Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Програмної інженерії
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти – другий (магістерський)

Дослідження методів аналізу природньої мови для заповнення форм в чат-ботах
(тема)

Виконав: студент 2 курсу, групи ІПЗм-18-1
Заєв А.О.
(прізвище, ініціали)

спеціальності 121– Інженерія програмного забезпечення
(код і повна назва спеціальності)
Освітньо-наукової програми
(тип програми)
Інженерія програмного забезпечення
(тип програми)

Керівник к.т.н., доц. Турута О.П.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Дудар З.В.
(підпис) (прізвище, ініціали)

2020 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
Кафедра Програмної інженерії

Рівень вищої освіти – другий (магістерський)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва)
Тип програми освітньо-наукова програма

Освітня програма Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

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

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

Студентові Заєву Андрію Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи (проекту): Дослідження методів аналізу природньої мови для заповнення форм в чат-ботах
затверджена наказом по університету від " " 2020 р. №
2. Термін подання студентом роботи (проекту) 18 травня 2020
3. Вихідні дані до роботи (проекту) алгоритми заповнення форм на основі повідомлень користувача, методи попередньої обробки природньої мови, пояснювальна записка. Використовувати ОС Windows, мову програмування Python, фреймворк машинного навчання PyTorch.

4. Перелік питань, що потрібно опрацювати в роботі: мета роботи, аналіз проблемної галузі і постановка задачі, огляд методів заповнення форм на основі даних з текстових повідомлень користувача, пристосування методів для вирішення поставленої задачі, аналіз якості моделей отримання даних форми та шляхів її покращення.

(Наступний аркуш завдання)

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка*
1.	Аналіз предметної галузі	03 квітня 2020 р.	
2.	Огляд існуючих методів	12 квітня 2020 р.	
3.	Дослідження методів аналізу природньої мови для заповнення форм	20 квітня 2020 р.	
4.	Підготовка пояснювальної записки	26 квітня 2020 р.	
5.	Спецчастина	30 квітня 2020 р.	
6.	Підготовка презентації та доповіді	06 травня 2020 р.	
7.	Попередній захист	08 травня 2020 р.	
8.	Нормоконтроль, рецензування	11 травня 2020 р.	
9.	Занесення диплома в електронний архів	14 травня 2020 р.	
10.	Допуск до захисту у зав. кафедри	15 травня 2020 р.	
* заповнюється вручну після виконання чергового пункту			

Дата видачі завдання 27 березня 2020 р.

Студент_____
(підпис)

Керівник роботи (проекту) _____ к.т.н., доцент Турута О.П.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Атестаційна робота магістра містить: 60 с., 11 рис., 9 табл., 3 додатки, 45 джерел.

МАШИННЕ НАВЧАННЯ, НЕЙРОННІ МЕРЕЖІ, ОБРОБКА ПРИРОДНЬОЇ МОВИ, ЧАТ-БОТИ, BERT, PYTHON.

Метою роботи є розробка алгоритму для отримання з тексту природньою мовою – текстового повідомлення від користувача – даних, що необхідні для заповнення форми, на основі сучасних підходів до обробки природньої мови.

Методи розробки базуються на нейромережевій моделі природньої мови BERT, мові програмування Python та фреймворку для машинного навчання PyTorch.

В результаті роботи розглянуто сучасні підходи до вирішення проблеми заповнення форм та до розуміння природньої мови у цілому, на основі цих підходів удосконалено існуючий алгоритм об’єднаної класифікації наміру та маркування слотів, реалізовано даний алгоритм в якості моделі для чат-боту.

MACHINE LEARNING, NEURAL NETWORKS, NATURAL LANGUAGE PROCESSING, CHATBOTS, BERT, PYTHON.

The aim of this work is to develop an algorithm for obtaining values for form filling from a natural language text – user message – using modern approaches to natural language processing.

Development methods are based on neural network model BERT for natural languages, programming language Python and machine learning framework PyTorch.

As the result of this work state-of-the-art approaches to form filling task solving and natural language understanding were analyzed. Based on them existing algorithm for joint intent classification and slot labeling was improved. Developed model was used as a model for chatbot.

ЗМІСТ

ВСТУП

В останні роки у зв’язку з розвитком інформаційних комп’ютеризованих систем та з ростом обчислювальної здатності комп’ютерів значно збільшився попит на практичне розв’язання різноманітних задач з обробки природньої мови, оскільки тепер з’явилася можливість їх використання у прикладних додатках. Прикладами таких задач є розпізнавання мови, питально-відповідальні системи, машинний переклад та інші.

До задач обробки природньої мови відноситься і заповнення комп’ютерних форм на основі повідомлень природньою мовою з боку користувача інформаційної системи. Дані інформаційні системи можуть являти собою як складні додатки, що використовуються на різноманітних підприємствах, так і персональні асистенти користувачів, такі як чат-боти. В останньому випадку використання даного підходу дозволяє автоматизувати рутинні дії користувача та, потенційно, зменшити кількість помилок у результуючих даних, оскільки користувач має тільки перевірити коректність даних замість введення їх вручну.

Через це методи аналізу природньої мови в цілому та конкретні реалізації даних методів до певних задач активно розвиваються. При цьому постає проблема їх можливої інтеграції між собою та подальшого розвитку в межах поставленої задачі, що визначає актуальність даної роботи.

Тема роботи тісно пов’язана з таким напрямком наукових досліджень на кафедрі Програмної інженерії як обробка природньої мови, а застосовані у ній методи та підходи широко використовуються у напрямку інтелектуального аналізу даних [1].

Метою роботи, таким чином, є розробка алгоритму для отримання з тексту природньою мовою даних, що необхідні для заповнення форми, на основі сучасних підходів до обробки природньої мови. Даний алгоритм має отримувати на вхід текст природньою мовою та видавати дію, яку користувач хоче здійснити, та набір вхідних даних для цієї дії в межах заданої предметної області.

Об’єктом дослідження є методи аналізу природньої мови, предметом – методи заповнення форм даними з тексту природньою мовою.

Для досягнення мети в роботі було використано наступні методи дослідження:

– аналіз задачі заповнення форм та існуючих підходів до її вирішення;
– синтез алгоритмів та моделей, що використовуються для аналізу природньої мови на різних етапах, для побудови цілісного підходу до вирішення задачі отримання необхідних даних виходячи з тексту природньою мовою;
– моделювання процесу отримання даних з тексту та заповнення форми за допомогою нейромережевого підходу;
– експерименти над розробленою мовною моделлю з використанням реальних вхідних даних, аналіз та обробка результатів даних експериментів.

Новизна роботи полягає в удосконаленні існуючих підходів до розв’язання об’єднаної задачі класифікації наміру (intent classification) та маркування слотів (slot labeling) на основі нейромережевої мовної моделі BERT, а саме у покращенні точності їх роботи у випадку задачі заповнення форм за рахунок використання комбінації методів, що не застосовувалася раніше.

Подальше використання результатів дослідження можливе у декількох напрямках:

– розробка нових застосувань (у тому числі чат-ботів), де існує необхідність автоматичного отримання даних з тексту, що вводиться користувачем;
– дослідження можливої адаптації розроблених моделей до інших галузей обробки природньої мови;
– використання в якості бази для можливого вдосконалення отриманих результатів для задачі заповнення форм.

1 ОГЛЯД ПРЕДМЕТНОЇ ГАЛУЗІ

Задача заповнення форм на основі тексту природньою мовою є окремим випадком двох окремих задач – класифікації наміру (intent classification) та маркування слотів (slot labeling) – з такої підгалузі обробки природньої мови як розуміння природньої мови (natural language understanding) [2].

Задача класифікації наміру (також розпізнавання наміру, intent detection) полягає у співвіднесенні речення або тексту природньою мовою з наміром користувача, який закладений у них. Оскільки всі наміри відомі наперед, а їх кількість обмежена, ця задача зводиться до класифікації вхідних текстів за цими намірами [3].

В свою чергу, задача маркування слотів (також заповнення слотів, slot filling) полягає у визначенні у вхідному реченні або тексті слів або словосполучень, що відповідають заданим слотам при відомому намірі. Ця задача є окремим випадком задачі маркування послідовності (sequence labelling), яка полягає у співвіднесенні елементів вхідної послідовності та їх маркувань; в даному випадку, елементами послідовності є слова у тексті, а маркування – слоти для заданого наміру [4].

Обробка природньої мови є галуззю штучного інтелекту, що вивчає проблеми аналізу та синтезу природньої мови. Вона включає в себе велику кількість задач різного характеру, у тому числі такі як [6]:

– видобування даних;

– синтез мовлення;

– розпізнавання мови;

– генерування природньої мови;

– машинний переклад;

– знаходження відповідей на питання.

Розуміння природньої мови в свою чергу спрямоване на отримання з тексту його змісту та, за необхідністю, виконання певних дій за основі цього змісту.

Історично різні задачі обробки природньої мови зводилися до використання

різних інструментів їх розв’язання. Так, рішення задачі класифікації наміру базувалися на методі опорних векторів (support vector machines, SVM) для мультикласової класифікації вхідного тексту [7].

В той же час для вирішення задачі маркування слотів застосовувався метод на основі умовних випадкових полів (conditional random field, CRF), що дозволяє виконувати класифікацію з урахуванням контексту [8].

На сьогоднішній день найбільшого розвитку набули методи, що засновані на використанні нейромереж [9]. Причиною цього став розвиток методів глибинного навчання та точність результатів, яких вони дозволяють досягти. Крім того, ці методи вирішення задач розуміння природньої мови зазвичай є загальними і дозволяють, з незначними змінами, вирішувати різні підзадачі.

Для обробки природньої мови переважними типами нейронних мереж були рекурентні нейронні мережі (recurrent neural networks, RNN) та згорткові нейронні мережі (convolutional neural networks, CNN). Ці типи мереж дозволяють знаходити внутрішній взаємозв’язок між елементами тексту природньою мовою (за допомогою зворотнього зв’язку у рекурентних мереж та узагальнення у згорткових мережах [10]) і тому показують хороші результати в області обробки таких текстів. Серед конкретних архітектур мереж, що застосовувалися, можна відзначити довгу короткочасну пам’ять (long short-term memory, LSTM) [11] та вентильний рекурентний вузол (gated recurrent unit, GRU) [12].

Рекурентні нейронні мережі на основі довгої короткочасної пам’яті, на відміну від звичайних рекурентних мереж, використовують спеціальні вузли пам’яті (див. рис. 2.1) з лінійною активаційною функцією для збереження інформації. Через лінійний характер функції активації дані мережі є більш стійкими до градієнтного розмиття ваг мережі з плином часу [13].

Використання даних моделей в області обробки природньої мови показало гарні результати через їх здатність до збереження довготривалих зв’язків між елементами вхідної послідовності та простоти їх навчання у порівнянні зі звичайними рекурентними мережами [14].

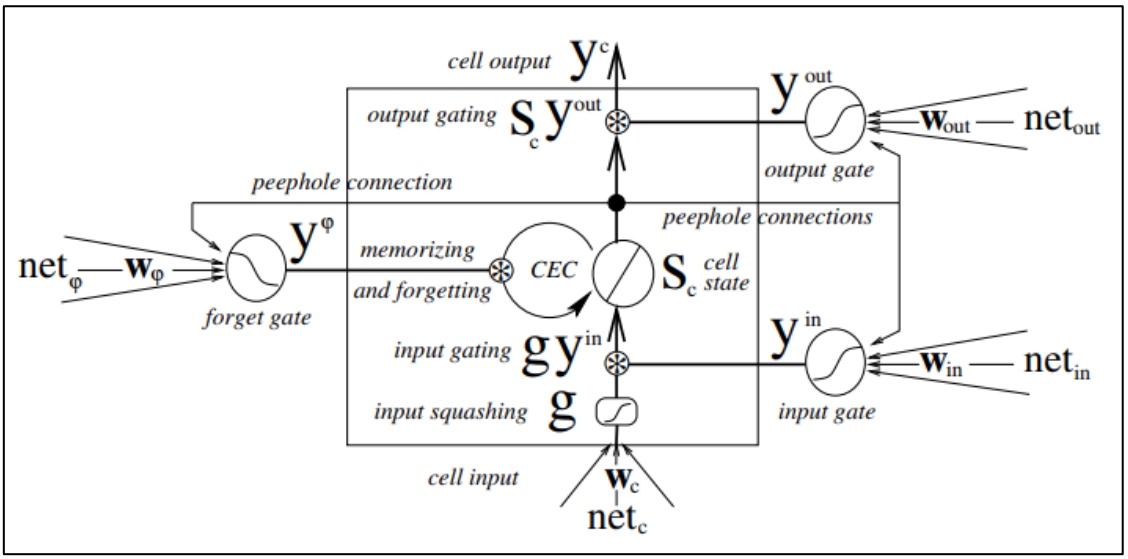

Рисунок 2.1 – Структура блоку довгої короткочасної пам’яті [15]

Подальшим розвитком моделей з довгою короткочасною пам’яттю стали рекурентні мережі з вентильними рекурентними вузлами (див. рис. 2.2). Цей вузол побудований на основі вузла пам’яті, але у такому вигляді, що новий вузол є простішим для реалізації та обчислення. Так, замість чотирьох вентилів у вузлі довгої короткочасної пам’яті в ньому наявні лише два [12].

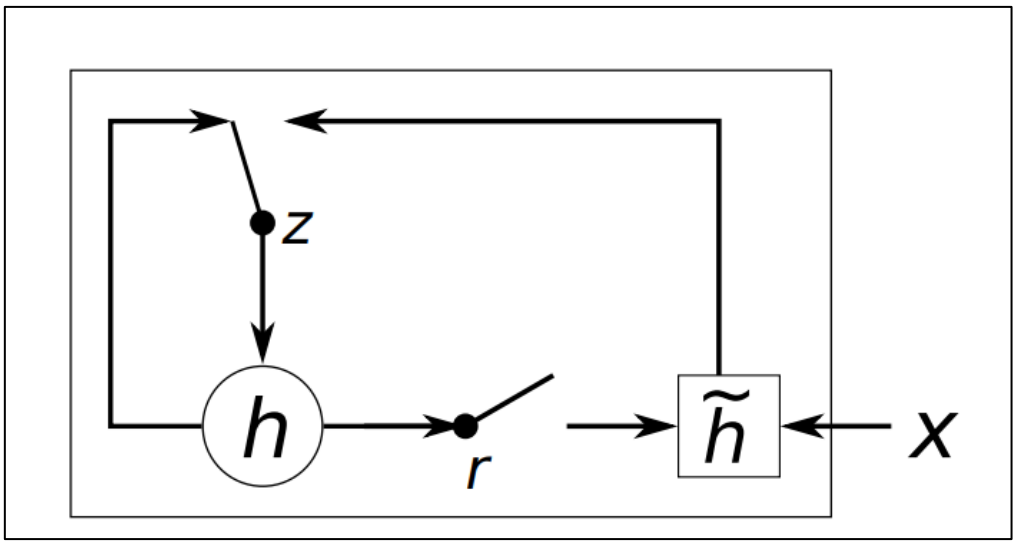

Рисунок 2.2 – Структура вентильного рекурентного вузла [13]

Додатково до цих архітектур, що є загальними для багатьох напрямів машинного навчання, було створено також специфічні механізми для обробки природньої мови – архітектура кодувальник-декодувальник (Encoder-Decoder) та

увага (attention).

В архітектурі кодувальник-декодувальник (див. рис. 2.3) мережа складається з двох основних блоків (підмереж) [16]:

– кодувальник, що перетворює вхідну послідовність довільної довжини у проміжну послідовність фіксованої довжини;
– декодувальник, що перетворює проміжну послідовність у вихідну послідовність заданої довжини.

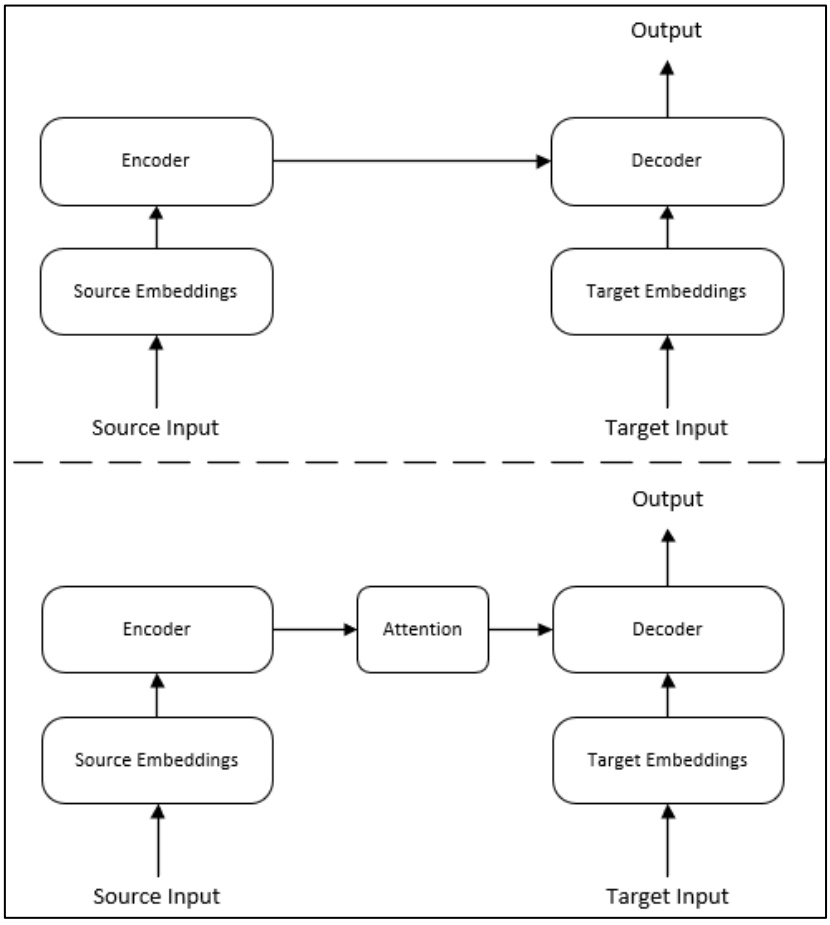

Рисунок 2.3 – Варіанти архітектури кодувальник-декодувальник

Така архітектура, на відміну від звичайних рекурентних мереж, дозволяє маніпулювати довжиною послідовностей, завдяки чому довжина вихідної послідовності може бути відмінною від довжини вхідної, що робить модель набагато більш гнучкою [16]. Кодувальник та декодувальник при цьому реалізуються з використанням існуючих архітектур, таких як вищезгадані LSTM та GRU.

Модель кодувальника-декодувальника може використовуватися двома шляхами: генерувати результуючу послідовність на основі вхідної або надавати оцінку заданій парі вхідних і вихідних послідовностей. У процессі навчання кодувальник і декодувальник навчаються як єдина мережа на основі пар вхідних та вихідних послідовностей.

Механізм уваги дозволяє визначити, як елементи двох послідовностей (наприклад, слова у реченнях) пов’язані між собою [17]. Початково він був розроблений для архітектури кодувальник-декодувальник для вирішення проблеми втрати даних у проміжній послідовності, коли вхідна послідовність має велику довжину, для чого шар уваги додавався між підмережами кодувальника та декодувальника, але надалі цей механізм виявився корисним і для інших застосувань у обробці природньої мови.

Самоувага (або також внутрішня увага) – це механізм уваги, що співвідносить різні позиції єдиної послідовності з метою розрахунку представлення даної послідовності. Самоувага знайшла своє використання у багатьох задачах розуміння природньої мови, в тому числі таких як розуміння тексту (reading comprehension), абстрактне узагальнення (abstractive summarization) і знаходження зв’язку між послідовними реченнями (textual entailment) [9]. Окрім цього, даний механізм також набув важливу роль як допоміжний у задачах, де постає проблема знаходження представлення заданого речення або речень [17].

Загальним недоліком підходів на основі згорткових та рекурентних нейронних мереж є складність їх навчання через обмежені можливості до паралелізації. В той же час, архітектура кодувальник-декодувальник та механізм уваги гарно себе зарекомендували, тому на їх основі було створено архітектуру Transformer [9]. Ця архітектура використовує підхід з кодувальником та декодувальником, але замість рекурентних шарів в ній використовуються виключно повнозв’язні шари та шари уваги, що дозволило спростити процес навчання і, як результат, покращити точність роботи моделей, що розроблені з використанням даної архітектури.

Однією з найважливіших таких моделей є BERT, що використовує

Transformer та підтримує попереднє навчання з використанням підходу, коли елементи вхідної послідовності випадково маскуються, а модель має передбачити, який саме елемент було замасковано. На відміну від попередніх моделей, таких підхід дозволяє знаходити взаємозв'язки у обох напрямках – як зліва направо, так і навпаки. Як результат, попередньо навчена модель BERT може бути додатково налаштована (fine-tuning) за допомогою єдиного вихідного шару для її використання у великій кількості різних задач без необхідності вносити суттєві зміни у загальну архітектуру моделі [18]. Це дозволило моделі BERT покращити результати у багатьох задачах обробки природньої мови, у тому числі таких як машинний переклад та знаходження відповідей на питання [19].

Задачі класифікації наміру та маркування слотів є типовими задачами розуміння природньої мови і тому для їх вирішення використовуються ті ж підходи, що згадані вище для галузі у цілому [20]. При цьому для випадку задачі заповнення форми, коли необхідно вирішити обидві задачі одночасно, існують два підходи до їх розв'язання:

- розв'язувати їх як дві окремі задачі. Для цього створюються дві окремі моделі, які ніяк не пов'язані одна з одною, що дозволяє використовувати різні підходи для їх розв'язання ціною ускладнення рішення у цілому [21, 22];
- розв'язувати їх одразу як об'єднану задачу за допомогою єдиної моделі. Даний підхід дозволяє спростити системи розуміння природньої мови оскільки необхідно навчати і налаштовувати лише одну модель [23, 24, 25].

Перший підхід активно застосовувався до розповсюдження ефективних моделей глибокого навчання у галузі обробки природньої мови. До цього задачі класифікації наміру та маркування слотів вирішувалися засобами різних інструментів і тому не могли бути об'єднані у єдину задачу та модель її вирішення. Недоліком цього підходу є те, що через незалежність моделей помилки мають тенденцію розповсюджуватися всередині цих моделей.

Другий підхід почав використовуватися у рішеннях цієї задачі, що базуються

на рекурентних нейронних мережах з використанням архітектури кодувальник-декодувальник. Загальна ідея полягає у тому, що до моделі додаються шари класифікації після декодувальника, один з яких відповідає за визначення наміру, а інший – за маркування слотів. При такому підході функція втрат задається як комбінація функцій втрат для обох підзадач, а для реалізації кодувальників та декодувальників використовуються LSTM [26] та BiRNN [27].

Однак, при такій реалізації об’єднаної задачі в моделі відсутні явні зв’язки між наміром та слотами цього наміру, хоча фактично слоти сильно залежать від наміру. Цю проблему спробували вирішити у архітектурі з вентилем для слотів (slot-gated) [28]. Дана модель побудована на основі рекурентної мережі з двонаправленою довгою короткочасною пам’яттю і використовує механізм уваги у поєднанні з спеціальним вентилем, який поєднує вихід шару для знаходження наміру з шаром знаходження послідовності маркувань (див. рис. 2.3).

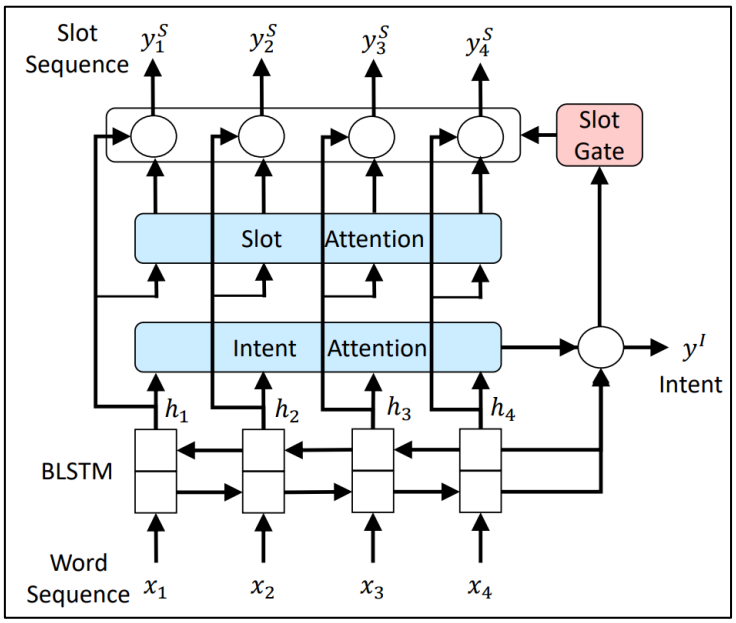

Рисунок 2.3 – Структура моделі з вентилем для слотів [28]

З появою BERT переваги другого підходу значно переважають недоліки у вигляді складності отриманої моделі, тому всі сучасні підходи до розв’язання задачі заповнення форм побудовані на основі розв’язання об’єднаної задачі класифікації наміру та маркування слотів [23].

2 РОЗРОБКА МОДЕЛІ ПРИРОДНЬОЇ МОВИ

2.1 Визначення та постановка задачі

Задача заповнення форм у чат-ботах полягає у визначення для заданого тексту природньою мовою типу форми, яку необхідно заповнити (намір користувача), та значення, якими необхідно заповнити поля форми даного типу (маркування слотів).

Будемо називати вхідний текст реченням, тоді речення S є кінцевою послідовністю слів W, причому довжина послідовності n є довільною, а всі символи, що не входять до алфавіту A заданої природньої мови, вважаються роздільниками між словами:

$$\begin{gathered} S = (W_1, W_2, \dots, W_n), \\ W = (a_1, a_2, \dots, a_k), a_i \in A, i \in [1, k] \end{gathered} \tag{3.1}$$

Нехай можливі наміри складають кінцеву множину I певного розміру m. Позначимо намір користувача для речення S як I_S і визначимо функцію *Intent* як таку, що її значення для заданого речення відповідає наміру користувача у цьому реченні (*U(S)* позначає множину усіх можливих речень):

$$Intent(S) = I_S, I_S \in I, \forall S \in U(S) \tag{3.2}$$

Нехай типи значень форми (далі – слоти) для заданого наміру I_i задаються як множина $L(I_i)$, при чому ця множина завжди включає в себе елемент, що відповідає випадку, коли слово не є допустимим значенням форми. Визначимо функцію *SlotLabels* як таку, що для заданого речення S її значення є послідовністю маркувань L_S для слів цього речення:

$$SlotLabels(S) = L_S = (l_1, l_2, \dots, l_n), l_i \in L(I_S), i \in [1, n] \tag{3.3}$$

Задача даної роботи тоді полягає у знаходженні таких апроксимацій функцій *Intent* та *SlotLabels*, що є якомога точнішими, при попередньо відомому наборі вхідних та вихідних даних *D* розміром *s*:

$$D = (P_1, P_2, \dots, P_s),$$
$$P_i = \{S_i, I_{S_i}, L_{S_i}\} \tag{3.4}$$

2.2 Опис загальної структури моделі

Архітектура нейромережевої моделі є одним з варіантів архітектури BERT, яка в свою чергу є архітектурою Transformer. Розглянемо їх більш детально. Як вже було згадано раніше, архітектура Transformer реалізує структуру кодувальник-декодувальник. Кодувальник E перетворює вхідну послідовність символів у неперервну послідовність, а декодувальник D генерує з неї вихідну послідовність символів, перетворюючи за раз один її елемент, причому довжина цієї послідовності може відрізнятися від довжин вхідної та проміжної послідовностей:

$$x = (x_1, x_2, \dots, x_n),$$
$$E: x \to z, z = (z_1, z_2, \dots, z_n), \tag{3.5}$$
$$D: z \to y, y = (y_1, y_2, \dots, y_m),$$

де x – вхідна послідовність символів довжиною n;

z – неперервна проміжна послідовність елементів;

y – вихідна послідовність символів довжиною m.

На кожному кроці дана модель є авторегресійною – попередньо згенеровані символи використовуються в якості додаткових вхідних даних для генерації наступних [9].

В архітектурі Transformer дані блоки складаються з двох типів шарів (див.

рис. 3.1):

– шари з самоувагою;

– поточкові повнозв’язні шари.

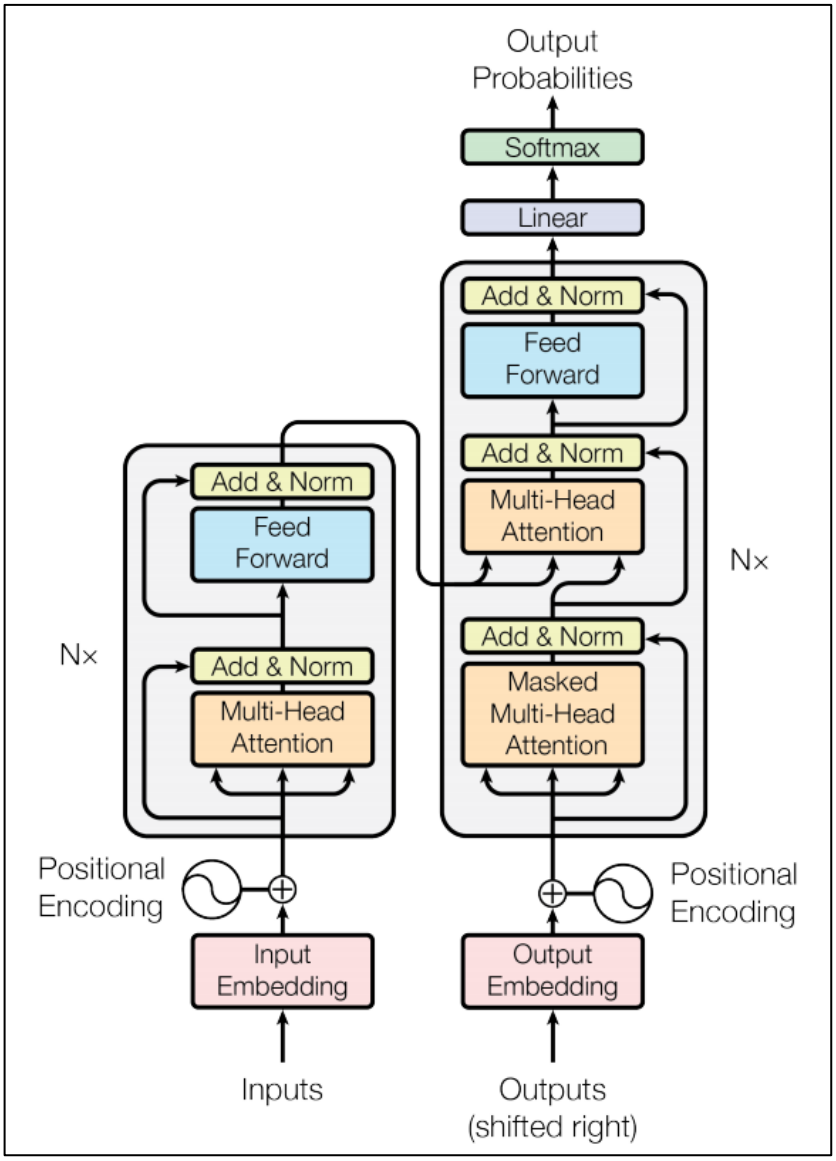

Рисунок 3.1 – Архітектура моделі Transformer [9]

Кодувальник складається зі N однакових шарів. У кожному шарі є два підшари. Перший - це різноспрямований механізм самоуваги (multi-head self-attention), а другий – це нейронна мережа прямого поширення. Навколо кожного з підшарів використовується залишкове з’єднання з подальшою нормалізацією отриманого значення в цьому шарі. Тоді кожен шар кодувальника можна описати як:

$$EncoderLayer(x) = FeedForwardSublayer(AttentionSublayer(x)),$$
$$FeedForwardSublayer(x) = LayerNorm\big(x + FFN(x)\big), \quad (3.6)$$
$$AttentionSublayer(x) = LayerNorm\big(x + SequenceAttention(x)\big)$$

де *EncoderLayer* – функція шару кодувальника;

FeedForwardSublayer – функція повнозв'язного підшару;

AttentionSublayer – функція підшару з увагою;

LayerNorm – функція нормалізації значення;

FFN – функція, що відповідає мережі прямого поширення;

SequenceAttention – функція розрахунку уваги для послідовності.

Декодувальник також складається з N однакових шарів. На додаток до двох підшарів у кожному шарі кодувальника, у декодувальнику наявний також третій підшар, який розраховує увагу над виходом кодувальника. Як і в кодувальнику, навколо кожного з підшарів використовується залишкове з'єднання з подальшою нормалізацією шару. Водночас, підшар самоуваги в декодувальнику використовує також додаткове маскування, щоб запобігти появі позицій послідовності у наступних позиціях. Таке маскування, в поєднанні зі зсувом вхідних даних на одну позицію, гарантує, що прогнози для позиції можуть залежати лише від відомих результатів у позиціях, що знаходяться перед нею.

Тепер розглянемо детальніше механізм уваги, що використовується у даній архітектурі. Функція уваги може бути описана як відображення запиту та набору пар «ключ – значення» на вихідні дані, де запит, ключі, значення та вихідні дані є векторами. Вихідне значення обчислюється як зважена сума значень, де вага, присвоєна кожному значенню, обчислюється функцією сумісності запиту з відповідним ключем.

На практиці увага обчислюється водночас для набору запитів та відповідними для них ключами та значеннями з використанням відповідних матричних операцій (див. формулу 3.7). Як видно з даної формули, розмірності матриць у такому випадку мають бути сумісними через використання матричного множення.

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (3.7)$$

де Q – матриця запитів;

K – матриця ключів;

V – матриця значень;

d_k – розмірність запитів та ключів.

В даному випадку *softmax* є матричним варіантом нормованої експоненційної функції [29]. Кожний елемент векторів в результуючій матриці обчислюється як:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (3.7)$$

де z – вектор дійсних чисел розмірності K.

Така реалізація функції уваги відрізняється від загальновживаної лише наявністю додаткового масштабуючого значення пропорційного до значення d_k. Таке масштабування базується на тому, що при великих значеннях d_k поточкові добутки зростають за величиною, підштовхуючи функцію *softmax* до регіонів, де вона має надзвичайно малі градієнти, що в свою чергу ускладнює процес навчання результуючої мережі.

Крім цього, при включенні даної функції уваги до шарів мережі використовується підхід з різноспрямованою самоувагою. Його ідея полягає у тому, що замість обробки запитів, ключів та значень вихідного розміру задається h проекцій цих значень у простір меншої розмірності. Значення уваги розраховується паралельно для всіх цих проекцій, після чого вони об’єднуються в єдине значення, яке проектується у вихідне значення такого ж розміру, що і початкова модель (див. формулу 3.8). Оскільки кожна проекція має меншу розмірність, ніж вихідна модель, обчислювальна складність залишається майже незмінною, але при цьому наявність окремих проекція робить мережу більш гнучкою. Кількість проекцій в даному випадку є гіперпараметром моделі.

$$MultiHead(Q,K,V) = Concat(head_1, head_2, \dots, head_h)W^O,$$
$$head_i = Attention\left(QW_i^Q, KW_i^K, VW_i^V\right), i = [1,h], \quad (3.9)$$
$$W_i^Q \in \mathbb{R}^{d_{model} \times d_k}, W_i^K \in \mathbb{R}^{d_{model} \times d_k}, W_i^V \in \mathbb{R}^{d_{model} \times d_v}, W_i^O \in \mathbb{R}^{d_{model} \times hd_v}$$

де W – матриці проекцій для запитів, ключів, значень та вихідних значень;

d_v – розмірність значення для уваги;

d_{model} – внутрішня розмірність моделі;

h – загальна кількість проекцій.

В архітектурі Transformer механізм уваги використовується трьома різними способами:

– у шарах уваги між кодувальником та декодувальником запити надходять із попереднього шару декодувальника, а ключі та значення надходять з виходу кодувальника. Це дозволяє кожній позиції в декодувальнику відвідувати всі позиції вхідної послідовності і імітує аналогічні механізми уваги у архітектурах кодувальник-декодувальник, заснованих на рекурентних або згорткових мережах;

– кодувальник містить шари самоуваги. У шарі самоуваги всі ключі, значення та запити надходять з одного й того ж самого місця, а сама з вихідних значень попереднього шару у кодувальнику;

– шари самоуваги в декодувальнику дозволяють кожній позиції в ній відвідувати всі позиції до неї, включно з нею самою. Для запобігання потоку інформації зліва в декодувальнику (що дозволяє забезпечити властивість авторегресії) всередині функції уваги, маскуючи всі значення на вході функції *softmax*, що відповідають некоректним з’єднанням, шляхом встановлення їх як мінус безкінечність.

Окрім підшарів з увагою, кожен з шарів у кодувальнику та декодувальнику включає також повнозв’язну нейромережу прямого поширення, яка застосовується до кожної позиції окремо та ідентично. В той же час, в залежності від шару значення параметрів є різними. Цей підшар задається як:

$$FFN(x) = \max(0, xW_1 + b_1)\, W_2 + b_2 \tag{3.10}$$

де W_1, W_2 – матриці ваг;

b_1, b_2 – вектори зсувів.

Конкретні значення гіперпараметрів даної архітектури (кількість шарів у блоках кодувальника та декодувальника, внутрішній розмір моделі та кількість проекцій для розрахунку уваги) обираються згідно потребам конкретної задачі.

Оскільки архітектура BERT, що описана вище, є базовою та не може використовуватися для конкретних задач у такому вигляді, розширимо її для об'єднаної задачі класифікації наміру та маркування слотів.

Для цього додамо до моделі два паралельні шари після останнього шару декодувальника. Перший шар буде відповідати за вирішення задачі класифікації наміру. В моделі BERT спеціально для класифікації вводиться спеціальний токен [CLS] на першій позиції речення, і для отримання ймовірностей для всіх наявних класів можна використати наступний шар:

$$\begin{aligned} P(I) &= softmax(h_1 W_I + b_I) \\ Intent^*(S) &= argmax(P(I)) \end{aligned} \tag{3.11}$$

де W_I – матриця ваг для шару класифікації наміру;

h_1 – внутрішній стан першого елементу вихідної послідовності BERT;

b_I – вектор зсуву для шару класифікації наміру;

P(I) – вектор ймовірностей кожного з намірів для даного речення;

Intent(S)* – передбачене значення наміру для даного речення.

Другий шар відповідає за знаходження маркувань слотів. Для цього кожен елемент вихідної послідовності, за винятком першого, що відповідає спеціальному токену у вхідній послідовності, необхідно класифікувати за відомими типами слотів та отримати відповідні ймовірності для кожного з цих типів (див. формулу 3.12). З цих ймовірностей ми можемо нарешті виділити маркування для всіх елементів вхідної послідовності.

$$P_i(L) = softmax(h_i W_L + b_L), i = 2, \dots, n,$$
$$l^*_{i-1} = argmax(P_i(L)) \quad (3.12)$$
$$SlotLabels^*(S) = (l^*_1, l^*_2, \dots, l^*_n)$$

де W_L – матриця ваг для шару маркування слотів;

h_i – внутрішній стан і-го елементу вихідної послідовності BERT;

b_L – вектор зсуву для шару маркування слотів;

$P_i(I)$ – вектор ймовірностей кожного з типів слотів для даного токена;

l^* – передбачений тип слоту для даного токену;

$SlotLabels^*(S)$ – передбачене маркування слотів для даного речення.

Структуру з двома додатковими шарами приймемо за базову і додатково до неї розглянемо два елементи, які можна використати для покращення точності маркування слотів:

– шар з умовним випадковим полем;

– вентильний вузол для слотів.

Умовні випадкові поля є класом методів, які дозволяють виконувати класифікацію з урахуванням контексту за рахунок використання графової структури. Використання умовних випадкових полів гарно зарекомендувало себе у вирішенні деяких задач обробки природньої мови [5], що робить актуальним перевірку їх придатності в якості додаткового інструменту при вирішенні задачі заповнення форм [30].

В даній моделі шар з умовним випадковим полем заміщує собою шар softmax, що відповідає за знаходження маркувань слотів, а шар для класифікації намірів залишається без змін.

Вентильний вузол для слотів (див. рис. 3.2) базується на тому, що маркування слотів безпосередньо залежать від наміру користувача (різним намірам відповідають різні набори можливих слотів). Виходячи з цього, передбачене значення класу наміру зв’язується з вихідними значеннями моделі BERT для токенів у реченні. Вихідне значення вузла множиться зі значенням моделі та подається на вхід softmax шару для класифікації.

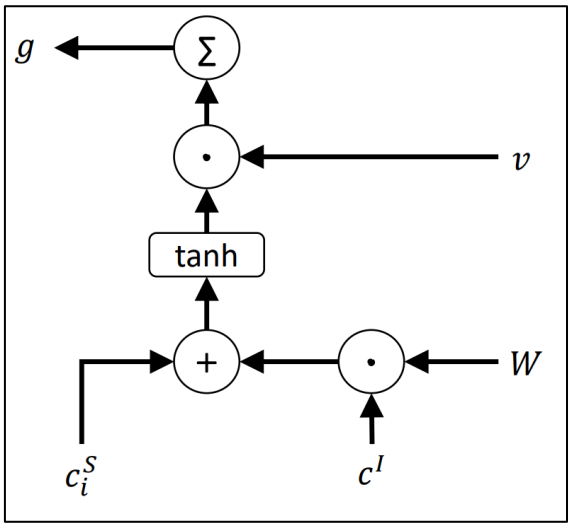

Рисунок 3.2 – Вентильний вузол для слотів

Використовуючи введені раніше позначення, структуру шару з вентильним вузлом для слотів можна описати формулою 3.13. Так само як і для шару з умовним випадковим полем, шар класифікації наміру залишається без змін.

$$g = \sum_i v \cdot \tanh(h_i + W_g h_1)$$
$$P_i(L) = softmax(h_i g W_L + b_L), i = 2, \dots, n \quad (3.13)$$

де v – вектор, значення якого знаходяться під час навчання;

W_g – матриця ваг для вентильного вузла;

g – коефіцієнт впливу наміру на маркування слотів;

h_i – внутрішній стан і-го елементу вихідної послідовності BERT;

W_L – матриця ваг для шару маркування слотів;

b_L – вектор зсуву для шару маркування слотів;

$P_i(I)$ – вектор ймовірностей кожного з типів слотів для даного токена;

З урахуванням цього в якості задачі навчання оберемо мінімізацію значення перехресної ентропії між передбаченими наміром або типом слоту та правильним наміром або типом слоту. Значення перехресної ентропії збільшується тим більше, чим далі передбачена ймовірність від реального значення [31], вона задається як:

$$Loss(y,p) = -\sum_{c=1}^{M} y_c \log p_c \quad (3.14)$$

де M – кількість класів;

y –дорівнює 1 коли спостереження відноситься до класу c, інакше 0;

p – передбачена ймовірність класу c.

Оскільки в задачі заповнення форми одночасно розв’язуються дві підзадачі, результуюча функція втрат може бути задана як сума втрат у задачі класифікації наміру та маркування слотів:

$$Loss = \alpha \times Loss_I + \beta \times Loss_L \quad (3.15)$$

де α, β – довільні гіперпараметри моделі.

2.3 Перетворення вхідних даних

Архітектура моделі BERT повністю повторює архітектуру Transformer, але вводить додатково спеціальний процес підготовки вхідних даних.

Оскільки нейронна мережа оперує чисельними значеннями, речення природньою мовою має бути перетворено у векторне представлення (embedding) [32]. В архітектурі BERT використовується комбінація трьох таких представлень для слів у реченні:

– представлення токенів;

– сегментне представлення;

– порядкове представлення.

Метою знаходження представлення токенів є перетворення слів речення природньою мовою у векторне представлення фіксованої розмірності. Для цього речення спочатку розбивається на токени, причому в BERT використовуються два допоміжні токени – токен [CLS] відображає все речення для подальшої

класифікації, а токен [SEP] позначає кінець речення.

Для розбиття речення на токени використовується метод WordPiece. Цей метод є детермінованим і засновується виключно на даних речення, його ідея полягає у тому, що слова речення розбиваються на частини (wordpiece) на основі готової моделі таких частин. Такий підхід дозволяє досягнути балансу між гнучкістю представлень на основі окремих букв та ефективністю представлення на рівні слів, оскільки мовний корпус є значно меншим за корпус всіх слів певної мови, однак при цьому несе інформацію про поширені блоки слів, на відміну від окремих букв [33].

Після розбиття речення на токени спеціальний шар мережі знаходить відповідне цій послідовності токені векторне представлення фіксованої розмірності (у моделі BERT вона дорівнює внутрішньому розміру моделі).

BERT здатний вирішувати задачі NLP, які включають класифікацію тексту за даною парою вхідних текстів. Прикладом такої проблеми є класифікація того, чи є два тексти семантично схожими. Для цього обидва тексти об'єднуються і подаються в модель, а для їх подальшого розрізнення використовується представлення сегментів.

В даному шарі наявні лише два можливі векторні представлення – представлення А відповідає першому реченню, представлення В – другому. Конкретні значення цих представлень знаходяться під час навчання моделі, їх розмірність співпадає з розмірністю представлення токенів.

Для задач з обробки природньої мови є характерною наявність послідовної залежності між елементами вхідних даних, оскільки слова в реченнях часто залежать від попередніх слів у цьому або попередніх реченнях. В підходах, що базуються на рекурентних нейронних мережах, ця залежність вбудована в саму архітектуру мереж [34]. Однак в архітектурі Transformer вона відсутня, тому в ній для цього використовується спеціальний механізм, що кодує вихідну позицію слова у реченні – позиційне векторне представлення слова.

Існують два підходи до знаходження позиційного представлення. Перший з них базується на фіксованих функціях, що генерують представлення для заданої

позиції (див. формулу 3.16). Такий вибір функцій зумовлений тим, що через їх періодичний характер модель має змогу виділяти відносне положення елементу у послідовність.

$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$
$$PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right) \quad (3.16)$$

де *pos* – положення слова у реченні;

i – індекс комірки у векторі результуючого представлення.

Другий підхід використовує представлення, що генеруються під час навчання моделі. Кожній позиції слова у реченні відповідає своє представлення, яке не залежить від того, яке саме це слово [35]. На практиці цей підхід призводить до збільшення кількісті параметрів мережі, що негативно впливає на швидкість навчання та роботи, але може покращувати точність роботи на певних задачах [19].

Після знаходження всіх трьох окремих представлень результуюче представлення знаходиться як їх сума для кожного окремого слова у реченні:

$$Embedding(S_i) = Token(S_i) + Segment(S_i) + Positional(S_i)$$
$$i = 1, \dots, n \quad (3.17)$$

де S_i – слово речення;

Token – функція знаходження представлення токену;

Segment – функція знаходження представлення сегменту;

Positional – функція знаходження позиційного представлення.

Приклад знаходження представлення для текстових вхідних даних у BERT зображено на рис. 3.3. Тут на вхід подається попередньо токенізоване речення (на практиці обидва кроки реалізують як один модуль) у форматі сумісному з BERT, після чого для нього генерується чисельне (векторне) представлення за алгоритмом, що наведено вище.

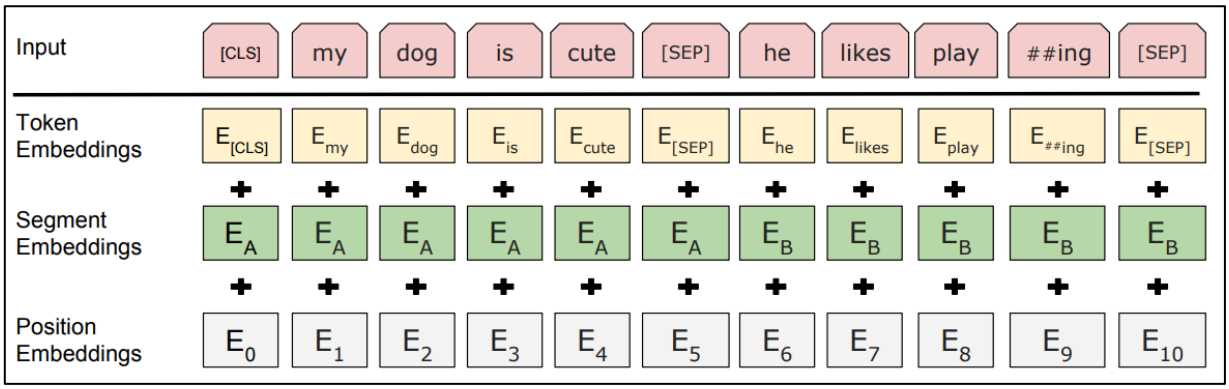

Рисунок 3.3 – Приклад знаходження представлення речення [19]

Перетворене таким чином у шарі векторних представлень речення далі подається на вхід першому шару кодувальника.

2.4 Альтернативні моделі BERT

Попри ефективність моделі BERT при вирішенні задач обробки природньої мови, їй притаманні також і певні недоліки, а саме велика кількість параметрів і, відповідно, обчислювальна складність при навчанні, особливо за неможливості використання попередньо навчених моделей (наприклад, при вирішенні певної задачі для мови, для якої не існує готових моделей, або якщо специфіка задачі не дозволяє їх використовувати). Крім того, фізичний розмір та швидкість роботи готових моделей не дозволяє використовувати BERT на мобільних пристроях. Виходячи з цих обмежень, було створено цілу низку оптимізованих моделей, серед яких особливо виділяються ALBERT та DistillBERT.

В ALBERT для оптимізації використовується дві техніки – факторизована параметризація представлень та спільне використання параметрів [36].

Факторизована параметризація представлень полягає у тому, що єдина матриця для побудови представлення розбивається на дві окремі матриці меншого розміру. Це дозволяє зменшити кількість параметрів, які необхідні для побудови представлення для вхідних даних. При цьому точність моделі не втрачається,

оскільки втрачені при розбитті контекстно-залежні параметри представлення все одно є частиною безпосередньо мережі, що навчається.

В свою чергу при спільному використанні параметрів між певними шарами значення параметрів повторюються. За замовчуванням в ALBERT повторюються значенням всіх шарів архітектури Transformer.

Результати використання ALBERT для поширених задач обробки природньої мови показують, що у порівнянні з BERT моделі мають значно меншу кількість параметрів при кращій точності роботи, але є складнішими для обчислення [36].

DistillBERT базується на підході дистиляції моделі. Дистиляція полягає у тому, що компактна модель тренується відтворювати поведінку більшої моделі або цілого ансамбля моделей [37].

У випадку DistillBERT в ролі більшої моделі виступає безпосередньо оригінальна модель BERT, а в ролі меншої – модель з архітектурою BERT, але з меншою в 2 рази кількістю шарів; менша модель при цьому ініціалізується параметрами шарів у вихідній моделі. Використання DistillBERT для практичних задач показує, що такий підхід дозволяє зменшити розмір мережі та швидкість її обчислення без суттєвих втрат у точності [38].

2.5 Оцінка якості моделі

Оцінку якості розробленої моделі будемо проводити на двох наборах даних, що використовуються у якості бенчмарків для задач класифікації наміру та маркування слотів – ATIS [39] та Snips [40].

Набір даних ATIS (Airline Travel Information System – інформаційна система для авіа подорожей) являє собою набір запитів до системи для отримання інформації з предметної області авіалінії (наприклад, дані про наявні рейси, ціну та доступність квитків тощо). Кожен запит містить відомі намір запиту та маркування слотів на його основі (див. рис. 3.4).

Intent	Text
flight	on [april]*depart_date.month_name* [first]*depart_date.day_number* i need a flight going from [phoenix]*fromloc.city_name* to [san diego]*toloc.city_name*
abbreviation	what does fare code [bh]*fare_basis_code* mean
airfare	give me the fares from [miami]*fromloc.city_name* to [cleveland]*toloc.city_name* [next]*depart_date.date_relative* [sunday]*depart_date.day_name*

Рисунок 3.4 – Приклад даних з набору ATIS

Цей набір даних включає в себе 4478 запитів для навчання, 500 для валідації та 893 для тестування; він містить 120 можливих маркувань слотів та 21 намір; класи намірів і, відповідно, маркувань слотів, є сильно незбалансованими.

Набір даних Snips містить колекцію команд, що зазвичай використовуються у сценаріях голосового помічника. Для кожної команди задано її тип та текст з маркуваннями для параметрів цієї команди (див. рис. 3.5). Цей набір включає в себе 13084 команд для навчання, 700 для валідації та 700 для тестування; всього наявні 72 можливих маркувань слотів і 7 намірів.

Command	Text
GetWeather	give me the weather forecast for [los angeles]*weatherLocation* [this weekend]*weatherDate*
TurnOnLight	The [bedroom]*room* is dark, please turn on the light!
SearchFlight	find me a flight from [Paris]*origin* to [New York]*destination*

Рисунок 3.5 – Приклад даних з набору Snips

Для оптимізації отриманої нейронної мережі скористаємося оптимізатором Adam [41], що використовується і у базових архітектурах Transformer та BERT. На відміну від звичайного алгоритму стохастичного градієнтного спуску, цей алгоритм має змінну швидкість навчання (learning rate), яка базується на оцінка моментів функції, що оптимізується.

Для попередження перенавчання та перевпевненості моделі і покращення її роботи у реальних умовах скористаємося двома техніками регулярізації:

– виключення (dropout);

– згладжування маркувань (label smooting).

Метод виключення полягає у тому, що під час навчання з мережі виключається певна задана кількість нейронів, причому вибір цих нейронів є випадковим. Окрім запобіганню перенавчанню, цей метод дозволяє також покращити швидкість, з якою відбувається навчання [42].

Метод згладжування маркувань застосовується у мережах, в яких у якості функції втрат використана перехресна ентропія у поєднанні з функцією softmax у шарах мережі. Ідея цього методу регулярізації полягає у тому, що отриманий вектор ймовірностей змішується з рівномірним розподілом (див. формулу 3.18). Це дозволяє зменшити кількість ситуацій, коли передбачена ймовірність деякого класу значно перевищує загальну точність моделі [43].

$$\tilde{y} = (1 - \alpha)y + \frac{\alpha}{K} \qquad i = 1, \dots, n \tag{3.18}$$

де y – вектор ймовірностей;

α – гіперпараметр, що визначає ступінь зглажування;

K – загальна кількість класів.

Оскільки модель природньої мови, що розробляється, базується на моделі BERT або її оптимізованих варіантах, то ці дві техніки регулярізації застосовуються на рівні шарів, що додані після базової моделі, а саме шарах класифікації наміру та маркувань слотів.

Нарешті, для оцінки точності роботи моделі на рівні окремих підзадач будемо використовувати дві метрики – точність (accuracy) та F1-оцінку – для результатів класифікації наміру та маркування слотів відповідно.

Точність є метрикою того, наскільки передбачені значення співпадають з правильними для заданого набору даних. Оскільки в даному випадку класифікація наміру працює з довільною кількістю класів (багатокласова класифікаці), то точність розраховується за формулою 3.19. Вибір цієї метрики зумовлений простотою її інтерпретації та адаптованістю до багатокласової класифікації.

$$Accuracy = \frac{correct}{total} \tag{3.19}$$

де *correct* – кількість вхідних речень, для яких намір передбачено правильно моделлю;

total – загальна кількість речень у наборі даних;

Accuracy – значення точності для заданої моделі та набору даних.

В свою чергу, метрика F1 використовується у випадках, коли класи в рамках набору даних є сильно незбалансованими [44]. Як було зазначено вище, використані набори даних ATIS та SNIPS є сильно незбалансованими з точки зору наявних маркувань слотів, що і зумовлює вибір цієї метрики для даної підзадачі.

Для розрахунку метрики F1 у випадку наявності довільної кількості класів спочатку будується матриця помилок (confusion matrix), в якій рядкам відповідають передбачені класи, а стовпцям – коректні класи, відповідно матриця є квадратною. Значення елементу матриці відповідає кількості елементів в наборі даних, для яких передбачений та справжній класи відповідають рядку та стовпцю цього елементу. В такому випадку сума значень на діагоналі відповідає правильним значенням у моделі, сума значень у рядку за виключенням елементу на діагоналі – помилкам першого роду, у стовпцю – помилкам другого роду. З цих значень обчислюються значення точності (precision) та повноти (recall), а метрика F1 тоді розраховується як гармонічне середнє цих двох величин.

Додатково до цих метрик також застосуємо загальну метрику для сумісної задачі заповнення форми. Ця метрика дорівнює 1 для елементу даних, якщо намір та всі слоти в реченні передбачено коректно, інакше 0. Значенням метрики для всього набору даних тоді буде середнє її значення між усіма елементами набору.

3 АНАЛІЗ РЕЗУЛЬТАТІВ РОБОТИ МОДЕЛІ

3.1 Технології реалізації моделі

Для реалізації архітектури моделі природньої мови, що описана у попередньому розділі, було обрано мову програмування Python та фреймворк машинного навчання PyTorch. Цей вибір зумовлений зрілістю цих інструментів для їх застосування у машинному навчанні в цілому та побудові нейронних мереж зокрема за рахунок високого рівня абстракції без втрат у швидкості навчання на передбачення [45].

В якості базових попередньо навчених моделей було використано наступні:

– bert-base-uncased, що реалізує модель BERT з 12 шарами і має загалом 110 мільйонів параметрів;
– albert-xxlarge-v1, що реалізує модель ALBERT з 12 шарами і має 220 мільйонів параметрів;
– distilbert-base-uncased, що реалізує модель DistilBERT з 6 шарами і має загалом 66 мільйонів параметрів.

Для всіх базових моделей використані варіанти моделей без врахування регістру символів у словах речення, оскільки для задачі заповнення форм він не несе суттєвої додаткової інформації, але при цьому вимагає додаткових обчислювальних ресурсів. За необхідності можуть бути використані і інші готові базові моделі за умови їх сумісності з форматом даних у BERT.

Для всіх моделей в усіх експериментах використовувались однакові параметри навчання – всього 10 епох, з яких обирається модель з найкращими показниками на валідаційному наборі даних. Значення всіх гіперпараметрів було зафіксовано, за виключенням їх налаштування у підрозділі 3.4.

Всі показники часу навчання вказані для навчання з використанням CUDA на графічному процесорі Nvidia GTX 1050; в якості розміру моделей вказано фізичний розмір файлів точки відновлення (checkpoint) моделі у форматі фреймворку PyTorch.

3.2 Експерименти з архітектурою мережі

На основі запропонованих архітектур додаткових шарів мережі, було обрано такі їх комбінації для проведення дослідження ефективності їх роботи:

– Baseline – базова архітектура з двома лінійними softmax шарами;
– CRF – архітектура, в якій softmax шар для класифікації маркувань слотів замінено на шар з умовним випадковим полем;
– SlotGate – архітектура, в якій softmax шар для класифікації маркувань слотів використовує спеціальний вентильний вузол замість лінійного перетворення;
– SlotGate+CRF– архітектура, в якій скомбіновано підходи з архітектур SlotGate та CRF.

В якості базової моделі було обрано оригінальну реалізацію BERT без додаткових оптимізацій (їх буде розглянуто окремо у наступному підрозділі), використовувалися значення за замовчуванням для гіперпараметрів.

Окрім введених метрик для поставлених задач було також додатково оцінено технічні параметри даних архітектур, а саме час навчання (середня тривалість однієї епохи навчання) та передбачення (середній час роботи на тестовому наборі даних між 10 запусками). Результати експериментів для набору даних ATIS наведено у таблиці 1.

Таблиця 1 – Результати роботи моделей різних архітектур для набору ATIS

Архітектура	Точність наміру	Метрика F1 для слотів	Точність для речення	Час навчання, хв	Час передбачення, с
Baseline	0.978	0.955	0.875	1.835	6.120
CRF	0.973	0.960	0.878	2.320	10.986
SlotGate	0.972	0.956	0.877	2.151	7.050
SlotGate+CRF	0.974	0.956	0.880	2.618	11.203

Можна бачити, що всі варіанти архітектур показують близькі показники метрик точності та F1 для тестового набору даних, причому в усіх випадках точність класифікації наміру перевищує метрику F1 та, особливою мірою, точність на рівні всього речення. З точки зору загальної точності для об’єднаної задачі найкращий результат показує архітектура з використанням комбінації вентильного вузла та шару з умовним випадковим полем для класифікації; в той же час, ця архітектура потребує більше часу для навчання та передбачення.

Для набору даних SNIPS (див. табл. 2) результати є подібними – метрики точності майже співпадають, незначною мірою за всіма показниками є оптимальною та ж сама архітектура, що і для набору даних ATIS.

Таблиця 2 – Результати роботи моделей різних архітектур для набору SNIPS

Архітектура	Точність наміру	Метрика F1 для слотів	Точність для речення	Час навчання, хв	Час передбачення, с
Baseline	0.984	0.962	0.919	4.423	4.720
CRF	0.984	0.959	0.911	7.272	8.851
SlotGate	0.983	0.962	0.919	6.620	5.138
SlotGate+CRF	0.984	0.963	0.920	7.513	9.437

З проведених експериментів можна зробити наступні висновки:

– всі запропоновані варіанти архітектури мережі загалом показують близькі значення метрик точності наміру, точності для речення та F1 для маркування слотів, з них можна виділити архітектуру SlotGate+CRF як оптимальну для обох наборів даних;

– міра впливу конкретної обраної архітектури на точність роботи моделі залежить від набору даних, з яких працює модель, тому вибір архітектури є доцільним робити з огляду на конкретну прикладну задачу, для вирішення якої планується використовувати модель.

3.3 Експерименти з базовою моделлю BERT

Як вже було зазначено вище, в якості можливих базових моделей BERT було обрано три альтернативні підходи до її реалізації. Проведемо дослідження впливу вибору конкретної базової моделі на метрики точності, а також на додаткові технічні параметри – вже зазначені раніше час навчання та передбачення, а також на фізичний розмір навченої мережі. Такий вибір зумовлений тим, що ці характеристики можуть впливати на сценарії, в яких можливе використання готової моделі.

Наприклад, мережа, що працює з високою точністю, але є повільною і має великий розмір, не може бути легко використана в якості компоненту додатків для мобільних пристроїв. І навпаки, швидка та компактна модель може програвати більш складним, що робить її менш корисною у ситуаціях, коли навіть невелика різниця у показниках точності може суттєво визначати придатність моделі у цілому для використання (такі як фінансова сфера, область медичних даних або інші галузі з високим ризиком у випадку помилки перебачення).

Розпочнемо з оцінки зазначених параметрів на наборі даних ATIS (див. табл. 3 та 4). З точки зору точності роботи моделей BERT і ALBERT є дуже близькими, в той час як DistilBERT дещо програє за усіма метриками, що є очікуваним виходячи з внутрішньої структури даної моделі, оскільки головною метою при її розробці було зменшити розмір моделі та час навчання без суттєвого впливу на точність роботи [37].

Таблиця 3 – Результати роботи моделей різних типів для набору ATIS

Базова модель	Точність наміру	Метрика F1 для слотів	Точність для речення
BERT	0.978	0.955	0.875
ALBERT	0.976	0.956	0.879
DistilBERT	0.973	0.951	0.865

В той же час, за технічними параметрами моделі суттєво відрізняються, їх відмінності повністю відповідають специфікаціям даних моделей – DistilBERT є найшвидшою та найкомпактнішою моделлю, оригінальний BERT займає середню позицію, а ALBERT програє обом моделям.

Таблиця 4 – Технічні параметри моделей різних типів для набору ATIS

Базова модель	Час навчання, хв	Час передбачення, с	Фізичний розмір, МБ
BERT	1.835	6.120	418.112
ALBERT	2.384	7.013	655.370
DistilBERT	0.957	3.542	253.602

Порівняємо отримані результати з показниками метрик точності на наборі даних SNIPS (див. табл. 5). За ними видно, що на іншому наборі даних всі моделі ведуть себе так само, як і на попередньому. Для задачі маркування слотів для моделі ALBERT в попередніх дослідженнях відзначалася наявність проблеми зі сходимістю в деяких сценаріях [5], однак в даному випадку ця проблема відсутня і модель має кращу метрику F1, ніж обидві інші.

Таблиця 5 – Результати роботи моделей різних типів для набору SNIPS

Базова модель	Точність наміру	Метрика F1 для слотів	Точність для речення
BERT	0.984	0.962	0.919
ALBERT	0.984	0.965	0.922
DistilBERT	0.981	0.960	0.906

Загалом за точністю на рівні всього речення переважає, як і у випадку з набором даних ATIS, модель ALBERT, але різниця у значеннях метрики не є значною.

Аналогічні результати спостерігаються і для технічних показників роботи моделей (див. табл. 6), що дозволяє говорити про їх незалежність від набору даних, що застосовується.

Таблиця 6 – Технічні параметри моделей різних типів для набору SNIPS

Базова модель	Час навчання, хв	Час передбачення, с	Фізичний розмір, МБ
BERT	4.423	4.720	420.400
ALBERT	5.758	5.631	656.120
DistilBERT	2.571	2.504	253.535

Загалом можна зробити наступні висновки з даних досліджень:

– на обраних наборах даних моделі BERT та ALBERT показують майже однакові результати з точки зору метрик;
– всі моделі мають відмінні технічні характеристики, їх загальна тенденція співпадає на обох наборах даних. DistilBERT є найбільш оптимальною за цими показниками, доцільним є її використання у випадках обмежених обчислювальних ресурсів за умови прийнятної втрати точності;
– не спостерігається суттєвої залежності показників моделей від набору даних, тому є можливим вибір конкретної з них лише на основі запланованої області використання.

3.4 Налаштування гіперпараметрів моделі

Розроблена модель базується на моделі BERT, тому частина гіперпараметрів наслідується з неї, а саме:

– кількість шарів Transformer;

– розмір прихованого стану мережі;

– кількість напрямів для різноспрямованої уваги.

Однак при використанні попередньо навчених моделей значення цих параметрів є зафіксованими, а їх додаткове налаштування буде вимагати повного перенавчання моделі BERT, що повністю нівелює існуючі переваги у її використанні, тому ці значення залишимо без змін (кількість шарів – 12, розмір стану – 768, кількість напрямів уваги – 23 для базової моделі BERT) і сконцентруємося на тих, які є специфічними для безпосередньо розробленої архітектури.

Серед суттєвих з точки зору потенційного впливу на якість моделі параметрів можна виділити наступні:

– параметри оптимізатора Adam (темп навчання та епсілон);

– коефіцієнт впливу помилок у задачі маркувань слотів;

– коефіцієнт виключення (dropout rate).

Налаштування параметрів оптимізатора Adam для моделей, що використовують BERT, було ретельно досліджено у роботах Chen та інш. [5] та Castelucci та інш. [23] з дуже близькими результатами, отримані в даних дослідженнях оптимальні значення було використано протягом всіх попередніх експериментів. Специфічного для розробленої моделі налаштування не проводилося.

Коефіцієнт впливу помилок у задачі маркувань слотів відповідає відношенню гіперпараметрів β до α з формули 3.15. Заміна двох параметрів на один можлива, оскільки при розрахунку помилки під час навчання моделі ключову роль відіграє відношення помилок для двох підзадач, а не їх абсолютні значення. Натомість, це дозволяє спростити процес налаштування моделі до нового набору даних.

Перевіримо вплив даного параметру на метрики точності роботи моделі для набору даних ATIS (див. табл. 7). Для простоти їх аналізу побудуємо графік динаміки метрик моделі в залежності від значення вагового коефіцієнту, де за віссю у відкладено відношення значення метрики у конкретній точці до її максимального значення загалом (див. рис. 3.1).

Таблиця 7 – Вплив вагового коефіцієнту помилок слотів для набору ATIS

Коефіцієнт впливу	Точність наміру	Метрика F1 для слотів	Точність для речення
1.0	0.976483763	0.952565004	0.871220605
2.0	0.967525196	0.955727337	0.871220605
4.0	0.976483763	0.957285991	0.87793953
6.0	0.975363942	0.955071955	0.875699888
8.0	0.974244121	0.956063269	0.874580067

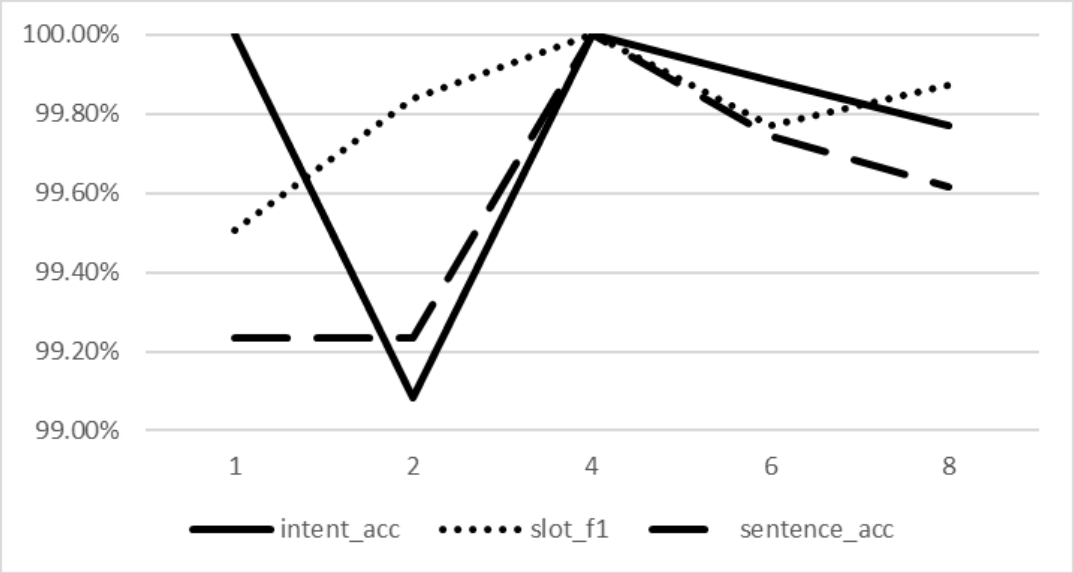

Рисунок 3.1 – Динаміка метрик якості моделі в залежності від значення вагового коефіцієнту для набору даних ATIS

З даного графіку, що зображує зміну значень метрик якості моделі відносно їх максимального значення, видно, що всі три метрики досягають свого максимального (або близького до нього) значення, коли коефіцієнт впливу помилок у маркуванні слотів дорівнює 4. Таке суттєве відхилення від базового значення вказує на те, що у наборі даних ATIS підзадачі не є рівноцінними з точки зору складності (класифікація наміру є більш точною, ніж класифікація маркувань слотів).

Тепер проведемо таке ж саме дослідження для набору даних SNIPS (див. табл. 8) та проаналізуємо його результати (див. рис. 3.2).

Таблиця 8 – Вплив вагового коефіцієнту помилок слотів для набору SNIPS

Коефіцієнт впливу	Точність наміру	Метрика F1 для слотів	Точність для речення
1	0.981428571	0.958310172	0.901428571
1.25	0.984285714	0.959688629	0.905714286
1.5	0.985714286	0.962426941	0.911428571
1.75	0.98	0.95935412	0.908571429
2	0.98	0.957488191	0.9

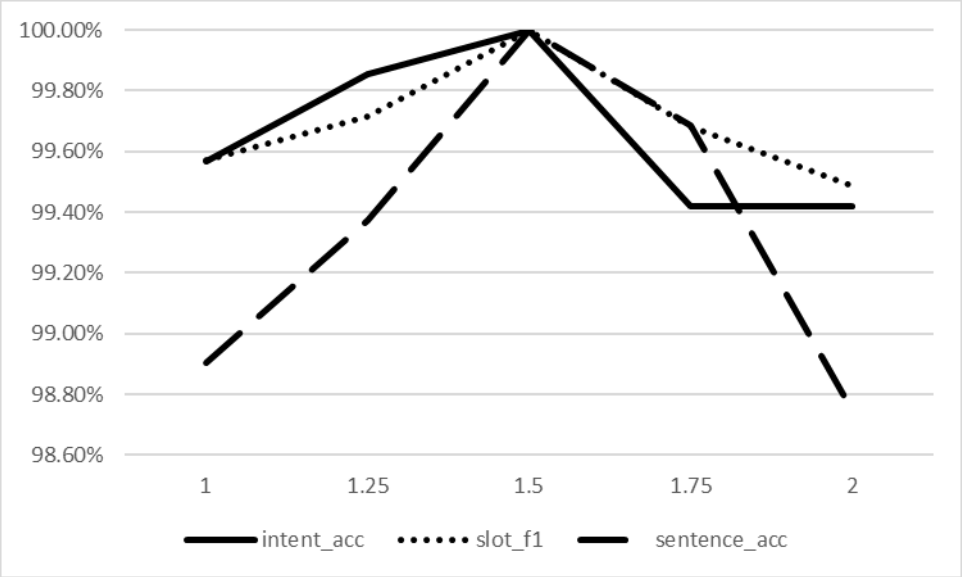

Рисунок 3.2 – Динаміка метрик якості моделі в залежності від значення вагового коефіцієнту для набору даних SNIPS

Результати для набору даних SNIPS є подібними до отриманих раніше для набору даних ATIS (наявне оптимальне значення гіперпараметру, при якому значення метрик якості досягають максимуму), однак в той же час можна бачити, що в цьому наборі даних підзадачі є більш збалансованими, а виграш від налаштування гіперпараметру є більшим.

В цілому можна зробити такі висновки щодо налаштування значення коефіцієнту впливу помилки при маркуванні слотів:

– даний гіперпараметр дозволяє додатково збалансувати між собою дві підзадачі в об’єднаній задачі заповнення форм;

– вплив параметру та його оптимальне значення сильно залежить від характеру набору даних, на якому відбувається навчання, і тому має підбиратися експериментально для кожного зі сценаріїв використання.

Нарешті, проведемо подібне дослідження впливу гіперпараметру на метрики моделі для коефіцієнту виключення (див. табл. 9).

Таблиця 9 – Вплив коефіцієнту виключення на метрики точності моделі

Коефіцієнт виключення	Точність наміру	Метрика F1 для слотів	Точність для речення
0.0	0.9731243	0.953954306	0.873460246
0.1	0.972004479	0.9543379	0.873460246
0.2	0.974244121	0.953602812	0.868980963
0.3	0.975363942	0.95119382	0.86450168
0.4	0.9731243	0.951026856	0.865621501
0.5	0.9731243	0.953300562	0.868980963

Як бачимо, у цілому суттєвий вплив даного коефіцієнту на значення метрик відсутній, як відсутній і тренд у отриманих значеннях. Такий результат пов’язаний з характером процесу навчання [5], в якому результуюча модель обирається як найкраща у середньому модель за епохами навчання окремо. Оскільки така поведінка у процесі навчання тісно пов’язано з характером даних, що використовується, має сенс проводити за необхідності налаштування даного параметру. За замовчуванням при цьому можна використовувати значення 0.1, що співпадає і з конфігурацією моделі BERT та похідних від неї.

ВИСНОВКИ

В результаті даної роботи розроблено алгоритм вирішення задачі заповнення форм на основі тексту природньою мовою. Цей алгоритм базується на нейромережевій моделі природньої мови BERT і дозволяє передбачувати намір користувача та параметри (слова або словосполучення), що відповідають цьому наміру.

Перед початком розробки моделі було проведено аналіз предметної області, в ході якого було виявлено існуючі підходи до вирішення задачі заповнення форм, а також близьких до неї задач та визначено можливі напрями їх удосконалення. Це дозволило створити модель для вирішення даної задачі, включаючи такі питання як навчання даної моделі та оцінка якості її роботи. Після цього було побудовано нейронну мережу, що відповідає розробленій моделі, з використанням мови Python та набору бібліотек PyTorch та проведено практичні експерименти з даною мережею.

Для підтвердження практичної цінності роботи було також розроблено прикладне застосування – чат-бот, що використовує розроблену нейромережеву модель для заповнення заданих форм на основі повідомлень користувачів вільним текстом.

Подальший розвиток даної роботи може здійснюватися у декількох напрямках. По-перше, є доцільним дослідити її пристосованість до роботи з різними мовами та визначити можливі шляхи, якими можна адаптувати модель під конкретно задану мову. По-друге, з подальшим поширенням архітектури BERT з’являється все більше різних її варіантів, які оптимізують певні її властивості, наприклад, фізичний розмір готової моделі та швидкість її роботи, і включення елементів цих архітектур може дозволити покращити результати роботи моделі у цілому для даної задачі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Yerokhin A., Turuta O., Babii A., Nechyporenko A. Intelligent information system of heterogeneous medical data analysis // 2017 12th International Scientific and Technical Conference on Computer Sciences and Information Technologies (CSIT). 2017. pp. 332–335.

2. Jurafsky D., Martin J.H. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. – Prentice Hall, 2009. – 988 p.

3. Dauphin Y.N. et al. Zero-Shot Learning for Semantic Utterance Classification // International Conference on Learning Representations (ICLR). 2014.

4. Surdeanu M. Overview of the TAC2013 Knowledge Base Population Evaluation: English Slot Filling and Temporal Slot Filling // Theory and Applications of Categories. 2013.

5. Chen Q., Zhuo Z., Wang W. Bert for joint intent classification and slot filling // arXiv preprint arXiv:1902.10909. 2019.

6. Eisenstein J. Introduction to Natural Language Processing. Cambridge: The MIT Press, 2019. – 535 p.

7. Haffner P., Tur G., Wright J.H. Optimizing Svms For Complex Call Classification // 2003 IEEE International Conference on Acoustics, Speech, and Signal Processing. 2003.

8. Raymond C., Riccardi G. Generative and discriminative algorithms for spoken language understanding // 8th Annual Conference of the International Speech Communication Association. 2007.

9. Vaswani A. et al. Attention Is All You Need // Advances in Neural Information Processing Systems. 2017. pp. 6000–6010.

10. Yerokhin A. et al. A new intelligence-based approach for rhinomanometric data processing // 2016 IEEE 36th International Conference on Electronics and Nanotechnology (ELNANO). Kiev. 2016. pp. 198-201.

11. Yao K. et al. Spoken language understanding using long short-term memory neural networks // 2014 IEEE Spoken Language Technology Workshop (SLT). 2014. pp. 189–194.

12. Cho K. et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation filling // arXiv preprint arXiv:1406.1078. 2014.

13. Chung J. et al. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling // NIPS Deep Learning and Representation Learning Workshop. 2014.

14. Meng L., Huang M. Dialogue Intent Classification with Long Short-Term Memory Networks // Natural Language Processing and Chinese Computing. NLPCC 2017. Lecture Notes in Computer Science, vol. 10619. 2017. pp. 42–50.

15. Gers F., Schmidhuber E. LSTM recurrent networks learn simple context-free and context-sensitive languages // IEEE Transactions on Neural Networks, vol. 12, no. 6. 2001. pp. 1333–1340.

16. Sutskever I., Vinyals O., Le Q.V. Sequence to sequence learning with neural networks // Advances in Neural Information Processing Systems. 2014. pp. 3104–3112.

17. Kim Y. et al. Structured Attention Networks // arXiv preprint arXiv: 1702.0088. 2017.

18. Wang D., Zheng T.F. Transfer learning for speech and language processing // 2015 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA), Hong Kong. 2015, pp. 1225–1237.

19. Devlin J. et al. Bert: Pre-training of deep bidirectional transformers for language understanding // arXiv preprint arXiv:1810.04805. 2019.

20. Dai A.M., Le Q.V. Semi-supervised sequence learning // Advances in neural information processing systems. 2015. pp. 3079–3087.

21. Liu P., Qiu X., Huang X. Adversarial Multi-task Learning for Text Classification // 55th Annual Meeting of the Association for Computational Linguistics (ACL), vol. 1. 2017. pp. 1–10.

22. Peng B. et al. Recurrent Neural Networks with External Memory for Spoken Language Understanding // Natural Language Processing and Chinese Computing.

NLPCC 2015. Lecture Notes in Computer Science, vol 9362. 2015. pp. 25–35

23. Castellucci G. et al. Multi-lingual Intent Detection and Slot Filling in a Joint BERT-based Model // arXiv preprint arXiv: 1907.02884. 2019.

24. Haihong E A Novel Bi-directional Interrelated Model for Joint Intent Detection and Slot Filling // 57th Annual Meeting of the Association for Computational Linguistics (ACL). 2019.

25. Gupta A. et al. CASA-NLU: Context-Aware Self-Attentive Natural Language Understanding for Task-Oriented Chatbots // Proceedings of EMNLP-IJCNLP. 2019.

26. Kurata G. et al. Leveraging Sentence-level Information with Encoder LSTM for Semantic Slot Filling // Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing. 2016. pp. 2077–2083.

27. Liu B., Lane I. Attention-Based Recurrent Neural Network Models for Joint Intent Detection and Slot Filling // Interspeech. 2016. pp. 685–689.

28. Goo C.-W. et al. Slot-Gated Modeling for Joint Slot Filling and Intent Prediction // Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers). 2018. pp. 753–757.

29. Bishop C. Pattern Recognition and Machine Learning. – New York: Springer-Verlag, 2006. – 738 p.

30. Zhou J., Xu W. End-to-end learning of semantic role labeling using recurrent neural networks // ACL 2015, vol. 1. 2015. pp. 1127–1137.

31. Murphy K. Machine Learning: A Probabilistic Perspective. Cambridge: The MIT Press, 2012. – 1098 p.

32. Mikolov T. et al. Distributed representations of words and phrases and their compositionality // Advances in Neural Information Processing Systems. 2013. pp. 3111–3119.

33. Wu Y. et al. Google's neural machine translation system: Bridging the gap between human and machine translation // arXiv preprint arXiv:1609.08144. 2016.

34. Hakkani-Tür D. et al. Multi-domain joint semantic frame parsing using bi-

directional RNN-LSTM // Interspeech. 2016. pp. 715–719.

35. Gehring J. et al. Convolutional Sequence to Sequence Learning // Proceedings of the 34th International Conference on Machine Learning, vol. 70. 2017.

36. Lan Z. et al. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations // arXiv preprint arXiv:1909.11942. 2019.

37. Hinton G., Vinyals O., Dean J. Distilling the Knowledge in a Neural Network // NIPS Deep Learning and Representation Learning Workshop. 2015.

38. Sanh V. et al. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter // The 5th Workshop on Energy Efficient Machine Learning and Cognitive Computing, NeurIPS. 2019.

39. Hemphill C.T., Godfrey J.J., Doddington G.R. The ATIS Spoken Language Systems Pilot Corpus // Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27. 1990.

40. Coucke A. et al. Snips Voice Platform: an embedded Spoken Language Understanding system for private-by-design voice interfaces // arXiv preprint arXiv:1805.10190. 2018.

41. Kingma D.P., Ba J. Adam: A Method for Stochastic Optimization // International Conference on Learning Representations (ICLR). 2014.

42. Srivastava N. et al. Dropout: A Simple Way to Prevent Neural Networks from Overfitting // Journal of Machine Learning Research, no. 15 (56). 2014. pp. 1929–1958.

43. Müller R., Kornblith S., Hinton G. When Does Label Smoothing Help? // Proceedings of NIPS. 2019.

44. Goutte C., Gaussier E. A Probabilistic Interpretation of Precision, Recall and F-Score, with Implication for Evaluation // Advances in Information Retrieval. Berlin: Springer, 2005. pp. 345–359.

45. Wolf T. et al. HuggingFace's Transformers: State-of-the-art Natural Language Processing // arXiv preprint arXiv:1910.03771. 2019.