зосереджуються на квантуванні, використовуючи лише цілі числа протягом усієї роботи моделі, усуваючи потребу в арифметиці з плаваючою комою. Цей підхід ґрунтується на цілочисельних наближеннях для нелінійних операцій, таких як GELU, SoftMax і Шару нормалізації. У результаті I-BERT досягає подібної або трохи вищої точності, ніж моделі, які використовують дійсні числа. Деталі: I-BERT має ту саму модель архітектури, що й BERT.

4.2.9. Модель DeBERTa

DeBERTa [84] скорочено від «Decoding-enhanced BERT with disentangled attention». Модель базується на RoBERTa та включає дві інноваційні методики. По-перше, вона використовує механізм розмежованої уваги, де кожне слово представлено двома векторами, один кодує його зміст, а інший його позицію. Вагові коефіцієнти уваги між словами обчислюються за допомогою роз'єднаних матриць на основі вмісту та відносних позицій. По-друге, модель використовує розширений декодер, який використовує також абсолютні позиції токенів, для передбачення замаскованих токенів під час попереднього навчання. Деталі: модель DeBERTa з 1,5 мільярдами параметрів перевершує продуктивність людини на SuperGLUE [85] тесті.

4.2.10. Модель SqueezeBERT

SqueezeBERT [86]— це модель, яка була побудована на основі SqueezeNet [87]– моделі комп'ютерного зору. SqueezeBERT — модель, яка базується на архітектурі Трансформера і є схожою на BERT. Автори намагаються застосувати методи, які зазвичай використовуються в моделях комп'ютерного зору, щоб мінімізувати розмір моделі та покращити її швидкість. Деталі: критична різниця між SqueezeBERT і BERT полягає в архітектурному виборі використання згрупованих згорток замість повністю зв'язаних шарів для шарів Q, K, V і FFN.

4.3. Порівняння ефективності моделей на основі Трансформерів для розв'язання задачі ідентифікації парафраз

Моделі на основі Трансформерів, описані вище, були проаналізовані, використовуючи інформацію про їх розмір, структуру та ефективність для ідентифікації парафраз.

Основні показники, які використовуватимуться для порівняння моделей:

- Точність і оцінка F1.
- Розмір моделі.
- Кількість речень, які модель може обробити за секунду.

Всі експерименти з моделями виконано за допомогою NVIDIA Tesla T4 із 16 ГБ оперативної пам’яті GPU та 50 ГБ системної оперативної пам’яті за допомогою платформи Google Collab [88].

Враховуючи обмеження обчислювальних ресурсів, було обмежено кількість моделей і використали такі експериментальні умови:

- Кожна модель була тонко налаштована (fine-tunned) лише протягом 5 епох.
- Були використані однакові глобальні параметри для точного налаштування (fine-tunning) всіх моделей.
- Кожна модель була тонко налаштована п’ять разів і в таблицях результатів повідомлено середнє значення та стандартне відхилення. Це було зроблено для того, щоб показати, наскільки стабільною є модель.

Важливо відзначити, що деякі з моделей в оригінальних статтях досягли кращих результатів, ніж зазначено тут. Є кілька причин, чому це сталося:

- Кожна модель має різні версії і залежать від кількості кінцевих параметрів, які впливають на розмір моделі, розміру словника, навчального набору даних та інших параметрів. У більшості випадків найкращі результати було показано за допомогою великих моделей, які ми не використовуємо через обмеження обчислювальних ресурсів.
- Кожну модель завжди можна спробувати оптимізувати за допомогою методів підбору гіперпараметрів. Ми не намагалися робити таку оптимізацію у даному експерименті через високу вартість цього процесу.

Ми використали бібліотеку Hugging Face [89] для доступу до існуючих моделей та тонкого налаштування цих моделей на корпусі MRPC [25].

Таблиця 4.1 Список моделей та їх версій в бібліотеці Hugging Face

N	Назва моделі	Назва моделі в бібліотеці Hugging Face
1	ALBERT base	albert-base-v2
2	ALBERT large	albert-large-v2
3	BERT base uncased	bert-base-uncased
4	BERT base cased	bert-base-cased
5	BERT large uncased	bert-large-uncased
6	BERT large cased	bert-large-cased
7	DistilBERT base	distilbert-base-uncased
8	Facebook BART base	facebook/bart-base
9	Google Electra base discriminator	google/electra-base-discriminator
10	Google Electra small discriminator	google/electra-small-discriminator
11	Google MobileBERT	google/mobilebert-uncased
12	I-BERT RoBERTa base	kssteven/ibert-roberta-base
13	Microsoft DeBERTa base	microsoft/deberta-base
14	Microsoft DeBERTa large	microsoft/deberta-large
15	RoBERTa base	roberta-base
16	RoBERTa large	roberta-large
17	SqueezeBERT	squeezebert/squeezebert-mnli-headless

Усі моделі, перелічені вище, побудовані з використанням архітектури Трансформерів, але вони мають різні конфігурації з точки зору розміру, шарів, розміру вбудованих елементів і розміру словника.

Таблиця 4.2 Список моделей та їх характеристики

N	Назва моделі	Кількість параметрів	Кількість шарів	Розмір прихованого шару	Розмір векторних представлень	Розмір словника
1	ALBERT base	11M	12	768	128	30 тис
2	ALBERT large	17M	24	1024	128	30 тис
3	BERT base uncased	110M	12	768	768	30 тис
4	BERT base cased	110M	12	768	768	30 тис
5	BERT large uncased	340M	24	1024	1024	30 тис
6	BERT large cased	340M	24	1024	1024	30 тис
7	DistilBERT base	67M	6	768	768	30 тис
8	Facebook BART base	140M	12	768	768	50 тис
9	Google Electra base discriminator	109M	12	768	768	30 тис
10	Google Electra small discriminator	13.5M	12	256	128	30 тис
11	Google MobileBERT	24M	24	512	128	30 тис
12	I-BERT RoBERTa base	124M	12	768	768	50 тис
13	Microsoft DeBERTa base	140M	12	768	768	50 тис
14	Microsoft DeBERTa large	400M	24	1024	1024	50 тис
15	RoBERTa base	125M	12	768	768	50 тис
16	RoBERTa large	355M	24	1024	1024	50 тис
17	SqueezeBERT	51M	12	768	768	30 тис

Ми можемо згрупувати моделі за кількістю параметрів у три групи – малі, звичайні та великі.

- Малі моделі: ALBERT base, ALBERT large, DistilBERT, Google Electra small discriminator, Google MobileBERT, SqueezeBERT.
- Звичайні моделі: BERT base uncased, Facebook BART base, Google Electra base discriminator, I-BERT RoBERTa base, Microsoft DeBERTa base, RoBERTa base.
- Великі моделі: BERT large uncased, BERT large cased, Microsoft DeBERTa large, RoBERTa large.

Як згадувалося вище, усі моделі були налаштовані з такими загальними параметрами:

- Розмір батча = 32.
- Тонке налаштування (fine-tunning) протягом 5 епох.
- Швидкість навчання $lr = 2 \times 10^{-5}$.
- Weight decay = 0,01.
- Стратегія оптимізації AdamW (PyTorch).

Таблиця 4.3 Список малих моделей та їх результатів під час тренування та валідації

N	Назва моделі	Точність на тренуванні	F1 на тренуванні	Точність на валідації	F1 на валідації
1	ALBERT base	93.92 ± 3.52	95.45 ± 2.67	88.53 ± 0.73	91.72 ± 0.49
2	ALBERT large	**95.56 ± 2.25**	**96.71 ± 1.66**	**88.73 ± 1.26**	**91.89 ± 0.92**
3	DistilBERT base	89.81 ± 0.35	92.39 ± 0.25	83.82 ± 0.99	88.66 ± 0.72
4	Google Electra small discriminator	85.09 ± 0.90	89.07 ± 0.68	83.38 ± 0.75	88.38 ± 0.43
5	Google MobileBERT	75.56 ± 5.03	82.85 ± 3.49	74.75 ± 5.58	82.62 ± 4.24
6	SqueezeBERT	93.02 ± 1.11	94.75 ± 0.84	88.19 ± 0.18	91.31 ± 0.15

Таблиця 4.4 Список малих моделей та їх результатів під час тестування

N	Назва моделі	Точність на тестуванні	F1 на тестуванні	Кількість класифікованих речень за секунду
1	ALBERT base	84.87 ± 1.30	88.64 ± 1.19	164.88 ± 0.59
2	ALBERT large	**86.50 ± 2.32**	**89.93 ± 1.68**	61.8 ± 0.03
3	DistilBERT base	81.25 ± 0.55	86.19 ± 0.46	438.70 ± 3.01
4	Google Electra small discriminator	81.47 ± 0.58	86.50 ± 0.38	**1242.70 ± 0.39**
5	Google MobileBERT	72.44 ± 4.74	80.62 ± 3.71	526.57 ± 0.28
6	SqueezeBERT	84.58 ± 0.49	88.22 ± 0.40	419.06 ± 0.6

Можемо спостерігати, що для малих моделей ALBERT має найкращу точність і показник F1. Ми помістили ALBERT base та ALBERT large в категорію малих моделей через невеликий розмір. Вони займають перші два місця і мають найменший розмір – всього 11 та 17 мільйонів параметрів. При цьому ALBERT є найповільнішим, що очікувано, оскільки автори ALBERT намагалися оптимізувати розмір і час навчання, а не швидкість.

Найшвидшою моделлю є малий дискримінатор Google Electra, яка має теж досить непогані результати, але з більше ніж в 7 разів кращою швидкістю.

Таблиця 4.5 Список середніх моделей та їх результатів під час тренування та валідації

N	Назва моделі	Точність на тренуванні	F1 на тренуванні	Точність на валідації	F1 на валідації
1	BERT base uncased	92.19 ± 1.34	94.19 ± 0.97	82.65 ± 1.43	87.96 ± 0.83
2	BERT base cased	**97.24 ± 1.77**	**97.95 ± 1.3**	82.99 ± 1.99	88.34 ± 1.18
3	Facebook BART base	93.32 ± 1.25	95.10 ± 0.89	86.76 ± 0.78	90.58 ± 0.68
4	Google Electra base discriminator	94.38 ± 0.66	95.84 ± 0.51	**88.24 ± 0.64**	**91.62 ± 0.42**
5	I-BERT RoBERTa base	92.55 ± 2.58	94.39 ± 2.00	87.79 ± 1.08	91.07 ± 0.85
6	Microsoft DeBERTa base	93.65 ± 1.70	95.25 ±1.26	**88.33 ± 1.17**	**91.63 ± 0.87**
7	RoBERTa base	92.67 ± 1.14	94.51 ± 0.88	88.19 ± 0.39	91.40 ± 0.25

Таблиця 4.6 Список середніх моделей та їх результатів під час тестування

N	Назва моделі	Точність на тестуванні	F1 на тестуванні	Кількість класифікованих речень за секунду
1	BERT base uncased	80.12 ± 2.05	85.50 ± 1.37	213.91 ± 3.09
2	BERT base cased	80.45 ± 2.37	85.92 ± 1.47	219.25 ± 2.15
3	Facebook BART base	85.92 ± 0.21	89.74 ± 0.18	156.89 ± 5.03
4	Google Electra base discriminator	85.31 ± 0.47	89.24 ± 0.38	207.08 ± 8.96
5	I-BERT RoBERTa base	**86.23 ± 0.50**	**89.63 ± 0.52**	220.21 ± 0.91
6	Microsoft DeBERTa base	**86.40 ± 0.48**	**89.86 ± 0.34**	169.76 ± 0.67
7	RoBERTa base	85.84 ± 0.52	89.46 ± 0.41	**226.28 ± 2.61**

Ми бачимо, що Microsoft DeBERTa має найкращу точність і показник F1 для моделей середнього розміру. Ця модель одна з найбільших у своїй категорії – 140 мільйонів параметрів. При цьому Microsoft DeBERTa є однією з найповільніших.

Найшвидшою моделлю є RoBERTa base, але, базуючись на даних експерименту, можна спостерігати, що більшість моделей середнього розміру працюють приблизно однаково з точки зору кількості класифікованих речень за секунду.

Важливо порівняти показники BERT base uncased та BERT base cased, що відрізняються лише даними, на яких тренувалися моделі. Одна модель під час тренування використовувала корпус, всі слова якого були модифіковані таким чином, що великі літери були замінені на відповідні малі літери, інший використовував корпус без змін. Можемо спостерігати, що модель, яка використовувала оригінальний корпус має кращі показники: приблизно на половину процентного пункту.

Таблиця 4.7 Список великих моделей та їх результатів під час тренування та валідації

N	Назва моделі	Точність на тренуванні	F1 на тренуванні	Точність на валідації	F1 на валідації
1	BERT large uncased	98.58 ± 0.43	98.95 ± 0.32	85.88 ± 0.59	90.09 ± 0.48
2	BERT large cased	98.22 ± 1.33	98.68 ± 0.99	86.86 ± 1.76	90.60 ± 1.40
3	Microsoft DeBERTa large	**98.58 ± 1.36**	**98.94 ± 1.03**	89.90 ± 0.75	92.70 ± 0.56
4	RoBERTa large	97.72 ± 2.07	98.30 ± 1.55	**90.15 ± 0.72**	**92.90 ± 0.51**

Таблиця 4.8 Список великих моделей та їх результатів під час тестування

N	Назва моделі	Точність на тестуванні	F1 на тестуванні	Кількість класифікованих речень за секунду
1	BERT large uncased	84.31 ± 0.17	88.63 ± 0.22	62.70 ± 0.2
2	BERT large cased	82.46 ± 0.96	87.00 ± 0.79	69.28 ± 0.8
3	Microsoft DeBERTa large	**88.79 ± 0.37**	**91.61 ± 0.39**	56.10 ± 0.3
4	RoBERTa large	88.42 ± 0.70	91.38 ± 0.52	**71.02 ± 0.2**

Серед великих моделей Microsoft DeBERTa large знову має найкращу точність і показник F1. Ця модель також найбільша у своїй категорії – 400 мільйонів параметрів. При цьому Microsoft DeBERTa є найповільнішою.

Найшвидшою моделлю є RoBERTa large, але як і в попередній категорії середня швидкість моделей є приблизно однаковою.

Важливо відзначити, що більші моделі з такою ж архітектурою дають кращі результати – у цьому випадку середня модель Google Electra краща за меншу версію на чотири відсоткові пункти. Менша модель має майже у вісім разів менше параметрів, ніж звичайна, але працює в п’ять разів швидше. Microsoft DeBERTa large показала себе краще за Microsoft DeBERTa base на два відсоткові пункти, схожа ситуація з RoBERTa та BERT.

4.4. Модель Трансформера з використанням дерева залежностей

Моделі на основі архітектури Трансформер досягають хороших результатів на різноманітних задачах, як показано вище. Проведено багато досліджень як можна використовуючи їхню структуру для того, аби покращити їх точність. Збагачення BERT та подібних моделей додатковою інформацією може підвищити їхню продуктивність і адаптивність до різних завдань.

Це можна пояснити тим, що для деяких завдань може знадобитися інформація, пов’язана з областю або завданням, яка не є представленою в тренувальному корпусі текстів. Доповнення моделі додатковою інформацією,

такою як метадані, числові дані або доменно-спеціальні вбудовування, може покращити його продуктивність у цих завданнях.

Щоб модель краще розуміла контекст та структуру речення, часто використовується синтаксична інформація, теги частин мови або лінгвістичні структури. Це може забезпечити додатковий контекст, покращуючи розуміння моделлю вхідного тексту.

Збагачуючи модель додатковою інформацією, вона отримує ширший контекст, підвищену адаптивність і покращену продуктивність у широкому спектрі завдань, що робить її більш універсальною та здатною в різних програмах для розуміння та обробки природної мови.

Наприклад, SyntaxBERT [90] — це варіант BERT, спеціально розроблений для захоплення синтаксичної інформації в тексті. На відміну від традиційних моделей BERT, які зосереджені на контекстному розумінні, SyntaxBERT робить акцент на розумінні структурних залежностей і граматичних зв’язків між словами в реченнях. Він використовує методи синтаксичного аналізу, щоб покращити свою здатність розуміти та представляти ці зв’язки. Це дозволяє більш детально розуміти структуру речень.

Включаючи синтаксичну інформацію в процес навчання, SyntaxBERT прагне покращити завдання, які значною мірою покладаються на граматичні структури, такі як розбір, машинний переклад і відповіді на запитання. Ця модель намагається вловити не лише семантичне значення, але й синтаксичні нюанси, присутні в природній мові, допомагаючи таким чином у завданнях, які потребують глибшого розуміння композиції та структури речення.

Ми пропонуємо використати дерево залежностей та зв’язки між словами у ньому для того та використати цю інформацію в шарі self – attention.

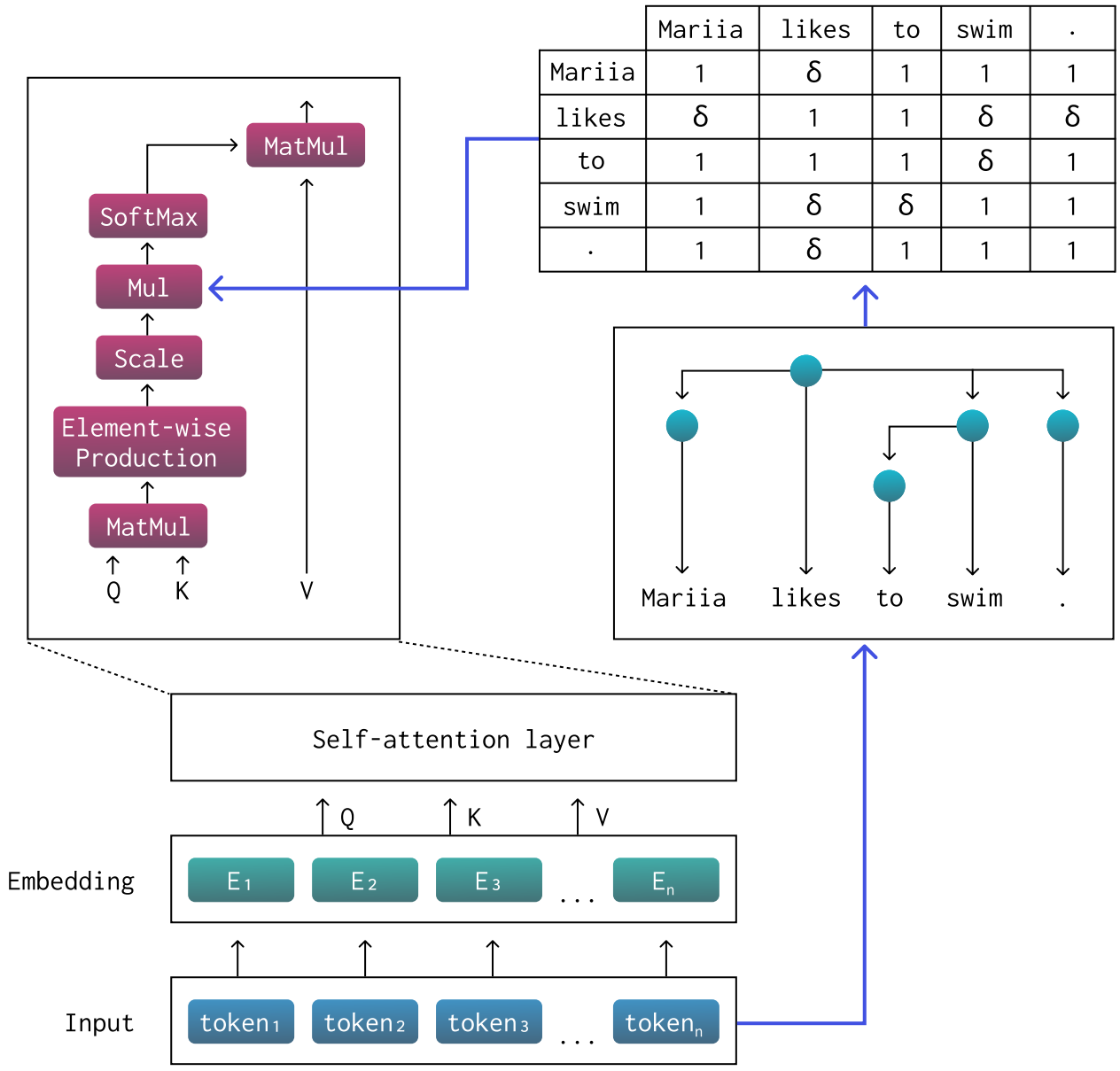


Рис. 4.4 Архітектура моделі Трансформер доповнена інформацією дерева залежностей

Для вхідної послідовності $s = (s_1, s_2 \dots s_n)$ визначимо граф $< V_s, E_s >$ та матрицю A.

$$dependencyTree(s) = < V_s, E_s > \tag{4.12}$$

$$a_{ij} = \begin{cases} \delta, & (s_i, s_j) \in E_s \vee (s_j, s_i) \in E_s \\ 1, & otherwise \end{cases} \tag{4.13}$$

Коефіцієнт δ – один з гіперпараметрів моделі та буде підібраний під час її оптимізації

Оновлений шар self – attention:

$$q_i = W_q\, x_i \tag{4.14}$$

$$k_i = W_k\, x_i \tag{4.15}$$

$$v_i = W_v\, x_i \tag{4.16}$$

$$w'_{ij} = \frac{q_i^T k_j}{\sqrt{d_k}}, \tag{4.17}$$

$$w^*_{ij} = w'_{ij} * a_{ij}, \tag{4.18}$$

$$w_{ij} = softmax\big(w^*_{ij}\big), \tag{4.19}$$

$$y_i = \sum_j w_{ij} v_j \tag{4.20}$$

де d_k - це розмірність вектору ключів, y_i – контекст, який використовується для репрезентації послідовності відповідно x_i, W_q, W_k, W_v – це матриці, що дозволяють підрахувати для кожного вектору його значення query, key, value.

4.5. Особливості додавання нових ознак до моделі

Щоб коректно закодувати дерево залежностей, потрібно брати до уваги особливості різних стратегій токенізації в моделях з архітектурою Трансформер.

Моделі типу BERT використовують токенізатор типу Word Piece (WPT). Цей токенізатор розбиває слова на одиниці, які називаються токенами. Таким чином відбувається сегментація слова на менші одиниці, що допомагає ефективніше обробляти рідкісні слова, невідомі слова та морфологічно складні

мови. Наприклад, таке слово, як “singing”, можна розділити на “sing” і “##ing”, де “##” означає, що під слово є частиною більшого слова.

Також використовуються спеціальні токен-маркери, щоб допомогти зрозуміти структуру послідовності:

- [CLS] – маркер класифікації. Він передує вхідному тексту та використовується для класифікаційних завдань.
- [SEP] – токен розділювач. Він розділяє два різні речення або фрази в одному введеному тексті.
- [MASK] – маркер маски. Він використовується під час попереднього навчання для завдань моделювання замаскованої мови, де деякі лексеми випадково замінюються на [MASK], щоб передбачити оригінальні лексеми.

Токенізатори використовують фіксований словник, отриманий із навчальних даних. Кожен токен у вхідному тексті зіставляється з вектором із цього словника.

Під час обробки двох речень одночасно (у таких завданнях, як ідентифікація парафраз), моделі типу BERT використовують ідентифікатори сегментів, щоб розрізняти речення. Кожній лексемі присвоюється ідентифікатор сегмента, щоб вказати відповідне речення.

Оскільки модель під час тренування працює зазвичай з певним набором вхідних даних одночасно (batch), потрібно стандартизувати вхідні послідовності до фіксованої довжини. [PAD] - маркери заповнення, додаються до послідовностей, які коротші за максимальну довжину, тоді як довші послідовності можуть бути скорочені, щоб відповідати максимальному розміру вхідних даних моделі. Для того, аби модель могла розрізняти справжні вхідні дані, від заповнення використовується маска уваги.

Токенізатори відіграють важливу роль у тому, як моделі типу BERT обробляють та розуміють текст, дозволяючи моделі ефективно обробляти вхідні

дані. Під час додавання нових ознак до моделі потрібно брати до уваги всі особливості токенізації, щоб правильно представити нові ознаки.

Розглянемо два приклади того, як можна додати нові ознаки до моделі типу BERT. Надалі, що на вхід модель отримує два речення (як для задачі ідентифікації парафраз).

Вхідні речення: “Mariia likes to swim.” “Mariia enjoys swimming.”

Базова токенізація моделі типу BERT для цих двох речень (будемо використовувати токенайзер моделі BERT Base Cased).

Таблиця 4.9 Базова токенізація речень “Mariia likes to swim.” та “Mariia enjoys swimming.”

Слово	Токен	Токен ID	Сегмент	Маска уваги
	[CLS]	101	0	1
Mariia	Mari	17978	0	1
	##ia	1465	0	1
likes	likes	7407	0	1
to	to	1106	0	1
swim	swim	11231	0	1
.	.	119	0	1
	[SEP]	102	0	1
Mariia	Mari	17978	1	1
	##ia	1465	1	1
enjoys	enjoys	16615	1	1
swimming	swimming	5947	1	1
.	.	119	1	1
	[SEP]	102	1	1
	[PAD]	0	0	0
	[PAD]	0	0	0

1) Ознака «Частина Мови» (POS tag)

Спочатку потрібно визначити для кожного слова його частину мови. Для цього можна використати бібліотеку Spacy [67].

Таблиця 4.10 Визначення частин мови для кожного слова речень “Mariia likes to swim.” та “Mariia enjoys swimming.”

Речення 1	Частина мови	Речення 2	Частина мови
Mariia	NNP	Mariia	NNP
likes	VBZ	enjoys	VBZ
to	TO	swimming	VBG
swim	VB	.	.
.	.		

При зіставленні частин мови з токенами потрібно брати до уваги, що різні програмні продукти можуть по різному робити токенізацію. Один токен моделі А може відповідати декільком токенам моделі В. У нашому випадку це явно спостерігається зі словом “Mariia”, яке було розбито на “Mari” та “##ia”.

Для того, аби коректно поєднати ознаки, ми використовуємо додаткові метадані, які токенізатори можуть надавати, а саме індекс початку та кінця кожного токена у вхідній послідовності. Використовуючи дану інформацію, можна однозначно зіставити токени однієї моделі з токенами іншої.

Важливо підмітити, що на додачу до звичайних POS тегів, ми використовуємо два додаткових – NA, UNKNOWN. NA використовується для того, аби позначати спеціальні токени моделі ([CLS], [SEP]). UNKNOWN використовується у тому випадку, коли модель не змогла визначити частину мови, або визначена частина мови не підтримується.

Для того, аби використовувати POS теги вхідної послідовності, потрібно побудувати векторне представлення для кожного з них і поєднати з векторним

представленням самого токена. Таким чином, фінальне представлення буде складатися з представлення токена, його позиції в реченні та частини мови.

Таблиця 4.11 Токенізація речень “Mariia likes to swim.” та “Mariia enjoys swimming.”, використовуючи інформацію про частини мови.

Слово	Токен	Токен ID	POS Tag	POS Tag ID
	[CLS]	101	NA	19
Mariia	Mari	17978	NNP	21
	##ia	1465	NNP	21
likes	likes	7407	VBZ	37
to	to	1106	TO	29
swim	swim	11231	VB	32
.	.	119	.	6
	[SEP]	102	NA	19
Mariia	Mari	17978	NNP	21
	##ia	1465	NNP	21
enjoys	enjoys	16615	VBZ	37
swimming	swimming	5947	VBG	34
.	.	119	.	6
	[SEP]	102	NA	19
	[PAD]	0		0
	[PAD]	0		0

2) Ознака на основі дерева залежностей

Спочатку потрібно для кожного речення побудувати свій граф залежностей. Для цього можна використати бібліотеку Spacy [67].

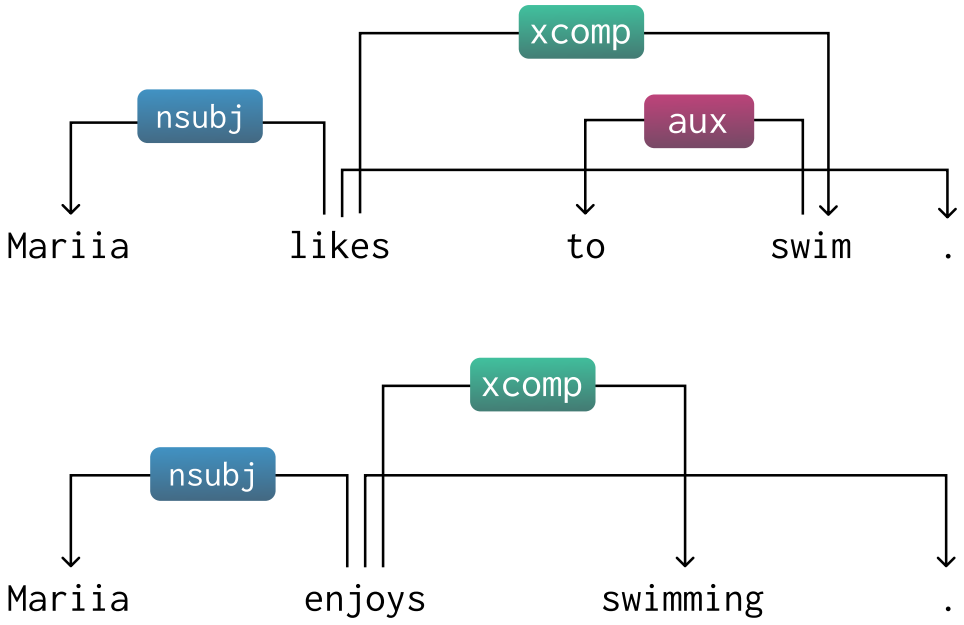


Рис. 4.5. Дерева залежностей для речень “Mariia enjoys swimming.” “Mariia likes to swim.”

Базуючись на цьому графі побудуємо матриці зв’язності графів (знехтуємо напрямом для спрощення моделі).

Таблиця 4.12 Матриця зв’язності речення “Mariia likes to swim.”

Слово \ Слово	Mariia	likes	to	swim	.
Mariia	0	1	0	0	0
likes	1	0	0	1	1
to	0	0	0	1	0
swim	0	1	1	0	0
.	0	1	0	0	0

Таблиця 4.13 Матриця зв’язності речення “Mariia enjoys swimming.”

Слово \ Слово	Mariia	enjoys	swimming	.
Mariia	0	1	0	0
enjoys	1	0	1	1
swimming	0	1	0	0
.	0	1	0	0

Маючи дані графи, можна побудувати повну матрицю відповідно до формул (4.12) та (4.13), враховуючи особливості токенізації.

Таблиця 4.14 Матриця ознаки на основі дерева залежностей для речень “Mariia likes to swim.” та “Mariia enjoys swimming.”

	[CLS]	Mariia likes to swim.						[SEP]	Mariia enjoys swimming.					[SEP]		
	101	17978	1465	7407	1106	11231	119	102	17978	1465	16615	5947	119	102	0	0
101	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
17978	1	1	1	*δ*	1	1	1	1	1	1	1	1	1	1	0	0
1465	1	1	1	*δ*	1	1	1	1	1	1	1	1	1	1	0	0
7407	1	*δ*	*δ*	1	1	*δ*	*δ*	1	1	1	1	1	1	1	0	0
1106	1	1	1	1	1	*δ*	1	1	1	1	1	1	1	1	0	0
11231	1	1	1	*δ*	*δ*	1	1	1	1	1	1	1	1	1	0	0
119	1	1	1	*δ*	1	1	1	1	1	1	1	1	1	1	0	0
102	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
17978	1	1	1	1	1	1	1	1	1	1	*δ*	1	1	1	0	0
1465	1	1	1	1	1	1	1	1	1	1	*δ*	1	1	1	0	0
16615	1	1	1	1	1	1	1	1	*δ*	*δ*	1	*δ*	*δ*	1	0	0
5947	1	1	1	1	1	1	1	1	1	1	*δ*	1	1	1	0	0
119	1	1	1	1	1	1	1	1	1	1	*δ*	1	1	1	0	0
102	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

При побудові даної матриці було враховано можливе розбиття одного слова на декілька токенів, службові токени та токени заповнення.

4.6. Результати експерименту

В цьому експерименті порівняємо доповнену модель Трансформера із ознаками дерева залежностей з базовою моделлю.

- DeBERTa Base
- DeBERTa Base + Dependency Tree

Модель DeBERTa була обрана, базуючись на результатах експерименту, описаного в підрозділі 4.3.

Основні показники, які використовуватимуться для порівняння моделей:

- Точність і оцінка F1.
- Розмір моделі.

Усе тонке налаштування для моделей, описаних у цьому розділі, виконано за допомогою NVIDIA Tesla V100 із 16 ГБ оперативної пам’яті GPU та 50 ГБ системної оперативної пам’яті за допомогою платформи Google Collab [88].

Умови експерименту:

- Кожна модель тонко налаштована лише протягом 10 епох.
- Використані загальні глобальні параметри для точного налаштування всіх моделей.
- Кожна модель тонко налаштована три рази і в таблицях результатів повідомлено середнє значення та стандартне відхилення.

Усі моделі були налаштовані з такими загальними параметрами:

- Розмір батча = 8.
- Тонке налаштування (fine-tunning) протягом 10 епох.
- Швидкість навчання lr =2×10^{-5}, лінійне зменшення протягом кожного кроку.
- Weight decay = 0,01.

- Стратегія оптимізації AdamW (PyTorch).

Враховуємо, що для моделі, яка використовує дерево залежності, потрібно спочатку підібрати параметр δ. Для цього будемо використовувати дані з корпусу MRPC (validation part).

Таблиця 4.15 Результати моделі DeBERTa Base + Dependency Tree під час валідації, підбір значення δ

N	Значення δ	Точність під час валідації	F1 під час валідації
1	1.2	89.71	92.81
2	1.1	88.97	92.12
3	0.9	**91.67**	**94.01**
4	0.8	90.44	93.15

Таблиця 4.16 Порівняння моделей

N	Назва моделі	Точність на тестуванні	F1 на тестуванні
1	DeBERTa Base	87.52 ± 0.26	90.80 ± 0.12
3	DeBERTa Base + Dependency Tree, $\delta = 0.9$	**87.67 ± 0.56**	**91.12 ± 0.27**

В результаті експериментів можемо спостерігати, що модель з використанням дерева залежностей має найкращу точність та значення F1.

4.7. Великі мовні моделі – LLM

Великі мовні моделі — це потужні алгоритми глибокого навчання для різних завдань обробки природної мови. Вони базуються на моделях з архітектурою Трансформер і навчаються на масивних наборах даних, що дає їм

змогу розуміти, перекладати, передбачати, генерувати текст. Ці моделі мають дуже велику кількість параметрів, які служать їх базою знань.

Хоча ці мовні моделі в більшості випадків створювалися для генерації тексту, їх можна використовувати для задачі класифікації.

Ми хотіли дослідити, чи можна використовувати моделі LLM для ідентифікації парафраз та наскільки складно їх налаштувати.

Для експерименту ми вибрали модель Llama 2 [91], яка була випущена в липні 2023 року. Важливо відзначити, що ця модель оптимізована для випадків використання діалогу та створення тексту.

Llama 2 доступна як ряд великих мовних моделей (LLM) із параметрами від 7 до 70 мільярдів. Для цього аналізу ми вибрали модель із 7 мільярдами параметрів через обмеження в обчислювальних потужностях.

Навчання та точне налаштування LLM є технічно та обчислювально складними через їх колосальний розмір. Щоб подолати цю проблему, було розроблено кілька підходів.

Ми використали бібліотеку Parameter-Efficient Finetuning (PEFT) [92] для тонкого налаштування Llama 2. Ми налаштовуємо лише кілька (додаткових) параметрів моделі за допомогою цієї бібліотеки, значно зменшуючи витрати на обчислення та зберігання.

Ця бібліотека підтримує кілька методів тонкого налаштування. Ми вибрали LoRa [93]. Головна перевага LoRa полягає в тому, що він не намагається тонко налаштувати оригінальну модель – він тримає попередньо навчені ваги замороженими (без змін). Замість цього він створює дві невеликі матриці, які будуть представляти оновлення. Завдання тонкого налаштування стає завданням побудови цих двох малих матриць.

$$h = W_0\, x + \Delta W x = W_0\, x + BAx = (W_0 + BA)x \qquad (4.21)$$

де $W_0 \in R^{d \times k}$ – оригінальні ваги моделі, матриці $B \in R^{d \times r}$ – та $A \in R^{r \times k}$, $r \ll \min(d, k)$. Оскільки r значно менше d та k, кількість параметрів матриць B та A може бути значно менше, ніж в оригінальній моделі.

Модель була налаштована, використовуючи параметри:

- Розмір батча = 16.
- Швидкість навчання lr = 2×10^{-5}.
- Weight decay = 0,01.
- Стратегія оптимізації AdamW (PyTorch).

Параметри LoRa:

- r = 16 (ранг матриць оновлення).
- LoRa Alpha = 16
- LoRa dropout = 0.1

Щоб налаштувати модель на наборі даних MRPC, ми використали одну NVIDIA A100 (40 ГБ – Google Collab).

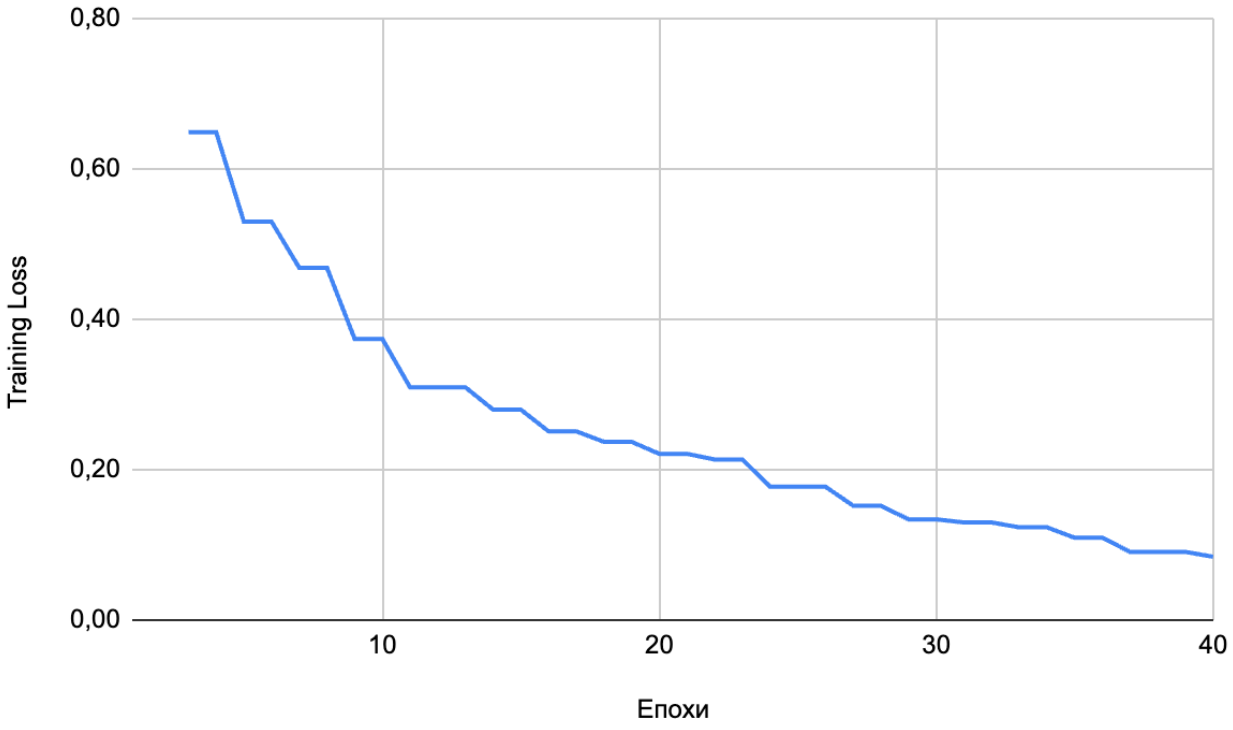


Рис. 4.6. Діаграма Training Loss під час тонкого налаштування Llama2

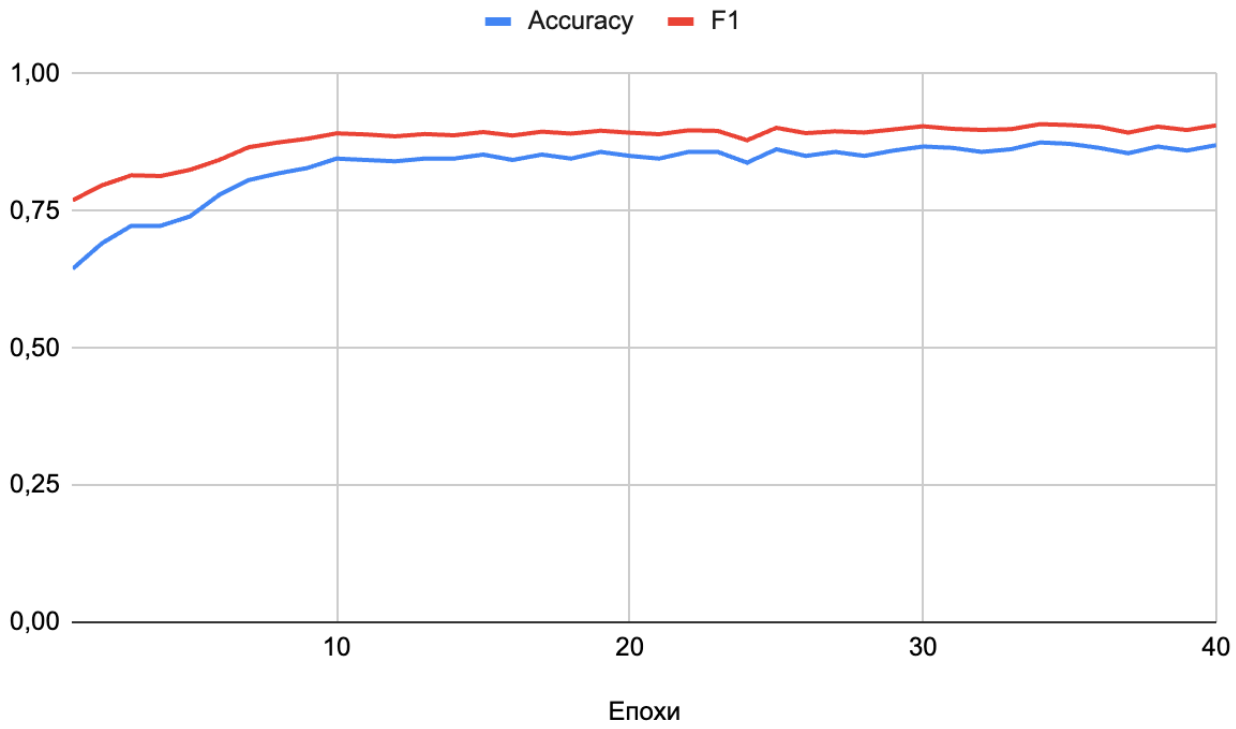


Рис. 4.7. Діаграма точності та F1 на валідації під час тонкого налаштування Llama2

Ми пробували точно налаштувати модель протягом 40 епох, але з рисунку 4.7 видно, що підвищення точності протягом останніх 20 – 30 епох є дуже незначним.

Таблиця 4.17 Результат експерименту з моделлю Llama 2 7B

Назва моделі	Точність на тестуванні	F1 на тестуванні
Llama 2 7b	84.52 ± 0.35	88.48 ± 0.38

В результаті експерименту можна сказати, що LLM також можна використовувати для виявлення парафраз, але вартість і складність їх тонкого налаштування значно вищі порівняно зі звичайними моделями на основі BERT. Ми не змогли досягти кращої точності з точно налаштованою моделлю Llama 2.

Також варто відмітити, що LLM моделі є моделями декодерами, таким чином вони було оптимізовані для генерації тексту.

Як наслідок, подальші покращення продуктивності моделі можуть бути зроблені шляхом дослідження великих моделей із ще більшою кількістю параметрів (наприклад модель Llama 2 70B) або підбором інших гіперпараметрів моделі.

4.8. Висновки до розділу 4

В результаті проведеного дослідження проаналізовано різні моделі з архітектурою Трансформер, а також використання матриці на основі дерева залежностей, як додаткової ознаки для моделі. Головним висновком є експериментальне підтвердження того, що дерева залежностей можуть покращити базові моделі з архітектурою Трансформер. В результаті отримано наступні результати:

1. Досліджено та проаналізовано різні моделі з архітектурою Трансформер.
2. В ході експериментів досліджено ефективність цих моделей для ідентифікації парафраз.
3. Досліджено та проаналізовано LLM модель Llama 2.
4. В ході експериментів досліджено ефективність Llama 2 для ідентифікації парафраз.
5. Запропоновано метод побудови ознаки на основі дерева залежностей для у self – attention шарі.
6. Створено модель, використовуючи дану модифікацію, та досліджено її ефективність.

ВИСНОВКИ

У результаті проведеного дослідження розглянуто моделі представлення семантики слів та речень та їх застосування для розв'язання задачі ідентифікації парафраз.

На основі результатів, отриманих під час дисертаційного дослідження, можна зробити наступні висновки:

1) При аналізі підходів до побудови векторного представлення речення помічено тенденцію переходу від ручних правил та застосування детермінованих алгоритмів до глибокого навчання моделей нейронних мереж. Враховуючи те, що сучасні моделі на основі Трансформерів та інших архітектур значно переважають їх попередників, у цьому є практичний сенс, але ціною стають більші розміри моделей та непрозорість прийняття рішень нейронними мережами. Щоб боротися з цими недоліками розроблені різні способи інтерпретації знань моделей. Проте ціль даної роботи – поєднати наявні сучасні моделі (наприклад, нейронні мережі) з класичними алгоритмами чи способами представлення даних (як от графи).
2) За допомогою синтаксичного дерева розбору проведено аналіз синтаксичної компоненти в моделях векторного представлення семантики речень. Експериментальне дослідження ефективності застосування даного представлення для розв'язання задачі граматичної корекції речення вказало на доцільність її використання. Важливо використовувати синтаксичну компоненту у комплексі з іншими (наприклад, семантичною) для найповнішого представлення речення. Використання класичних алгоритмів, як от Ерлі, з певними модифікаціями, може бути використане як цінне джерело синтаксичних ознак при побудові моделі. Складність використання цього алгоритму полягає у необхідності побудови повної граматики мови. Різні природні

мови будуть мати різні граматики – це суттєво ускладнює застосування такого підходу до менш досліджених мов.

3) Проведений аналіз використання дерева залежностей, основи для репрезентації структури речення. Побудована модель на основі цих ознак для ідентифікації парафраз з використанням методів машинного навчання (такого як SVM). Вона ефективно вирішує поставлену задачу і досягає конкурентних результатів у цьому класі моделей.
4) Досліджено та проаналізовано різні моделі з архітектурою Трансформер та їх модифікації. Дані моделі розділено на три класи за кількістю параметрів та досліджено їх ефективність на задачі ідентифікації парафраз.
5) Аналізуючи попередні результати використання синтаксичного дерева розбору, дерева залежностей та сучасних моделей на основі архітектури Трансформер, запропоновано модифікацію шару self-attention з використанням ознак на основі дерева залежностей. В результаті експериментів проаналізовано якість моделі. Запропонована модифікація показала кращі показники точності та F1, ніж базова модель. Таким чином продемонстровано доцільність застосування цих ознак для покращення розуміння структури речення.
6) Зважаючи на велику популярність LLM моделей для генерації тексту, проаналізовано їх здатність до класифікації даних. В результаті експериментів розглянуто Llama 2 модель, що показала співмірні результати з базовими моделями на основі архітектури Трансформер. Проте з точки зору практичної цінності цих моделей для такого класу задач, їх використання та тонке налаштування вимагає значно більше обчислювальних ресурсів, ніж для звичайних моделей.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Jacob Devlin, Ming-Wei Chang, Kenton Lee, & Kristina Toutanova. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. https://doi.org/10.48550/arXiv.1810.04805
2. Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, & Veselin Stoyanov. (2019). RoBERTa: A Robustly Optimized BERT Pretraining Approach. https://doi.org/10.48550/arXiv.1907.11692
3. Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, & Radu Soricut. (2020). ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. https://doi.org/10.48550/arXiv.1909.11942
4. Tomas Mikolov, Kai Chen, Greg Corrado, & Jeffrey Dean. (2013). Efficient Estimation of Word Representations in Vector Space. https://doi.org/10.48550/arXiv.1301.3781
5. Vrublevskyi, V., & Marchenko, O. (2019). Grammar Error Correcting by the Means of CFG Parser. 2019 IEEE International Conference On Advanced Trends In Information Theory (ATIT), 430–435. https://doi.org/10.1109/ATIT49449.2019.9030458
6. Vrublevskyi, V., & Marchenko, O. (2020). Paraphrase Identification Using Dependency Tree and Word Embeddings. 2020 IEEE 2nd International Conference On Advanced Trends In Information Theory (ATIT), 372–375. https://doi.org/10.1109/ATIT50783.2020.9349338
7. Vrublevskyi, V., & Marchenko, O. (2022). Development and Analysis of a Sentence Semantics Representation Model. Cybernetics and Systems Analysis, 58(1), 16–23. https://doi.org/10.1007/s10559-022-00430-9

8. Vrublevskyi, V. N., & Marchenko, O. O. (2023). Review of approaches for paraphrase identification. Bulletin of Taras Shevchenko National University of Kyiv. Physics and Mathematics, (1), 71–78. https://doi.org/10.17721/1812-5409.2023/1.10
9. Marchenko, O., & Vrublevskyi, V. (2023). Comparison of Transformer-based Deep Learning Methods for the Paraphrase Identification task. X International Scientific Conference "Information Technology and Implementation" (IT&I-2023), 447-455.
10. Models for representing the semantics of natural language sentences. (n.d.). vrublevskiyvitaliy/phd_sentence_semantic_models. GitHub. https://github.com/vrublevskiyvitaliy/phd_sentence_semantic_models
11. Christiane Fellbaum (1998, ed.) WordNet: An Electronic Lexical Database. Cambridge, MA: MIT Press.
12. Maki, W. S., McKinley, L. N., & Thompson, A. G. (2004). Semantic distance norms computed from an electronic dictionary (WordNet). Behavior Research Methods, Instruments, & Computers, 36(3), 421–431. https://doi.org/10.3758/BF03195590
13. Jurafsky, D., & Martin, J. (2023). Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. (Vol. 3) Retrieved January 1, 2024, from https://web.stanford.edu/~jurafsky/slp3/ed3book.pdf
14. Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, & Jeffrey Dean. (2013). Distributed Representations of Words and Phrases and their Compositionality. https://doi.org/10.48550/arXiv.1310.4546
15. Vakhovska, O. (2023). Основи комп'ютерної лінгвістики. Навчально-методичний посібник [Basics of computational linguistics. A handbook].
16. Marneffe, M.C., Manning, C., Nivre, J., & Zeman, D. (2021). Universal Dependencies. Computational Linguistics, 47(2), 255–308. https://doi.org/10.1162/coli_a_00402

17. Banarescu, L., Bonial, C., Cai, S., Georgescu, M., Griffitt, K., Hermjakob, U., Knight, K., Koehn, P., Palmer, M., & Schneider, N. (2013). Abstract Meaning Representation for Sembanking. In Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse (pp. 178–186). Association for Computational Linguistics.
18. Kasper, R. (1989). A Flexible Interface for Linking Applications to Penman′s Sentence Generator. In Speech and Natural Language: Proceedings of a Workshop Held at Philadelphia, Pennsylvania, February 21-23, 1989.
19. Arora, S., Liang, Y., & Ma, T. (2017). A Simple but Tough-to-Beat Baseline for Sentence Embeddings. International Conference on Learning Representations.
20. Amidi, A., & Amidi, S. (n.d.). Recurrent Neural Networks cheatsheet. CS 230 - Recurrent Neural Networks Cheatsheet. Retrieved January 1, 2024, from https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks
21. Richard Socher, Christopher D. Manning, & Andrew Y. Ng (2010) Learning continuous phrase representations and syntactic parsing with recursive neural networks. In Proceedings of the NIPS-2010 Deep Learning and Unsupervised Feature Learning Workshop, pages 1–9, 2010.
22. Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks, 5(2), 157-166. https://doi.org/10.1109/72.279181
23. Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. Neural Computation, 9(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735
24. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., … Polosukhin, I. (2017). Attention is All you Need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), Advances in Neural Information Processing Systems (Vol. 30). https://doi.org/10.48550/arXiv.1706.03762

25. Dolan, B., Quirk, C., & Brockett, C. (2004). Unsupervised Construction of Large Paraphrase Corpora: Exploiting Massively Parallel News Sources. In COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics (pp. 350–356). COLING.
26. Ganitkevitch, J., Van Durme, B., & Callison-Burch, C. (2013). PPDB: The Paraphrase Database. In Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (pp. 758–764). Association for Computational Linguistics.
27. Marelli, M., Bentivogli, L., Baroni, M., Bernardi, R., Menini, S., & Zamparelli, R. (2014). SemEval-2014 Task 1: Evaluation of Compositional Distributional Semantic Models on Full Sentences through Semantic Relatedness and Textual Entailment. In Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014) (pp. 1–8). Association for Computational Linguistics. https://doi.org/10.3115/v1/S14-2001
28. Kaggle. (2017). Quora duplicate questions. Kaggle. Retrieved January 1, 2024 from https://www.kaggle.com/competitions/quora-question-pairs
29. Wieting, J., & Gimpel, K. (2018). ParaNMT-50M: Pushing the Limits of Paraphrastic Sentence Embeddings with Millions of Machine Translations. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers) (pp. 451–462). Association for Computational Linguistics. https://doi.org/10.18653/v1/P18-1042
30. Fellbaum, C. (1998). WordNet: An electronic lexical database. MIT Press. https://doi.org/10.7551/mitpress/7287.001.0001
31. Landauer, T. K., Foltz, P. W., & Laham, D. (1998). An introduction to latent semantic analysis. Discourse Processes, 25(2–3), 259–284. https://doi.org/10.1080/01638539809545028
32. Boonthum, C. (2004). iSTART: Paraphrase Recognition. In Proceedings of the ACL Student Research Workshop (pp. 31–36). Association for Computational Linguistics.

33. Sowa, J. F. (1992). Conceptual graphs as a universal knowledge representation. Computers & Mathematics with Applications, 23(2), 75–93. https://doi.org/10.1016/0898-1221(92)90137-7
34. Sleator, D., & Temperley, D. (1993). Parsing English with a Link Grammar. In Proceedings of the Third International Workshop on Parsing Technologies (pp. 277–292). Association for Computational Linguistics.
35. Mihalcea, R., Corley, C., & Strapparava, C. (2006). Corpus-Based and Knowledge-Based Measures of Text Semantic Similarity. In Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1 (pp. 775–780). AAAI Press.
36. Peter D. Turney. (2002). Mining the Web for Synonyms: PMI-IR versus LSA on TOEFL. https://doi.org/10.48550/arXiv.cs/0212033
37. Leacock, C., Chodorow, M., & Miller, G. (1998). Using Corpus Statistics and WordNet Relations for Sense Identification. Computational Linguistics, 24(1), 147–165.
38. Wu, Z., & Palmer, M. (1994). Verb Semantics and Lexical Selection. In 32nd Annual Meeting of the Association for Computational Linguistics (pp. 133–138). Association for Computational Linguistics. https://doi.org/10.3115/981732.981751
39. Madnani, N., Tetreault, J., & Chodorow, M. (2012). Re-examining Machine Translation Metrics for Paraphrase Identification. In Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (pp. 182–190). Association for Computational Linguistics.
40. Papineni, K., Roukos, S., Ward, T., & Zhu, W.J. (2002). Bleu: a Method for Automatic Evaluation of Machine Translation. In Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (pp. 311–318). Association for Computational Linguistics. https://doi.org/10.3115/1073083.1073135

41. Doddington, G. (2002). Automatic Evaluation of Machine Translation Quality Using N-Gram Co-Occurrence Statistics. In Proceedings of the Second International Conference on Human Language Technology Research (pp. 138–145). Morgan Kaufmann Publishers Inc..
42. Snover, M., Dorr, B., Schwartz, R., Micciulla, L., & Makhoul, J. (2006). A Study of Translation Edit Rate with Targeted Human Annotation. In Proceedings of the 7th Conference of the Association for Machine Translation in the Americas: Technical Papers (pp. 223–231). Association for Machine Translation in the Americas.
43. Aha, W., Kibler, D., & Albert, M. (1991). Instance-Based Learning Algorithms. Machine Learning, 6, 37-66. https://doi.org/10.1023/A:1022689900470
44. Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP) (pp. 3982–3992). Association for Computational Linguistics. https://doi.org/10.18653/v1/D19-1410
45. Indurkhya, N., & Damerau, F. J. (2010). Handbook of Natural Language Processing. Chapman & Hall/CRC.
46. Глибовець М.М., Олецький О.В. (2002). Штучний інтелект: Підручник. Вид. дім «КМ Академія»
47. Волохов В.М. (2013). Методичні рекомендації до лабораторного практикуму побудови мовних процесорів з дисципліни «Системне програмування»
48. Earley, J. (1970). An Efficient Context-Free Parsing Algorithm. Commun. ACM, 13(2), 94–102. https://doi.org/10.1145/362007.362035
49. Willink, E.D. (2001). Meta-compilation for C++.
50. Aycock, J., & Horspool, R. (2002). Practical Earley Parsing. Comput. J., 45, 620-630. https://doi.org/10.1093/comjnl/45.6.620

51. Appel, A. W., & Palsberg, J. (2009). Modern compiler implementation in Java. Cambridge University Press.
52. Aho, A. V., & Peterson, T. G. (1972). A Minimum Distance Error-Correcting Parser for Context-Free Languages. SIAM Journal on Computing, 1(4), 305–312. https://doi.org/10.1137/0201022
53. Tanaka, E., & Kasai, T. (1976). Synchronization and substitution error-correcting codes for the Levenshtein metric. IEEE Transactions on Information Theory, 22(2), 156-162. https://doi.org/10.1109/TIT.1976.1055532
54. Marchenko, O., Anisimov, A., Zavadskyi, I., & Melnikov, E. (2018). English Text Parsing by Means of Error Correcting Automaton. In M. Silberztein, F. Atigui, E. Kornyshova, E. Métais, & F. Meziane (Eds.), Natural Language Processing and Information Systems (pp. 281–289). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-91947-8_28
55. Fu, K. S. (1977). Error-Correcting Parsing For Syntactic Pattern Recognition. In A. Klinger, K. S. Fu, & T. L. Kunii (Eds.), Data Structures, Computer Graphics, and Pattern Recognition (pp. 449–484). https://doi.org/10.1016/B978-0-12-415050-8.50020-3
56. Dahlmeier, D., Ng, H., & Wu, S. (2013). Building a Large Annotated Corpus of Learner English: The NUS Corpus of Learner English. In Proceedings of the Eighth Workshop on Innovative Use of NLP for Building Educational Applications (pp. 22–31). Association for Computational Linguistics.
57. Ng, H., Wu, S., Wu, Y., Hadiwinoto, C., & Tetreault, J. (2013). The CoNLL-2013 Shared Task on Grammatical Error Correction. In Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task (pp. 1–12). Association for Computational Linguistics.
58. Napoles, C., Sakaguchi, K., & Tetreault, J. (2017). JFLEG: A Fluency Corpus and Benchmark for Grammatical Error Correction. In Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers (pp. 229–234). Association for Computational Linguistics. https://doi.org/10.18653/v1/E17-2037

59. Napoles, C., & Callison-Burch, C. (2017). Systematically Adapting Machine Translation for Grammatical Error Correction. In Proceedings of the 12th Workshop on Innovative Use of NLP for Building Educational Applications (pp. 345–356). Association for Computational Linguistics. https://doi.org/10.18653/v1/W17-5039
60. CNAP/SMT-for-GEC. GitHub. (n.d.). Retrieved January 1, 2024, from https://github.com/cnap/smt-for-gec
61. Milajevs, D., Kartsaklis, D., Sadrzadeh, M., & Purver, M. (2014). Evaluating Neural Word Representations in Tensor-Based Compositional Settings. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP) (pp. 708–719). Association for Computational Linguistics. https://doi.org/10.3115/v1/D14-1079
62. Wan, S., Dras, M., Dale, R., & Paris, C. (2006). Using Dependency-Based Features to Take the 'Para-farce' out of Paraphrase. In Proceedings of the Australasian Language Technology Workshop 2006 (pp. 131–138).
63. Harold W. Kuhn (1955). The Hungarian method for the assignment problem. Naval Research Logistics (NRL), 52. https://doi.org/10.1002/nav.3800020109
64. Riesen, K., Neuhaus, M., & Bunke, H. (2007). Bipartite Graph Matching for Computing the Edit Distance of Graphs. In F. Escolano & M. Vento (Eds.), Graph-Based Representations in Pattern Recognition (pp. 1–12). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-72903-7_1
65. Sidorov, G., Velasquez, F., Stamatatos, E., Gelbukh, A., & Chanona-Hernández, L. (2014). Syntactic N-grams as machine learning features for natural language processing. Expert Systems with Applications, 41(3), 853–860. https://doi.org/10.1016/j.eswa.2013.08.015
66. Scikit-learn Machine Learning in Python . scikit. (n.d.). Retrieved January 1, 2024, from https://scikit-learn.org/stable/
67. Spacy · Industrial-strength Natural Language Processing in Python. (n.d.). https://spacy.io/

68. Xue, N., Weischedel, R., Marcus, M., Palmer, M., Hovy, E., Belvin, R., Pradhan, S., & Ramshaw, L. (2010). OntoNotes: A Large Training Corpus for Enhanced Processing. In Handbook of Natural Language Processing and Machine Translation. Springer.
69. Kozareva, Z., & Montoyo, A. (2006). Paraphrase Identification on the Basis of Supervised Machine Learning Techniques. In T. Salakoski, F. Ginter, S. Pyysalo, & T. Pahikkala (Eds.), Advances in Natural Language Processing (pp. 524–533). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/11816508_52
70. Ul-Qayyum, Z., & Altaf, W. (2012). Paraphrase Identification using Semantic Heuristic Features. Research Journal of Applied Sciences, Engineering and Technology, 4, 4894-4904.
71. Blacoe, W., & Lapata, M. (2012). A Comparison of Vector-based Representations for Semantic Composition. In Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (pp. 546–556). Association for Computational Linguistics.
72. Cheng, J., & Kartsaklis, D. (2015). Syntax-Aware Multi-Sense Word Embeddings for Deep Compositional Models of Meaning. In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (pp. 1531–1542). Association for Computational Linguistics. https://doi.org/10.18653/v1/D15-1177
73. Ji, Y., & Eisenstein, J. (2013). Discriminative Improvements to Distributional Sentence Similarity. In Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (pp. 891–896). Association for Computational Linguistics.
74. Paraphrase identification (state of the art). Paraphrase Identification (State of the art) - ACL Wiki. (n.d.). Retrived January 1, 2024, from https://aclweb.org/aclwiki/Paraphrase_Identification_(State_of_the_art)

75. Bloem, P. (n.d.). TRANSFORMERS FROM SCRATCH. Transformers from scratch. Retrieved January 1, 2024, from https://peterbloem.nl/blog/transformers
76. Victor Sanh, Lysandre Debut, Julien Chaumond, & Thomas Wolf. (2020). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. https://doi.org/10.48550/arXiv.1910.01108
77. Geoffrey Hinton, Oriol Vinyals, & Jeff Dean. (2015). Distilling the Knowledge in a Neural Network. https://doi.org/10.48550/arXiv.1503.02531
78. Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, & Luke Zettlemoyer. (2019). BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. https://doi.org/10.48550/arXiv.1910.13461
79. Kevin Clark, Minh-Thang Luong, Quoc V. Le, & Christopher D. Manning. (2020). ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators. https://doi.org/10.48550/arXiv.2003.10555
80. Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., & Bowman, S. (2018). GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. In Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP (pp. 353–355). Association for Computational Linguistics. https://doi.org/10.18653/v1/W18-5446
81. Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, & Denny Zhou. (2020). MobileBERT: a Compact Task-Agnostic BERT for Resource-Limited Devices. https://doi.org/10.48550/arXiv.2004.02984
82. Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W. Mahoney, & Kurt Keutzer. (2021). I-BERT: Integer-only BERT Quantization. https://doi.org/10.48550/arXiv.2101.01321
83. Zhen Dong, Zhewei Yao, Amir Gholami, Michael Mahoney, & Kurt Keutzer. (2019). HAWQ: Hessian AWare Quantization of Neural Networks with Mixed-Precision. https://doi.org/10.48550/arXiv.1905.03696

84. Pengcheng He, Xiaodong Liu, Jianfeng Gao, & Weizhu Chen. (2021). DeBERTa: Decoding-enhanced BERT with Disentangled Attention. https://doi.org/10.48550/arXiv.2006.03654
85. Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, & Samuel R. Bowman. (2020). SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems. https://doi.org/10.48550/arXiv.1905.00537
86. Forrest N. Iandola, Albert E. Shaw, Ravi Krishna, & Kurt W. Keutzer. (2020). SqueezeBERT: What can computer vision teach NLP about efficient neural networks?. https://doi.org/10.48550/arXiv.2006.11316
87. Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, & Kurt Keutzer. (2016). SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. https://doi.org/10.48550/arXiv.1602.07360
88. Bisong, E. (2019). Building Machine Learning and Deep Learning Models on Google Cloud Platform (1st ed.). Apress Berkeley, CA. https://doi.org/10.1007/978-1-4842-4470-8
89. Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Le Scao, T., Gugger, S., Drame, M., Lhoest, Q., & Rush, A. (2020). Transformers: State-of-the-Art Natural Language Processing. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations (pp. 38–45). Association for Computational Linguistics. https://doi.org/10.18653/v1/2020.emnlp-demos.6
90. Bai, J., Wang, Y., Chen, Y., Yang, Y., Bai, J., Yu, J., & Tong, Y. (2021). Syntax-BERT: Improving Pre-trained Transformers with Syntax Trees. In Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume (pp. 3011–3020). Association for Computational Linguistics. https://doi.org/10.18653/v1/2021.eacl-main.262

91. Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, & Thomas Scialom. (2023). Llama 2: Open Foundation and Fine-Tuned Chat Models. https://doi.org/10.48550/arXiv.2307.09288

92. Huggingface. (n.d.). Huggingface/PEFT: 🤗 PEFT: State-of-the-art parameter-efficient fine-tuning. GitHub. Retrieved January 1, 2024, from https://github.com/huggingface/peft

93. Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, & Weizhu Chen. (2021). LoRA: Low-Rank Adaptation of Large Language Models. https://doi.org/10.48550/arXiv.2106.09685

ДОДАТКИ

Додаток А

Список публікацій здобувача за темою дисертації:

***Наукові праці, в яких опубліковані основні наукові результати дисертації*:**

1. Vrublevskyi, V., & Marchenko, O. (2022). Development and Analysis of a Sentence Semantics Representation Model. Cybernetics and Systems Analysis, 58(1), 16–23. https://doi.org/10.1007/s10559-022-00430-9

2. Vrublevskyi, V. N., & Marchenko, O. O. (2023). Review of approaches for paraphrase identification. Bulletin of Taras Shevchenko National University of Kyiv. Physics and Mathematics, (1), 71–78. https://doi.org/10.17721/1812-5409.2023/1.10

Наукові праці, які засвідчують апробацію матеріалів дисертації:

1. Vrublevskyi, V., & Marchenko, O. (2019). Grammar Error Correcting by the Means of CFG Parser. In *Proceedings of the 2019 IEEE International Conference On Advanced Trends In Information Theory (ATIT)*, 430–435. https://doi.org/10.1109/ATIT49449.2019.9030458
2. Vrublevskyi, V., & Marchenko, O. (2020). Paraphrase Identification Using Dependency Tree and Word Embeddings. In *Proceedings of the 2020 IEEE 2nd International Conference On Advanced Trends In Information Theory (ATIT)*, 372–375. https://doi.org/10.1109/ATIT50783.2020.9349338
3. Marchenko, O., & Vrublevskyi, V. (2023). Comparison of Transformer-based Deep Learning Methods for the Paraphrase Identification task. In *Proceedings of the 2023 X International Scientific Conference "Information Technology and Implementation" (IT&I-2023)*, 447-455. https://ceur-ws.org/Vol-3624/Short_5.pdf