НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

«До захисту допущено»
В. о. завідувача кафедри
_____ О.Л. Тимощук
«___»_____20__ р.

Дипломна робота

на здобуття ступеня бакалавра
з напряму підготовки 6.050101 «Комп'ютерні науки»
на тему: «Чатбот на основі навчання з підкріпленням»

Виконав:
студент IV курсу, групи КА-55
Котирло Віталій
Володимирович

Керівник:
доцент, д.т.н. Недашківська
Н.І.

Консультант з економічного розділу:
доцент, к.е.н. Шевчук О.А. _____

Консультант з нормоконтролю:
доцент, к.т.н. Коваленко А. Є. _____

Рецензент:
проф., д. т. н. Теленик С.Ф. _____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.
Студент _____

Київ – 2019 року

РЕФЕРАТ

Дипломна робота: 136 с., 15 рис., 21табл., 6 додатки, 19 джерел.

ДІАЛОГОВІ СИСТЕМИ, ЧАТ-БОТ, НАВЧАННЯ З ПІДКРІПЛЕННЯМ, ГЛИБОКЕ НАВЧАННЯ, СИМУЛЯЦІЯ КОРИСТУВАЧА, ОБРОБКА ПРИРОДНЬОЇ МОВИ, ІНФОРМАЦІЙНИЙ ПОШУК, БАЗА ЗНАНЬ, ГЛИБИННІ Q-МЕРЕЖІ.

Об’єктом дослідження є задача симулювання дискусії чи розмови з людиною відповідно до заданого практичного завдання. Предметом дослідження є методи і алгоритми навчання з підкріпленням для побудови діалогової системи у вигляді чат-боту.

Метою дипломної роботи є розробка та навчання програмного продукту для реалізації діалогу природньою мовою між діалоговою системою та симулятором користувача з використанням методів глибокого навчання з підкріпленням.

У роботі розроблено і навчено програмний продукт у вигляді чат-бота на основі глибокого навчання з підкріпленням, який демонструє роботу системи замовлення квитків в кінотеатрах. Побудовано симулятор користувача, який відіграв роль середовища для агента, який здатний завершувати 89% відсотків діалогів успіхів. Для реалізації інтерфейсу створено бот в месенджері Telegram. Чат-бот розгорнуто за допомогою сервісу Heroku для забезпечення безперервного доступу та можливості використання програми з будь-якого гаджету.

Програма написана мовою Python з використанням PyCharm та Google Colaboratory.

При написанні роботи використано наукові статті в галузі машинного навчання, глибокого навчання та глибокого навчання з підкріпленням.

ABSTRACT

The work consist of 136 pages 15 images 21 tables 19 sources

The theme: «A deep reinforcement learning chat-bot».

DIALOGUE SYSTEMS, CHATBOT, REINFORCEMENT LEARNING, DEEP LEARNING, USER SIMULATION, NATURAL LANGUAGE PROCESSING, INFORMATION SEARCH, KNOWLEDGE BASE, DEEP Q-NETWORK.

The object of research is the task of simulating a discussion or conversation with a person in accordance with a given practical task. The subject of the study is methods and algorithms of reinforcements learning for building a dialogue system in the form of chat-bot.

The purpose of the thesis is to develop and teach the software product to implement a dialogue in the natural language between the dialogue system and the user's simulator using the methods of deep reinforcement learning.

In the work, the software product in the form of a chat-bot was developed and trained on the basis of deep reinforcement learning, which demonstrates the work of the system of booking tickets in cinemas. A user simulator has been constructed, which has played the role of environment for agent that can complete 89% of the success dialogs. To implement the interface, a bot has been created in Telegram Messenger. The Chat-box is hosted with the help of Heroku to provide continuous access and the ability to use the program from any gadget.

The program is written in Python using PyCharm and Google Colaboratory.

Scientific articles in the field of machine learning, deep learning and deep reinforcement learning were used during writing the work.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

DST(ТСД) – трекер стану діалогу

IOB - формат Inside-Outside-Beginning.

IR(ІП) –інформаційний пошук.

FAQ(ЧаПи) – часті питання.

POL – навчання стратегій.

ML(МН) – машинне навчання.

RL – навчання з підкріпленням.

NLP – обробка природньої мови

NLU – розуміння природньої мови.

NLG – генерування природньої мови.

ВСТУП

Створення інтелектуальної системи, яка не лише імітує людську розмову, але й доцільно відповідає на питання та спілкується на різноманітні теми : від останніх новин кіно до теорії відносності, системи, яка до того ж може виконувати прикладні завдання, такі як бронювання квитків, завжди було однією з провідних цілей в розвитку штучного інтелекту. До недавнього часу ця мета залишалася недосяжною. Однак зараз ми спостерігаємо багатообіцяючі результати.

Результати як в академічному науковому співтоваристві, так і у виробництві. Це пов’язано з тим, що великі обсяги даних стали доступними для використання, а також з видатними досягнення в області глибокого навчання та навчання з підкріпленням.

Розмовний штучний інтелект є фундаментальним для природних інтерфейсів користувача. Це швидко зростаюча галузь, що приваблює багато дослідників у спільнотах з обробки природних мов (NLP), інформаційного пошуку (IR) та машинного навчання (ML). В останні роки зросла індустрія навчальних посібників і оглядових документів в даних областях.

РОЗДІЛ 1
ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність задачі

Остання тенденція, яка привертає увагу більшості технологій - це чат-боти. Поєднання миттєвої реакції на запит та зручна форма відповіді робить їх привабливою заміною чи доповненням до тенденції використання веб-додатків.

В загальних рисах, бот - це не що інше, як програмне забезпечення, яке виконуватиме автоматичні завдання. У даній роботі було зосереджено увагу на класі ботів, які взаємодіють з користувачем за допомогою чату, тобто в чат-ботах. Найбільш природним визначенням є таке: «Чат-бот - розроблена програма, яка може підтримувати дискусію чи розмову з людиною». Наприклад, будь-який користувач може відправити бота запит або твердження і бот відповість або зробить відповідну дію у зручній для людини формі.

Чат-бот повинен взаємодіяти у формі миттєвого обміну повідомленнями. За допомогою штучного повторення шаблонів людської розмови методами машинного навчання комп'ютери можуть навчатися самостійно, без програмування зокрема методами природньої обробки мов.

Чат-бот запрограмований на робота незалежно від людського оператора. Він може відповідати на запитання, які сформульовані до нього на природній мові та відповідати як реальна людина. Він може надавати відповіді на основі комбінації попередньо визначених сценаріїв і додатків машинного навчання.

Коли задають питання, чат-бот буде реагувати на основі бази знань, доступної йому на той момент. Якщо розмова вводить поняття, яке не запрограмовано для розуміння, він або відхиляє розмову, або за можливості передає комунікацію людині. У будь-якому випадку, він може вчиться на кожній взаємодії. Таким чином, чат-бот буде поступово адаптуватись і набувати актуальності.

В даний час багато компаній шукають різні способи використання чат-ботів для покращення взаємодії з клієнтами. Бронювання квитків, технічна підтримка, допомога при виборі товарів, замовлення послуг, консультація клієнтів – далеко не повний список можливих прикладних застосувань чат-ботів.

Сьогодні чат-боти отримують особливу популярність у зв'язку з широким використанням месенджерів, тому такий вид взаємодії став інтуїтивно зрозумілим та зручним для користувачів.

Компанії можуть економити кошти та ресурси розмітивши в соціальних мережах чат-бот, який замінить функціонал повноцінного веб чи мобільного додатку. В цьому випадку користувач зможе використовувати послуги компанії, не встановлюючи сторонніх додатків та не відволікаючись від звичних справ.

Через ці переваги, боти розглядаються як область з величезним потенціалом для роботи з клієнтами та стимулювання продажів. Тим не менш, більшість чат-ботів сьогодні зазвичай будуються на основі систем, які дають користувачеві меню і користувач здійснює переходи виключно через це жорстке меню. До того ж, більшість чат-ботів є закритими доменними[1] ботами(тобто цілеспрямованими, з обмеженою областю навчання та використання), що означає, що вони зосереджені на одній конкретній задачі і навчаються тільки для конкретного використання. Alexa, Siri є прикладами відкритих доменних ботів(загального призначення) [11]. Проте боти для бронювання квитків в кінотеатрі є закритими доменними ботами. Боти із закритими доменами можуть бути як на основі штучного інтелекту, так і на основі правил. Хоча користувачі очікують, що боти загального призначення розумні у всіх аспектах, мета бот-системи полягає не тільки в цьому. Метою бот-системи залишається автоматизація служби за допомогою розмовного інтерфейсу, що дозволяє користувачеві отримати доступ до сервісу з

[1] Домен – предметна область, може характеризуватись певним поняттям.

платформ, які вони часто відвідують. Це означає, що не треба відмовлятися від побудови бота на основі правил, який дійсно якісно працює через те, що бот не розумний, як людина. При цьому, необхідно продовжувати будувати інтелектуальні системи, оскільки чим розумніші вони з точки зору розуміння кінцевого користувача, тим більше вони будуть корисні.

1.2 Аналіз існуючих класифікації чат-ботів

На основі знань, до яких має доступ бот, чат-боти можна розділити на два види: відкритий і закритий [6]. Також чат-боти можна класифікувати за допомогою інших параметрів, а саме:

a) Область знань;
b) Послуга, що надається;
c) Мета;
d) Метод генерації відповіді.

1.2.1. Класифікація за областю знань.

Тут чат-боти класифікуються на підставі знань, які вони отримують, або кількості даних, на яких вони навчаються. На рисунку 1.1 наведена схема класифікації чат-ботів. Класи такі:

1) З відкритим доменом: такі боти можуть говорити на загальні теми і відповідати належним чином;

2) Із закритим доменом: такі боти зосереджені на певній області знань і можуть не відповідати на інші питання.

Наприклад, бот для бронювання місць в ресторані не скаже вам ім'я першого чорного президента Америки. Він може розповісти вам жарт, сказати, що сьогодні прекрасна погода та відповісти ще на декілька легких запитань, але це не є очікуваним від нього, тому що його робота полягає в тому, щоб забронювати стіл і дати вам інформацію про ресторан.

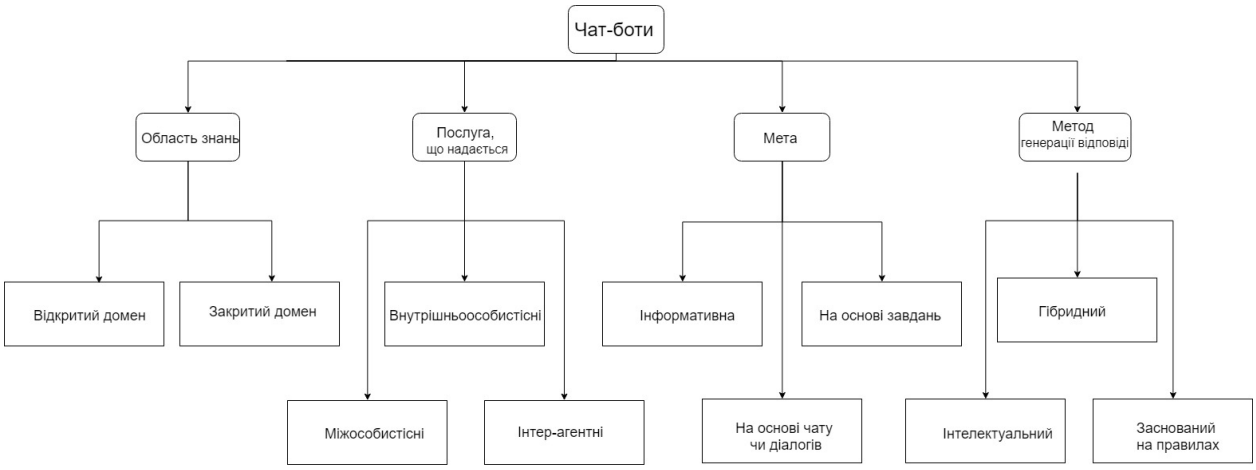


Рисунок 1.1 - Класифікація чат-ботів

1.1.2. Класифікація за послугою, що надається.

Тут, боти класифікуються на основі проксеміки, подібно до методів класифікації, але замість метричної близькості, вона заснована на сентиментальній близькості бота до користувача, кількості особистої взаємодії, яка відбувається і також залежить від завдання, яке виконує бот.

1) Міжособистісні: до цієї категорії підпадають чат-боти, які знаходяться в області комунікації, що належить до діапазону соціальних або особистих відстаней на діаграмі проксеміки. Боти, які надають такі послуги, як : бронювання ресторанів, бронювання польотів, боти FAQ і т.д. Ці чат-боти не повинні бути супутниками користувача, вони повинні отримувати інформацію і передавати її користувачеві. Вони можуть мати особистість, бути

дружніми і, можливо, запам'ятовувати інформацію про користувача, але зазвичай цього від них не очікують.

2) Внутрішньоособистісні: ці чат-боти існуватимуть у межах особистої території користувача, наприклад, додатків месенжерів, таких як : facebook messenger, slack, whatsapp і виконують завдання, які лежать в особистому домені користувача. Управління календарем, зберігання думок користувача тощо. Вони будуть супутниками для користувача і розуміють його, як людина. В теперішні дні вони не достатньо розвинені та поширені, але з розвитком методів обробки природньої мови, вони стануть більш розповсюдженні.

3) Інтер-агентні: подібні боти будуть поширені в областях пов’язаних з інтернетом речей. В цьому випадку дві системи спілкуються один з одним для виконання завдання. Оскільки боти стають усюдисущими, вони будуть потребувати можливості взаємодії один з одним. Незабаром виникне потреба в протоколах для комунікації між собою. Це може бути служба, яка керує іншими ботами або комунікаціями, що полегшує розробникам і користувачам інтеграцію різних послуг в розмовній екосистемі. Інтеграція Alexa-Cortana є прикладом взаємодії між агентами.

1.1.3. Класифікація за метою

Тут боти класифікуються на основі основної мети, яку вони прагнуть досягти.

1) Інформативна : зазвичай це алгоритм пошуку інформації, заснований на алгоритмі, який дозволяє або отримати результат запиту з бази даних, або виконати зіставлення рядків. У більшості випадків вони будуть посилатися на статичне джерело інформації, таке як : сторінка FAQ сайту або база даних з записами.

Приклад: FAQ боти.

Завдання, пов'язані з цим: співставлення рядків, виділення іменованих сутностей, генерація відповіді.

2) На основі чату чи діалогів: ці боти говорять з користувачем, як інша людина. Їхня мета - правильно відповісти на речення, яке вони отримали. Тому вони часто будуються з метою продовження розмови з користувачем на основі таких методів, як : зворотні питання та ухилення від відповідей. Приклад: Siri, Alexa, mitsuku, Jenny, Tay, Xaoice [12].

Задачі та алгоритми: вилучення імені об'єкта, пошук інформації, узгодження рядків, виявлення релевантності.

3) На основі завдань: вони виконують одне певне завдання, наприклад, бронюють рейс або допомагають вибрати послугу. У більшості випадків дії, які необхідні для виконання завдання уже попередньо відомі, визначений також потік подій, включаючи винятки. Боти розумні в контексті запиту інформації і розуміння користувачем даних, що вводяться.

Приклад: боти для бронювання квитків.

1.1.4. Класифікація за методом генерування вхідних даних і відповідей

Ця класифікація враховує метод обробки вихідних даних і отримання відповідей.

1) Суто інтелектуальні системи генерують відповіді та використовують специфічні методи для обробки природньої мови, щоб зрозуміти запит користувача. Ці системи використовуються, коли область(домен) вузький і доступних даних достатньо для підготовки системи.

Це системи, які мають намір підтримувати ієрархічну структуру базових значень мови, можуть використовуватися, коли доступна велика кількість даних і система може бути навчена на цих даних. Вони часто використовують алгоритми NLP і NLU для обробки вхідних даних і генерування пропозицій.

Штучні нейронні мережі, LSTM, SVM, моделі послідовностей, генеративні алгоритми. Ще не створено суто генеративних моделей, навіть найсучасніші системи, такі як : Alexa, Siri і Cortana, базуються на основі напівправил. На рисунку 1.5 наведено приклад генеративної моделі.

Рисунок 1.5 - Генеративна модель

2) Системи, які засновані на правилах або пошуку, використовують шаблони зіставлення і є досить жорсткими. Вони можуть бути використані, коли кількість можливих результатів фіксовано, а сценарії можуть бути прописані вручну. Моделі на основі пошуку можуть використовуватися, коли дані обмежені, а домен розмов обмежений декількома сценаріями розмови. Отже, якщо всі можливі відповіді можна уявити, то модель на основі пошуку працює досить якісно. Крім того, коли бот не повинен відображати інтелект у всіх сценаріях, але може дозволити собі заперечувати знання відповіді, такі засновані на пошуку моделі працюють досить якісно. Системи бронювання, системи FAQ або будь-які інші системи, які витягують інформацію, можуть добре працювати навіть з пошуковим ботом. На рисунку 1.4 наведено приклад моделі на основі пошуку.

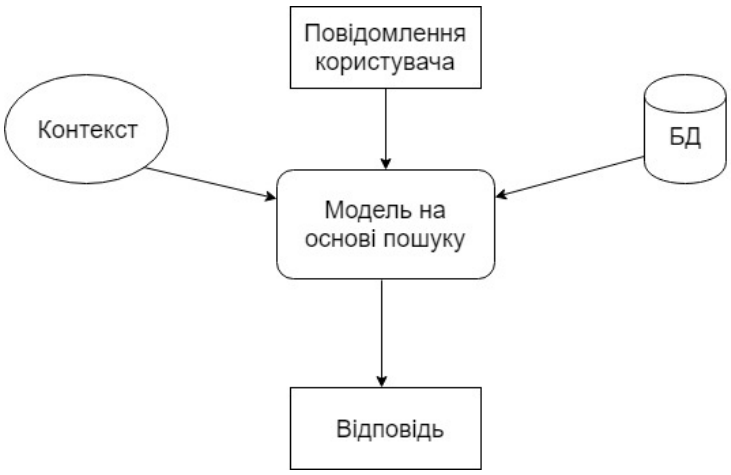


Рисунок 1.4 - Модель на основі пошуку

3) Гібридні системи використовують суміш правил і машинного навчання. Прикладом може бути система, яка використовує блок-схему для керування напрямком розмови, але надає відповіді, які генеруються з використанням обробки природних мов. На рисунку 1.2 представлено приклад схеми для гібридного чат-бота.

Боти не повинні належати виключно до однієї категорії. Ці категорії існують у кожному боті в різних пропорціях. Наприклад, для всіх ботів будуть потрібні якісь можливості чату, можливо, бота для магазину знадобиться витягання інформації, коли справа доходить до FAQ, і пошук на сайті, коли справа доходить до надання результатів продукту. Послуга, яку надає бот, може включати в себе всі доступні види алгоритмів. Наприклад, бот буде із закритим доменом, але буде запрограмований на малі світські розмови.

1.3 Аналіз існуючих рішень та підходів та загальна архітектура чат-бота

Як видно вище, можливі декілька категорій чат-ботів. Чат-бот буде комбінацією однієї або декількох категорій. Наприклад, він може бути інформативним і заснованим на правилах, або може бути на основі завдань,

але використовує методи для чат-ботів загального призначення. Проте всі чат-боти наслідують загальну архітектуру, загальні стадії розробки. Для кожного рівня загальної стадії розробки, для кожної категорії, будуть використані алгоритми або методи, що відносяться до цієї категорії. Навіть у різноманітних чат-ботах, які отримали приз Лобнера, можна спостерігати загальну архітектуру для потоку інформації [13].

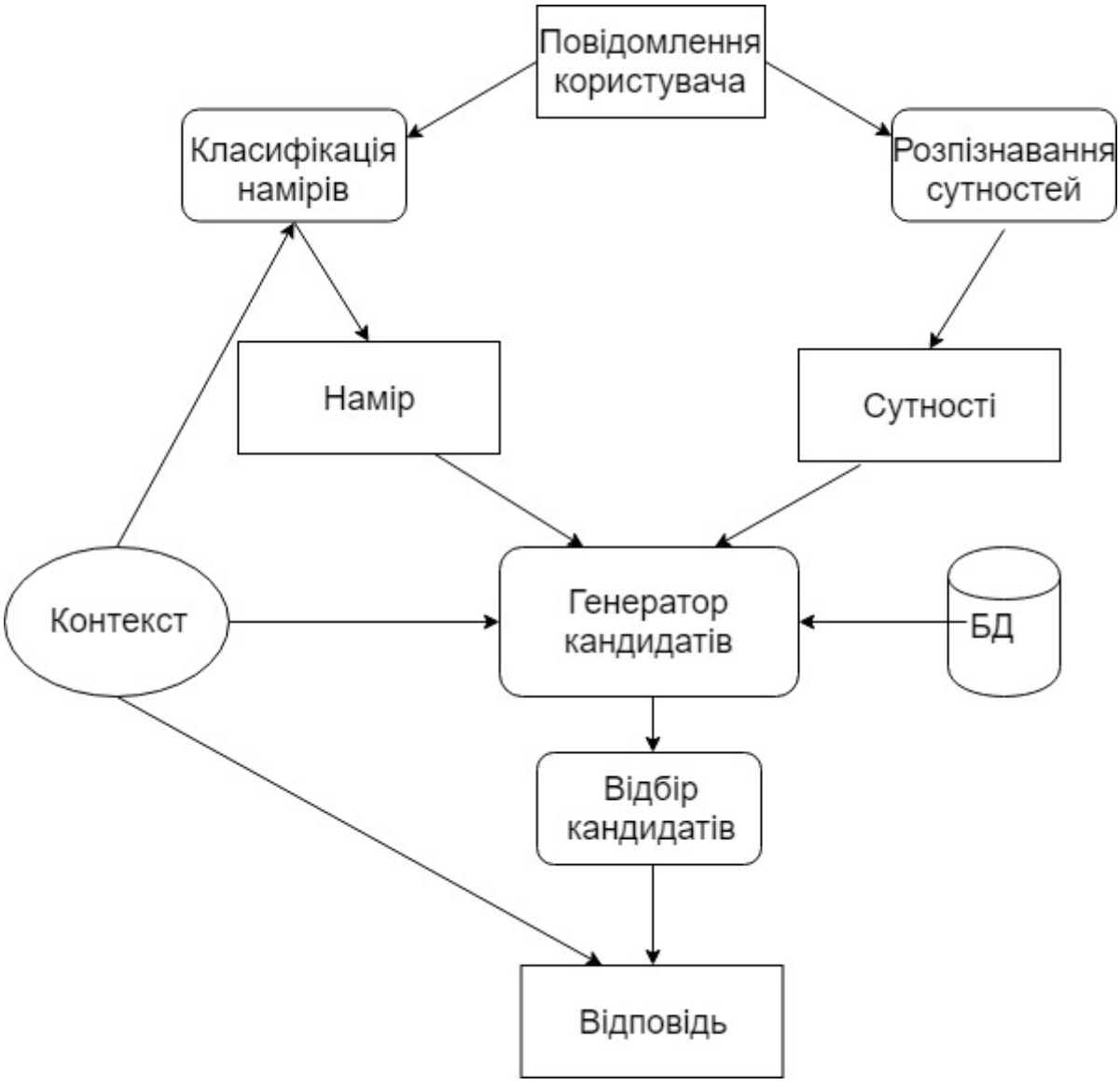


Рисунок 1.2 - Приклад гібридного чат-бота

Чат-бот складається з чотирьох послідовних етапів або модулів, які показані на рисунку 1.3. Вхід приймається і обробляється у необхідному форматі. Далі витягаються іменовані сутності і виявляється намір. Вони

використовуються для генерування можливих відповідей з яких найбільш релевантна відповідь подається користувачеві.

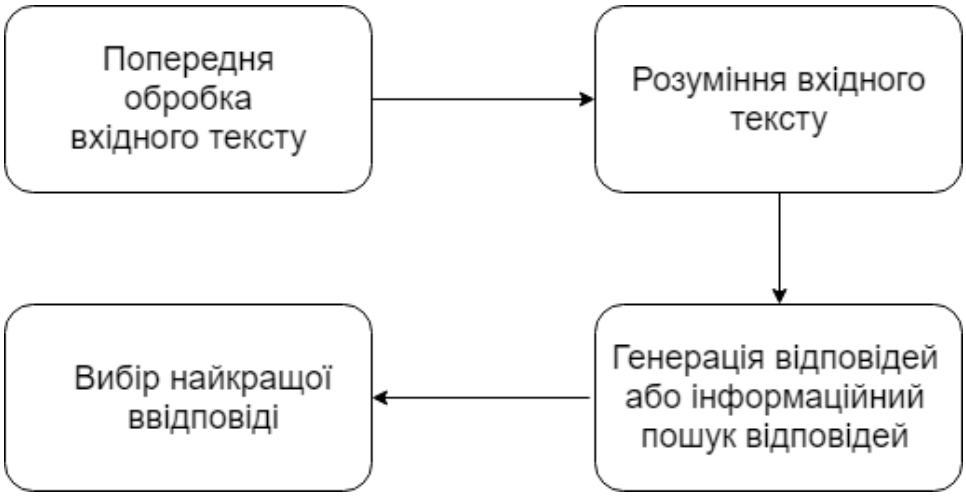


Рисунок 1.3 - загальна схема чат-бота

1.3.1 Завдання, що стоять перед моделями

Різні завдання, що входять у різні етапи, такі: розбиття речення, визначення границь речення, розбір речення, тегування - це деякі з основних алгоритмів обробки вводу, які часто використовуються в системах подібного типу. Якщо користуватися бібліотеками, ці завдання маскуються в функції. Для розуміння вхідних даних мають місце наступні алгоритми : витягнення іменних сутностей, виявлення намірів, виявлення неоднозначності, виявлення контексту, аналіз настрою, класифікація запитів, концепцій та виявлення синонімів [6].

Після того, як запит або вхідні дані структуровані у бажану форму буде генеруватись відповідь. Вона може генеруватися через заздалегідь визначений формат або через машинне навчання. Методи узагальнення та спрощення також можна використовувати, якщо відповідь вибирається з джерела тексту.

Після генерації набору відповідей-кандидатів, обирається найкраща та повертається користувачу.

1.3.2 Готові рішення для вилучення імен, об'єктів та ключових сутностей

Однак не все з вищезазначеного потрібно робити з нуля. Доступні різні інструменти і платформи, які забезпечують готові та напівготові шаблони, модулі та алгоритми. Порівняння цих рішень можна побачити в таблиці 1.1.

1) Recast.ai дозволяє користувачеві створювати власні запити і тренувати бота на них. Крім надання послуг машинного навчання для вилучення об'єктів і намірів, він надає багато чого: від шаблонів до аналітики для розгорнутих ботів. Він забезпечує хостинг та інтеграцію ботів на різних платформах. Можна також додати зовнішні інтеграції і створити власну логіку в роботі. Він має вбудовану підтримку аналізу настроїв і декількох мов. Крім того, для додавання власних запитів, він також має інтерфейс перетягування для створення потоку розмов.

2) Wit.ai допомагає класифікувати намір і витягувати іменовані та власні сутності з даних. Потім він повертає мітки до програми або бота. Це дозволяє витягувати таку інформацію, як місце розташування, час, дату, погоду, а також дозволяє створювати власні наміри. Він перетворює введену користувачем інформацію у структуровані дані, а також допомагає в розмові шляхом надсилання тексту. Однак для виконання дій на структурованих даних вихід wit.ai повинен бути надісланий до додатку або серверу, який надасть відповідь назад до wit.ai, який в свою чергу потім повертає її до платформи розмови.

3) API.ai допомагає витягувати наміри і об'єкти з запиту. Також реалізована можливість навчання бота з користувацькими намірами і сутностями Як і Wit.ai, він надає лише послугу обробки тексту, а не бот-хостинг.

1.3.3 Готові рішення для генерування тексту та чат-ботів на основі правил

1) Chatfuel надає інтерфейс перетягування для створення бота, який побудований на правилах. Його модуль штучного інтелекту дозволяє навчати бота для відображення вхідних даних на вихід, але в порівнянні з recast.ai та іншими модулями, він досить жорсткий з точки зору потоків розмов. Він також дозволяє отримувати нагадування та інтегруватися з послугами, такими як : електронна пошта та IFTTT. Він також дозволяє інтегрувати json формат для розміщення користувальницької логіки в роботі. Найбільш привабливою точкою сервісу є те, що надзвичайно просто побудувати бота на основі правил, який підходить для малого бізнесу. Він також дозволяє користувачеві обирати списки для розсилки. Він потребує лише інтеграції зі сторінкою faceboook і не потребує інших установок.

2) Pandora bot надають платформу для створення ботів. Він також робить користувальницькі боти на платній основі. Він використовує AIML і надає IDE для створення ботів, а також API AIaaS.

3) Платформи для обробки тексту:

Крім того, різні хмарні платформи від таких компаній, як : Amazon, IBM, Google, Microsoft випустили власні розмовні API, які дозволяють розробникам аналізувати дані, перетворювати мову в текст і навпаки, і навіть створювати відповідні відповіді. Перевага використання цих платформ полягає в тому, що їх легше інтегрувати з вже існуючою хмарою.

Ці моделі побудовані на величезних обсягах даних і використовують потужні можливості глибокого навчання, які надають цим платформам точність, яку розробник не може досягти за допомогою простих ресурсів. Приклад: Microsoft bot framework, LUIS, Amazon Lex та Poly, IBM Watson, Google Speech API, Google Natural Language API.

4) Для навчання за власними даними:

Можна також побудувати свої прототипи з нуля і перевірити їх на наборі даних. Їх можна тренувати, використовуючи такі платформи, як Tensorflow та Keras. Потім модель може бути масштабована з використанням власного сервера або платформи для машинного навчання, таких як : Amazon, Google та інші згадані вище.

Таблиця 1.1 – Порівняння готових розв’язків

Номер	Платформа	Класифікація намірів	Вилучення суб'єктів	Аналіз настрою	Вилучення ключових слів	Базованість на правилах	Мова
1	Recast. AI	+	+	+	-	+	3 мови: англійська, французька та іспанська мають всі функціональні можливості, 17 інших мають обмежені можливості вилучення об'єктів і класифікацію намірів. Один агент може підтримувати кілька мов одночасно.
2	Wit.ai	+	+	+	-	+	50 мов доступні або 39 з яких знаходяться в бета-версії
3	Api.ai	+	+	-	-	-	Підтримуються 15 мов. З кожним запитом, мовний тег повинен бути відправлений. Один агент підтримує одну умову, без можливості зміни
4	Chatfuel	+-	-	-	-	+	Підтримується більше 40 мов
5	Pandora Bots	+	+	-	+	+	Багатомовна
6	Microsoft Bots, LUIS	+	+	+	+	-	Більше 30 мов доступні

Продовження таблиці 1.1

7	Google Natural language API	-	+	+	+	-	Підтримка 9 мов, виявлення настроїв суб'єктів і класифікація тексту доступні тільки для англійської мови
8	IBM Watson	-	+	+	+	-	Різні послуги доступні для багатьох мов в тому числі виявлення ключових слів та об'єктів. Англійська та іспанська мають найбільший пакет послуг

1.4 Проблематика предметної області

Дивлячись на поточний вибух популярності чат-ботів, можна подумати, що чат-боти досягли успіху і у нас є агенти, які спілкуються як люди. Проте використання чат-ботів загального призначення виявляється розчаровуючим, оскільки вони не роблять того, що очікують користувачі. Однак, з іншого боку, цільонаправлені чат-боти працюють достатньо якісно.

Ми очікуємо, що агенти зможуть відповісти на все, не розуміючи, що завдання агента – дати відповідь на відповідний конкретному чат-боту запит. Ми дійсно прагнемо до того, щоб штучний загальний інтелект отримав позитивну оцінку в широких масах. Але для цього не потрібно, щоб кожен бот імітував його. Поки бот виконує завдання, для спрощення якій він призначений, ми повинні бути задоволені.

Проте навіть зі скоректованими очікуваннями важко передати команду, яка містить глибокий та комплексний запит типу " Покажи повідомлення Лізи з нагадуванням про те, що сталося в моїй останній замітці" або навіть продовження розмови. Ботам важко аналізувати такі конструкції, але машинне навчання допомагає витягувати об'єкти та частини міток, однак це

вимагає навчання їх на специфічних для домену даних, а для багатьох доменів важко знайти необхідний обсяг інформації. Без штучного інтелекту, що дійсно живить ботів, боти стають текстовою версією автоматичної служби підтримки мобільних операторів, яка говорить: "натисніть 1 для меню, натисніть 2 для бронювання скарги, натисніть 3 для перегляду інформації про поточний тариф". але більш природно і у вигляді текстових повідомлень. Проте, якими б примітивними не були б чат-боти, вони роблять сферу послуг більш доступною. Далі опишемо різні проблеми, які все ще стоять на шляху розуміння і генерації природньої мови. Це набір загальних пунктів, які повинні виконуватись, щоб системи відчували себе природними.

A) Усвідомлення контексту

Розмовні агенти зараз не усвідомлюють контексту. Якщо користувач каже "Напиши Максу і скажи йому, що час вечірньої події змінено на 7 вечора". Агент повинен мати можливість перевіряти календар для подій або запитувати, про яку подію користувач говорить. По-друге, якщо користувач каже: "Я хотів би замовити чорну каву середнього розміру", а потім скаже "Змінити замовлення на великий розмір", агент повинен мати можливість визначити, який фактор повинен бути змінений.

B) Різноманітність відповідей:

Агенти зараз не мають варіативності та різноманітності, коли вони базуються на правилах або коли вони навчаються витягувати з певного набору відповідей. Можна розробити генеративні алгоритми, які можуть генерувати різноманітні відповіді на основі ситуації після вивчення набору попередньо сформованих відповідей, зберігаючи необхідний сенс.

C) Відповіді засновані на задумах та намірах

Створення відповідей, що базуються на намірі, дасть можливість видавати різноманітні відповіді і зменшить потребу в специфічних даних даної області. Оскільки відповіді не будуються на основі даних тренувань, то вони будуються на подіях. Бот знає інформацію, яку він повинен передати, і передає інформацію за допомогою динамічно згенерованого відгуку.

D) Особистість

Оскільки розмовні агенти розповсюджуються, ми хочемо, щоб вони були більш людськими. Одним з аспектів гуманізації ботів буде їхня особистість. Попередньо навчені моделі реагують відповідно до відповідей, на яких він навчається. Для особистості всі відповіді в навчальному наборі повинні бути введені відповідно до особистості. Це було б досить затратно. Отже, треба розробити способи генерування відповідей відповідно до заданого набору характеристик, що допоможе зробити ботів більш людськими.

E) Поінформованість користувачів:

Розмовні агенти, особливо внутрішньоособистісні боти, повинні знати своїх користувачів. Вони повинні пам'ятати, хто є користувачем і які його переваги та вподобання. Можна легко інтегрувати вподобання в бот, якщо він підключений або має доступ до профілю соціальних мереж користувача, але це буде важко зробити на основі лише розмовної інформації. Необхідні дослідження для вилучення користувальницької персони на основі розмов користувачів та вбудовування інформації про користувача в відповіді бота. Це не тільки зробить бот більш людяним, а й змусить користувача відчути себе пов'язаним з ботом на особистому рівні.

F) Безперервність розмови:

Для віртуальних помічників, які прагнуть бути друзями або супутниками користувача, бот повинен мати можливість розмовляти, коли користувач продовжує тему, яку він сам і обірвав за певний час до цього. Наприклад, якщо користувач говорив про предмет, а потім каже: "Чи пам'ятаєте ви, що чашка, про яку я говорив, мала блакитну ручку?". Бот повинен мати можливість згадати всі інші властивості, що пов'язані зі згаданою користувачем чашкою.

G) Розповідь та наратив :

Бот повинен вміти описувати послідовність подій в порядку їх виникнення. Наприклад, якщо користувач розмістив замовлення на сайті, а

потім попросив свого бота скасувати його, бот повинен мати можливість розповісти про свої дії. Або, якщо користувач запитує бота про події сьогоднішнього дня, бот повинен мати можливість з'єднати всі події разом і розповісти їх послідовно.

1.5 Висновки

У цьому розділі обґрунтовано актуальність використання чат-ботів необхідністю автоматизації в сфері послуг, миттєвою реакцією на запит та інтуїтивно зрозумілим інтерфейсом та незалежністю від людського оператора.

Також проаналізовано класифікацію чат-ботів з прикладами застосування, плюсами та мінусами використання, та рекомендаціями щодо застосування. Для поставленої задачі бронювання квитків в кінотеатр обрано міжособистісний чат-бот з закритим доменом та на основі завдань, який використовує гібридний метод генерування відповідей. Це обґрунтовано тим, що нам необхідна зосередженість на вдалому бронюванні квитків, що бот знаходиться в діапазоні соціальних відстаней на діаграмі проксеміки. А також тим, що бот хоч і виконує інформативну функцію, має доступ до бази знань та симулює мову людини, але виконує конкретне завдання бронювання квитків та може робити дії лише в контексті поставленої задачі

Чат-боти у майбутньому не будуть обмежені вікном месенджера, це універсальний метод комунікацій та задоволення людських потреб, що буде розвиватись та вдосконалюватись. Це обгрунтовано тим, що вищезгадана архітектура може бути узагальнена для всіх систем взаємодії. Форми спілкування будуть змінюватися з часом, від тексту з емодзі до комунікацій за допомогою нейрозв’язків, але те, що залишатиметься незмінним – це наша потреба передати свої думки. Боти є початком інтерфейсу між людьми і загальним штучним інтелектом. Архітектури і дослідженні інструменти

допомагають нам зрозуміти, якого роду бот повинен бути побудований для нашої системи, чого очікувати від ботів і які інструменти використовувати для його побудови.

РОЗДІЛ 2
ВИБІР І ОПИС МАТЕМАТИЧНИХ МОДЕЛЕЙ ДЛЯ ВИБРАНИХ ПРОЦЕСІВ

2.1 Аналіз математичних основ задачі

На рисунку 2.1 показано діалог «людина-агент» у процесі прийняття рішення. Приклад ілюструє види проблем, які, як очікується, вирішить чат-бот:

Turn	Dialogue
1	*usr:* Good morning!
	agt: Good morning, I hope you had a great weekend.
2	*usr:* not really.
	agt: You sound depressed. What can I do for you?
3	*usr:* I'd like to know where sales are lagging behind our forecast?
	agt: The worst region is [country], where sales are 15% below projections.
4	*usr:* Do you know why?
	agt: The forecast for [product] growth was overly optimistic.
5	*usr:* How can we turn this around?
	agt: Here are the 10 customers in [country] with the most growth potential, per our CRM model.
6	*usr:* Can you set up a meeting with the CTO of [company]?
	agt: Yes, I've set up a meeting with [person name] for next month when you are in [location].
7	*usr:* Thanks!

Рисунок 2.1 - Діалог людини та агента під час процесу прийняття рішення

a) відповідь на запитання: агент повинен надавати стислі, прямі відповіді на запити користувачів на основі знань, отриманих з різних джерел даних, включаючи текстові колекції, веб-документи та попередньо скомпільовані бази знань, такі як таблиці зі знижками в прикладі, що показаний у пунктах 3 - 5 на рисунку 2.1.

b) завершення завдання: агент повинен виконати завдання користувача, такі як : резервування ресторану, планування зустрічей (наприклад, пункти 6,7 на рисунку 2.1) і планування ділових поїздок.

c) соціальний чат: агент повинен безперешкодно і належним чином спілкуватися з користувачами - подібно до людини, а також надавати корисні рекомендації (наприклад, пункти 1, 2 на рисунку 2.1).

2.1.1Діалог як метод прийняття рішень

Можна уявити, що діалог на рисунку 2.1 може бути колективно виконаний набором агентів, також відомих як боти, кожен з яких призначений для вирішення конкретного типу завдання.

Тому діалог на рисунку 1.1 можна сформулювати як процес прийняття рішень. Він має природну ієрархію: процес верхнього рівня вибирає, який агент активуватиметься для певної підзадачі (наприклад, відповідь на запитання, планування зустрічі, надання рекомендацій або просто випадковий чат), а також процес низького рівня, керований вибраним агентом, вибирає примітивні дії для завершення підзавдання.

Такі ієрархічні процеси прийняття рішень можуть бути представлені у вигляді певних математичних структур варіантів, наприклад, Марковські процеса прийняття рішень (MDP), де варіанти узагальнюють примітивні дії до дій вищого рівня. Це розширення традиційної конфігурації MDP, де агент може вибирати лише примітивну дію на кожному кроці часу, в якому агент може вибирати "багатоступеневу" дію, яка, наприклад, може бути послідовністю примітивних дій для завершення підзадачі. .

Якщо розглядати кожен варіант як дію, то до системи можна застосовувати методи навчання з підкріпленням. Агент діалогу здійснює навігацію в MDP, взаємодіючі зі своїм середовищем через послідовність дискретних кроків[2]. На кожному кроці агент спостерігає за поточним станом і вибирає дію відповідно до стратегії. Потім агент отримує винагороду і

[2] У нашому контексті кроком можна вважати одну репліку користувача чи чат-боту.

спостерігає за новим станом, продовжуючи цикл, поки епізод не закінчиться. Метою навчання діалогу є пошук оптимальної стратегії для максимізації очікуваних винагород. Таблиця 2.1 формулює зразок діалогових агентів, що використовують цей вищезазначений загальний вигляд RL, де простір дії характеризує складність проблем, а винагорода - це цільові функції, які необхідно оптимізувати.

Подібний погляд на ієрархічні MDP вже застосовувався для розробки деяких широкомасштабних чат-ботів загального призначення. Останніми прикладами є Sounding Board 3, соціальний чат-бот, який виграв у 2017 році Amazon Alexa Prize, і Microsoft XiaoIce 4, можливо, найпопулярніший соціальний чат-бот, який привернув понад 660 мільйонів користувачів по всьому світу з моменту його випуску в 2014 році. Обидві системи використовують ієрархічний менеджер діалогів: майстер (верхній рівень), який керує загальним процесом розмови, і набір вузькоспрямованих ботів (низького рівня), які обробляють різні типи сегментів розмови (підзадачі).

В таблиці 2.1 показані стан, дія та нагорода для кожного з типів ботів при застосуванні навчання з підкріпленням. Функції винагороди в Таблиці 2.1, які здаються суперечливими (тобто, необхідно мінімізувати CPS для ефективного виконання завдань, але максимізувати CPS для поліпшення взаємодії з користувачем), вказують на те, що необхідно збалансувати довгострокові та короткострокові вигоди при розробці система діалогу. Наприклад, XiaoIce - це соціальний чат-бот, оптимізований для залучення користувачів, але при цьому оснащений більш ніж 230 ботами низького рівня, більшість з яких орієнтовані на забезпечення якості та виконання завдання. XiaoIce оптимізований на очікувану CPS, що відповідає довгостроковій, а не короткостроковій взаємодії.

Таблиця 2.1 Навчання з підкріпленням для діалогів. CPS означає кількість кроків за одну сесію, і визначається як середня кількість фраз між ботом і користувачем під час сесії.

Тип бота	**Стан**	**Дія**	**Нагорода**
Бот для відповідей на запитання	Інформація для розуміння намірів запиту користувача	уточнення питань або відповідь	актуальність відповіді, (мін) CPS
Цільонаправлений бот	Інформаціія для розуміння мети користувача	акт діалогу[3]	коефіцієнт успішності, (мін) CPS
Бот загального призначення	Інформація про історія розмов і намір користувача	відповіді	втягнутість користувачів, виміряна в CPS
Бот високого рівня	Інформація для розуміння намірів користувача верхнього рівня	варіанти	втягнутість користувачів, виміряна в CPS

Незважаючи на те, що RL підтримує уніфіковану структуру ML для побудови агентів діалогу, застосування RL вимагає навчання агента діалогу шляхом взаємодії з реальними користувачами, що може бути дуже витратним у багатьох областях. Отже, на практиці часто використовується гібридний підхід, який поєднує сильні сторони різних методів МЛ. Наприклад, можна використовувати імітацію та / або методи навчання (якщо існує велика кількість людських розмовних корпусів), щоб отримати достатньо якісного агента перед застосуванням RL, щоб продовжити його вдосконалення.

[3] Про акти діалогів буде детальніше описано в розділі 3.2

2.1.2 Основи машинного навчання

Мітчелл (1997) дає широке визначення машинному навчанні, включи в це визначення будь-яку комп'ютерну програму, яка покращує продуктивність при виконанні задачі T, що вимірюється за допомогою P, через досвід E.

Діалог, як показано у Таблиці 2.1, є чітко визначеною проблемою навчання з використанням *T, P* і *E*, яка визначається наступним чином:

a) *T*: розмова є з користувачем, щоб виконати його мету.

b) *P*: сукупна винагорода, визначена в таблиці 2.1.

c) *E*: набір діалогів, кожен з яких є послідовністю взаємодій користувача та агента.

В якості простого прикладу виступає однокроковий агент, який призначений для одноразової відповіді на питання. Він може поліпшити свою ефективність, що виміряна з точністю або релевантністю отриманих відповідей у завданні QA, через досвід пар з запитанням-відповідями, які були промарковані людиною.

Навчання з підкріпленням. Навчання з вчителем застосовується до завдань прогнозування на фіксованому наборі даних. Проте, в інтерактивних проблемах, таких як діалоги[4], може виявитися складним отримати приклади бажаних моделей поведінки, які є правильними і репрезентативними для всіх стратегій, в яких агент повинен діяти. На недосліджених територіях агент повинен навчитися діяти, взаємодіючи з невідомим оточенням самостійно. Ця навчальна парадигма відома як навчання з підкріпленням (RL), де існує петля зворотного зв'язку між агентом і зовнішнім середовищем. Іншими словами, в той час як навчання з вчителем (SL) вивчає попередній досвід, наданий

[4] Як показано в таблиці 2.1, навчання діалогу формулюється як проблема RL, де агент дізнається стратегію π, яка в кожному діалозі діалогу вибирає відповідну дію A з набору на основі станів діалогу s, щоб досягти найбільшої сукупної винагороди.

досвідченим зовнішнім керівником, RL навчається, переживаючи його самостійно. RL відрізняється від SL в декількох важливих аспектах .

a) Компроміс між дослідженням та використанням. У RL, агент повинен збирати винагороди від середовища. Це ставить питання про те, яка експериментальна стратегія призводить до більш ефективного навчання. Агент повинен використовувати те, що він вже знає, для отримання високих винагород, в той же час він повинен досліджувати невідомі стани і дії, щоб зробити кращий вибір дій в майбутньому.

b) Відстрочена винагорода і тимчасове призначення кредитів. У RL інформація про навчання недоступна у вигляді (x, y^*), як у SL. Замість цього середовище надає лише відстрочені винагороди, коли агент виконує послідовність дій. Наприклад, ми не знаємо, чи увінчається діалог успіхом під час виконання завдання, тобто до кінця сесії.. Агент, отже, повинен визначити, яку з дій в послідовності слід віднести до отримання можливої винагороди. Ця проблема відома як тимчасове призначення кредиту.

c) Частково спостережні стани. У багатьох проблемах RL спостереження, що сприймається з середовища на кожному кроці, наприклад, речення користувача на кожному кроці діалогу, надає лише часткову інформацію про весь стан середовища на основі якого агент вибирає наступну дію. Нейронні підходи вивчають глибоку нейронну мережу для представлення стану шляхом кодування всієї інформації, що спостерігається на поточних і минулих етапах, наприклад, всі попередні кроки діалогу і результати пошуку з зовнішніх баз даних.

Центральною проблемою як для SL, так і для RL є генералізація - здатність агента добре працювати на невідомих входах. Було запропоновано багато теорій та алгоритмів навчання для вирішення проблеми, наприклад, для пошуку оптимального компромісу між кількістю наявного досвіду навчання та іншими факторами, щоб уникнути перенавчання. У порівнянні з традиційними методами, нейронні підходи забезпечують потенційно більш

ефективне рішення, використовуючи потужність навчання глибоких нейронних мереж.

2.1.3 Глибоке навчання

Розглянемо проблему класифікації тексту: маркування текстового рядка (наприклад, документа або запиту) до якоїсь категорії, наприклад, «спорт» і «політика». На рисунку 2.2 показано схеми класичного алгоритму машинного навчання та глибокого навчання. Як показано на рисунку 2.2 (ліворуч), класичний алгоритм ML спочатку відображає текстовий рядок у векторне представлення x, використовуючи ручне конструювання ознак(наприклад, n-грами, сутності і фрази тощо), потім навчає лінійний класифікатор з шаром *softmax* для обчислення розподілу міток $y = f(x; W)$, де W – матриця вагів, отримана з даних для тренування з використанням градієнтного спуску для мінімізації кількості неправильно класифікованих елементів. Основні зусилля сфокусовані на проектуванні ознак.

Замість використання ручного проектування ознак x, DL підходи оптимізують представлення ознак та класифікацію за допомогою глибоких нейронних мереж(DNN), як показано на рисунку 2.2 (праворуч). Можна помітити, що DNN складається з двох половин. Верхню половину можна розглядати як лінійний класифікатор, подібний до класичної моделі ML на рисунку 2.2 (ліворуч), за винятком того, що вхідний вектор h не базується на ручному проектуванні ознак, а вивчається за допомогою нижньої половини. DNN, яка може розглядатися як генератор ознак, оптимізований спільно з класифікатором в комплексі. На відміну від класичного ML, спроба розробки класифікатора DL в основному полягає в оптимізації архітектури DNN для ефективного навчання.

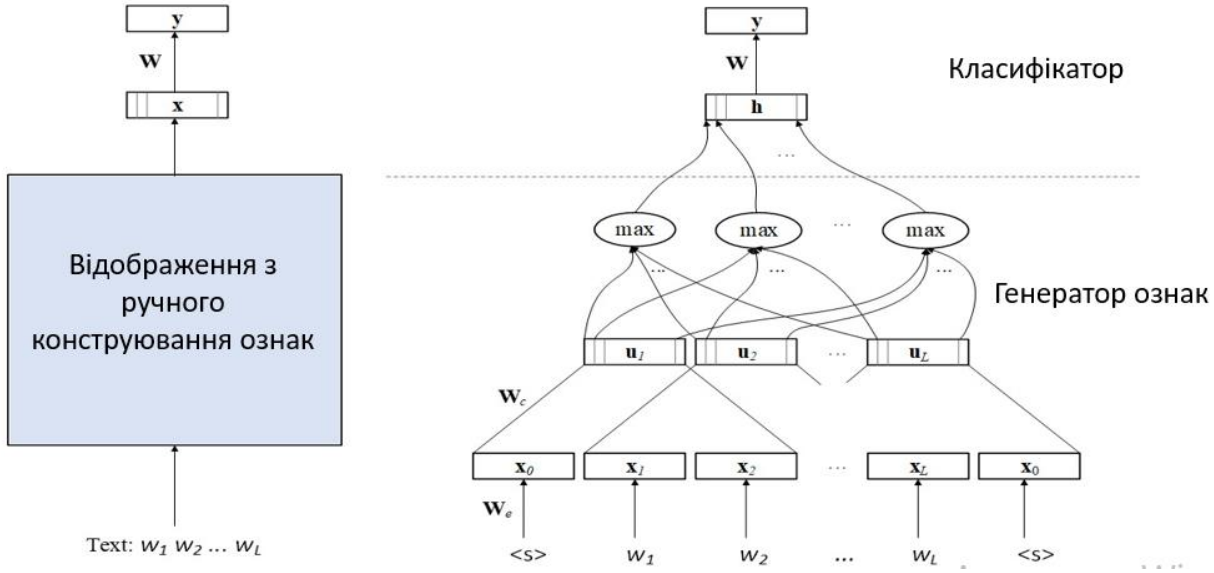


Рисунок 2.2: Блок-схема класичного машинного навчання (ліворуч) і глибокого навчання (праворуч).

Для задач НЛП, залежно від типу лінгвістичних структур, які необхідно захопити у вхідному тексті, можна застосовувати різні типи структур нейронної мережі (NN), такі як : згорткові шари для локальних залежностей слів і рекурентні шари для послідовностей слів. Ці шари можуть бути об'єднані та складені для формування архітектури, що зможе захоплювати різні семантичні та контекстні дані на різних абстрактних рівнях. Далі наведемо кілька широко використовуваних в області NLP шарів NN:

Word Embedding шар. У символічному просторі кожне слово представлено у вигляді одного унітарного вектора, розмірність якого n - розмір попередньо визначеного словника. Словник часто великий, наприклад, $n > 100000$. Ми застосовуємо модель Word Embedding, яка параметризована за допомогою лінійної матриці проекцій $We \in R(n \times m)$, для того, щоб зіставити кожний унітарний вектор з m-мірним дійсним вектором ($m<<n$) в нейронному просторі, де більш семантично близькі один одному Word Embedding вектори знаходяться ближче.

Рекурентні шари. Приклад рекурентних нейронних мереж (RNN) наведено на рисунку 2.3. RNN зазвичай використовуються для вбудовування

та аналізу реченнь, де текст розглядається як послідовність слів, а не набір. Вони перетворюють текст на щільний і маловимірний семантичний вектор, послідовно і рекурентно обробляючи кожне слово, і відображаючи підпослідовність в низьковимірний вектор, як

$$h_i = RNN(x_i, h_{i-1}) := g\,(W_{ih}^T x_i + W_r^T h_{i-1}),$$

де x_i – вбудовування(embedding) і-го слова в текст;

W_{ih} і W_r є навчальними матрицями;

h_i - семантичне представлення послідовності слів до і-го слова.

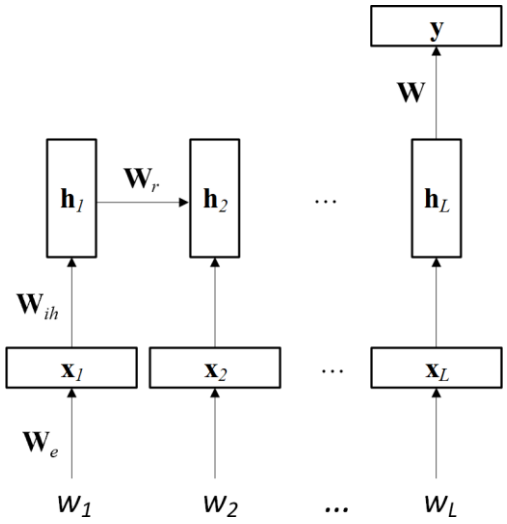


Рисунок 2.3 - Приклад рекурентних нейронних мереж

2.1.5 Приклади моделей

Цей розділ дає короткий опис двох моделей DNN, призначених для завдань ранжування і створення тексту. Вони складаються з шарів NN, описаних в попередньому розділі.

2.1.5.1 DSSM для ранжування.

У задачі ранжування, з урахуванням вхідного запиту *x*, ми хочемо класифікувати всі його відповіді-кандидати *y*, виходячи з функції подібності *sim(x, y)*. Завдання має фундаментальне значення для багатьох додатків IR та NLP, таких як : ранжування запитів-документів, вибір відповідей у QA та вибір відповіді в діалозі.

DSSM означає глибоко структуровані семантичні моделі(Deep Structured Semantic Models) або, більш загально, глибоко структурована модель семантичної подібності (Deep Semantic Similarity Model). DSSM є глибокою моделлю навчання для вимірювання семантичної подібності пари входів *(x, y)*[5] [4]. На рисунку 2.4 показано архітектуру DSSM, яка складається з пари DNN *f1* і *f2*, які відображають входи *x* і *y* в відповідні вектори в загальному маловимірному семантичному просторі. Тоді подібність *x* і *y* вимірюється косинусною відстанню двох векторів. *f1* і *f2* можуть мати різні архітектури в залежності від *x* і *y*. Наприклад, для обчислення подібності пари зображення-текст, *f1* може бути CNN, а *f2* бути RNN.

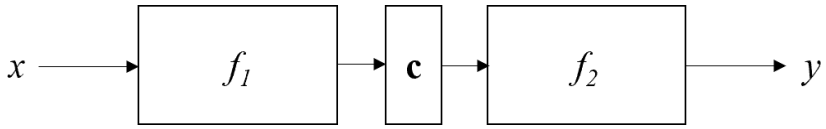


Рисунок 2.4: Архітектура DSSM.

Нехай θ - параметри *f1* і *f2*. θ вчаться ідентифікувати найефективніші представлення ознак *x* і *y*, оптимізованих безпосередньо для кінцевої задачі. Іншими словами, ми вивчаємо прихований семантичний простір, параметризований θ, де семантична відстань між векторами в просторі

[5] DSSM може бути застосований до широкого кола завдань залежно від визначення (x; y). Наприклад, (x; y) може бути парою запит-документ для рейтингу під час веб-пошуку, парою документів в рекомендації , парою питання-відповідь в QA , парою речень на різних мовах в машинному перекладі, а також парою текст-зображення у субтитрах і так далі.

визначається завданням або, більш конкретно, навчальними даними завдання. Наприклад, при ранжуванні Web-документів відстань вимірює релевантність пари запит-документ, а θ оптимізується з використанням парної функції витрат. Розглянемо запит x і два кандидати-документи $y+$ і $y-$, де $y+$ є більш релевантним до x, ніж $y-$. Нехай $\mathrm{sim}\theta(x, y)$ є подібністю x і y в семантичному просторі, параметризованому θ як

$$\mathrm{sim}\theta(x, y) = \cos(f1(x), f2(y)).$$

Хочемо максимізувати

$$\Delta = \mathrm{sim}\theta(x, y+) - \mathrm{sim}\theta(x, y-).$$

Ми робимо це, оптимізуючи гладку функцію втрат

$$L(\Delta; \theta) = \log(1 + \exp(-\gamma\Delta)).$$

де γ - коефіцієнт масштабування стохастичного градієнтного спуску, який позначено на формулі 2.1

$$\theta \leftarrow \theta - \frac{\alpha}{N} \sum_{i=1}^{N} \nabla_\theta L(y_i^*, f(x_i; \theta)) \qquad (2.1)$$

2.1.5.2 Seq2Seq для генерації тексту.

У задачі генерування тексту, що заданий вхідним текстом x, необхідно створити вихідний текст y. Це завдання є фундаментальним для таких додатків, як : машинний переклад та генерація відповідей в діалозі.

Seq2seq позначає архітектуру послідовність до послідовності, яка також відома як архітектура кодера-декодера. На рисунку 2.5 показано архітектуру seq2seq моделі. Seq2Seq зазвичай реалізується на основі послідовних моделей, таких як : RNNs або gated RNNs. Моделі Gate RNNs, такі як : довга короткочасна пам'ять (LSTM) і мережі на основі Gated Recurrent Unit (GRU), є розширеннями схеми RNN на рисунку.2.3 і часто є більш ефективними при захопленні довгострокових залежностей завдяки використання закритих клітин, які мають похідні, що не зникають та не вибухають [3].

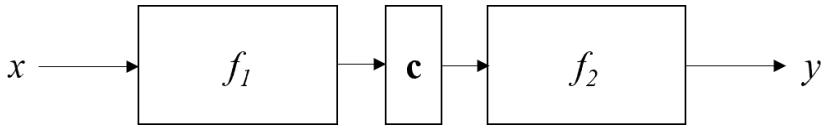


Рисунок 2.5 - Архітектура seq2seq.

Seq2seq визначає ймовірність генерації *y* відносно *x* як *P (y|x)*. Як показано на рисунку 2.5, модель seq2seq складається з (1) вхідного RNN або кодера *f1*, який кодує вхідну послідовність *x* у вектор контексту *c*, як правило в якості простої функції кінцевого прихованого стану; і (2) вихідний RNN або декодер *f2*, який генерує вихідну послідовність *y*, зумовлену *c*. *x* і *y* можуть бути різної довжини. Дві RNN, параметризовані *θ*, проходять спільне навчання, щоб мінімізувати функцію втрат над усіма парами *(x, y)* у навчальних даних.

$$L(\theta) = \frac{1}{M} \sum_{i=1..M} \log - P_\theta(y_i|x_i) \qquad (2.2)$$

У формулі 2.2 визначена функція помилок для заданої RNN

2.2 Навчання з підкріпленням

Навчання з підкріпленням (RL) - це парадигма навчання, де інтелектуальний агент вчиться робити оптимальні рішення, взаємодіючи з невідомим оточенням. У порівнянні з навчанням з вчителя, особливий виклик у RL - це вчитися без заздалегідь відомих правильних результаті. Як далі зазначено в роботі, це призведе до алгоритмічних міркувань, які часто є унікальними для RL [2].

Рисунок 2.5 - Взаємодія між агентом RL і зовнішнім середовищем.

Взаємодія агент-середовище часто моделюється як дискретно-марковський процес прийняття рішень, або MDP, описаний п’ятіркою $M = (S, A, P, R, \gamma)$:

1) S є, можливо, нескінченним набором станів, в яких може перебувати середовище;

2) A - це, можливо, нескінченний набір дій, які агент може прийняти в стані;

3) $P(s`| s, a)$ дає ймовірність переходу середовища в новий стан s` після дії a, що приймається в стані s;

4) $R(s, a)$ - середня винагорода, що негайно отримується агентом після виконання дії a в стані s;

5) $\gamma \in (0, 1)$ - коефіцієнт дисконтування.

Процес може бути записаний як траєкторія *(s1, a1, r1, ...)*, яка генерується на етапі $t = 1, 2, \ldots$ наступним чином:

1) агент спостерігає за поточним станом середовища $s` \in S$ і приймає дію при $a_t \in A$;

2) перехід навколишнього середовища до наступного стану s_{t+1}, розподіленого за ймовірністю переходу $P(\cdot | s_t, a_t)$;

3) пов'язана з переходом безпосередня винагорода $r \in R$, середнє значення якої $R(s, a)$.

Опускаючи індекси, кожен крок призводить до появи кортежу $(s, a, r, s`)$, що називається переходом. Метою агента RL є максимізація довгострокової винагороди шляхом прийняття оптимальних дій (які будуть визначені найближчим часом). Стратегія його вибору дій, позначена π, може бути детермінованою або стохастичною. У будь-якому випадку використовуємо $\pi(s)$ для позначення вибору дії, тобто слідуємо π у стані s. З огляду на стратегію π, значення стану s є середньою дисконтованою довгостроковою винагородою від цієї стратегії:

$$V^{\pi}(s) \coloneqq \mathbb{E}[r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots | s_1 = s, a_i \sim \pi(s_i), \forall i \geq 1]$$

Ми зацікавлені в оптимізації стратегії, щоб V^{π} максимізувався для всіх стратегій. Позначимо через π^* оптимальну стратегію, а V^* відповідне їй значення функції (також відоме як оптимальне значення функції). У багатьох випадках більш зручно для знаходження значення винагороди використовувати іншу функцію, що називається Q-функцією:

$$Q^{\pi}(s, a) \coloneqq \mathbb{E}[r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots | s_1 = s, a_1 = a, a_i \sim \pi(s_i), \forall i > 1],$$

яка вимірює середню дисконтовану довгострокову винагороду, вибираючи спочатку a у стані s, а потім наступну стратегію π. Оптимальна Q-функція, що відповідає оптимальній стратегії, позначається Q^*.

Тепер опишемо два популярних класів алгоритмів на прикладі Q-learning та policy gradient відповідно.

2.2.1 Q- learning.

Перше сімейство базується на спостереженні, що оптимальна стратегія може бути відразу знайдена, якщо доступна оптимальна Q-функція. Зокрема, оптимальну стратегію можна визначити так:

$$\pi^*(s) = \arg max Q^*(s, a)$$

Тому, велике сімейство алгоритмів RL фокусується на вивченні *Q*(s, a)* і колективно називаються методами на основі значень функції.

На практиці важко представляти *Q (s, a)* таблицею з одним записом для кожного окремого *(s, a)*. Наприклад, кількість станів у грі Go більше, ніж 2×10^{170}. Отже, доводиться часто використовувати компактні форми для представлення Q. Зокрема, припускати, що Q-функція має задану параметричну форму, параметризовану деяким вектором θ. Прикладом є лінійна апроксимація:

$$Q(s, a; \theta) = \varphi(s, a)^T \theta, \qquad (2.3)$$

де $\varphi(s, a)$ - d-мірний вручну класифікований вектор для пари дій (s, a);

θ - відповідний вектор коефіцієнтів, який слід вивчити з даних.

Загалом, $Q (s, a; \theta)$ можуть мати різні параметричні форми. Наприклад, у випадку Deep Q-Network (DQN) $Q (s, a; \theta)$ приймає форму глибоких нейронних мереж. Крім того, можна представити Q-функцію

непараметричним шляхом, використовуючи дерева рішень або гауссівські процеси.

Щоб дізнатися Q-функцію, змінюємо параметр θ, використовуючи наступне правило оновлення, після спостереження за переходом стану *(s, a, r, s`)*:

$$\theta \leftarrow \theta + \alpha\big(r + \gamma maxQ(s`, a`;\ \theta) - Q(s, a; \theta)\big)\nabla_\theta Q(s, a; \theta)$$

Вищезгадана зміна відома як Q- learning, який застосовує невелику зміну до параметру θ, яка керована параметром розміру кроку α [5].

Незважаючи на свою популярність, Q-learning часто буває досить нестабільним і вимагає багатьох прикладів, перш ніж досягти гарного наближення до $Q*$. Часто бувають корисними дві модифікації. Перша – це experience replay. Замість використання спостережуваного переходу до оновленого θ використовуючи вищезазначену формулу, можна зберігати параметри в буфері відтворення, і періодично переходити до нього, щоб виконати наступний крок алгоритму Q-learning. Таким чином, кожен перехід можна використовувати кілька разів, що збільшує ефективність моделі. Крім того, це допомагає стабілізувати навчання, запобігаючи занадто швидкій зміні розподілу даних у часі при оновленні параметра θ.

Друга - двохмережева реалізація, приклад більш загального ітеративного алгоритму підбору значення. Тут необхідно зберігати додаткову копію Q-функції, що називається цільовою мережею, яка параметризована за допомогою $\theta target$. Під час навчання $\theta target$ є фіксованим і використовується для обчислення тимчасової різниці для оновлення θ. Зокрема, рівняння тепер приймає такий вигляд:

$$\theta \leftarrow \theta + \alpha\left(r + \gamma maxQ\left(s`, a`;\ \theta_{target}\right) - Q(s, a; \theta)\right)\nabla_\theta Q(s, a; \theta)$$

Періодично *θtarget* оновлюється до значення θ і процес продовжується.

Останнім часом в базовому Q-learning було внесено ряд поліпшень, подібних до описаних вище, таких як : дуельна Q-мережа, подвійний Q-learning і конвергентний SBEED алгоритм.

2.2.2 Policy Gradient.

Інше сімейство алгоритмів намагається оптимізувати стратегію безпосередньо, без необхідності вивчати Q-функцію. Тут сама стратегія безпосередньо параметризована по θ, а $\pi(s; \theta)$ часто є розподілом над діями. При будь-яких θ, стратегія оцінюється за середньою довгостроковою винагородою, що потрапляє в траєкторію довжини

$$H, \tau = (s_1, a_1, r_1, \dots, s_H, a_H, r_H)$$

Якщо можна оцінити градієнт $\nabla_\theta J$

$$J(\theta) := \mathbb{E}\left[\sum_{t=1}^{H} \gamma^{t-1} r_t | a_t \sim \pi(s_t; \theta)\right],$$

по траєкторіям, то можна застосувати метод SGD, щоб максимізувати *J*

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

Один з таких алгоритмів, відомий як REINFORCE, оцінює градієнт наступним чином [14]. Нехай τ – це довжина траєкторія *H*, породжена $\pi(; \theta)$; тобто на $a_t \sim \pi(s_t; \theta)$ для кожного *t*. Тоді стохастичний градієнт на основі цієї єдиної траєкторії задається так:

$$\nabla_\theta J(\theta) = \sum_{t=1}^{H-1} \gamma^{t-1} (\nabla_\theta \log \pi (a_t|s_t;\theta) \sum_{h=t}^{H} \gamma^{h-t} r_h)$$

REINFORCE може сильно страждати на велику дисперсію на практиці, оскільки його градієнтна оцінка залежить безпосередньо від суми винагород по всій траєкторії. Його дисперсія може бути зменшена шляхом використання функції оціночної вартості поточної стратегії, яка часто називається критиком в алгоритмах актора-критика.

$$\nabla_\theta J(\theta) = \sum_{t=1}^{H-1} \gamma^{t-1} (\nabla_\theta \log \pi (a_t|s_t;\theta) \widehat{Q}(s_t, a_t, h))$$

де $\widehat{Q}\ (s, a, h)$ – плановане значення функції для поточної стратегії; $\pi(s;\theta)$, яка використовується для апроксимації

$$\sum_{h=t}^{H} \gamma^{h-t} r_h .\ \hat{Q}(s, a, h)\ ,$$

яку можна дослідитии за допомогою стандартних методів тимчасової різниці (подібно до Q-навчання), але існують і інших багато варіантів.

2.3 Архітектура та модулі чат-бота

Отже, ми готові моделювати розмови між агентом діалогу та користувачем як проблему RL. Тут діалогова система є агентом RL, а користувачем є середовище. На кожному кроці діалогу, агент стежить за станом діалогу, на основі інформації, що була виявила до цього в розмові, а потім робить дію; дія може бути відповіддю користувачеві у вигляді дій,

діалогу або внутрішньої операції, такої як пошук в базі даних або виклик API.

Далі користувач відповідає наступною реплікою, яка буде використовуватися агентом для оновлення стану внутрішнього діалогу на наступному кроці. Потім обчислюється винагорода для вимірювання якості та/або вартості для цього кроку.

Цей процес є взаємодією агент-середовище, яку обговорювали до цього. Тепер обговоримо, як визначається функція винагороди.

Відповідна функція винагороди повинна відображати бажані особливості системи діалогу. У діалогових системах, орієнтованих на виконання конкретних завдань, ми хотіли б, щоб система успішно допомагала користувачеві в мінімально можливій кількості кроків. Отже, природно дати високу винагороду (скажімо +20) в кінці розмови, якщо завдання успішно вирішено, або досить низьку винагорода (скажімо -20) у випадку провалу. Крім того, можна надати невеликий штраф (скажімо, 1 бал) кожному проміжному кроку розмови, щоб заохочувати агента зробити діалог якомога коротшим. Це, звичайно, просто спрощена ілюстрація того, як встановити функцію винагороди для діалогів, орієнтованих на завдання, але на практиці можуть використовуватися більш складні функції винагороди, такі як ті, що вимірюють різноманітність і когерентність розмови. На рисунку 2.6 показана архітектура для багатокрокових цілеспрямованих діалогів.

Для побудови системи часто на практиці застосовується такий шаблон архітектури, зображена на рисунку 2.6. Він складається з наступних модулів.

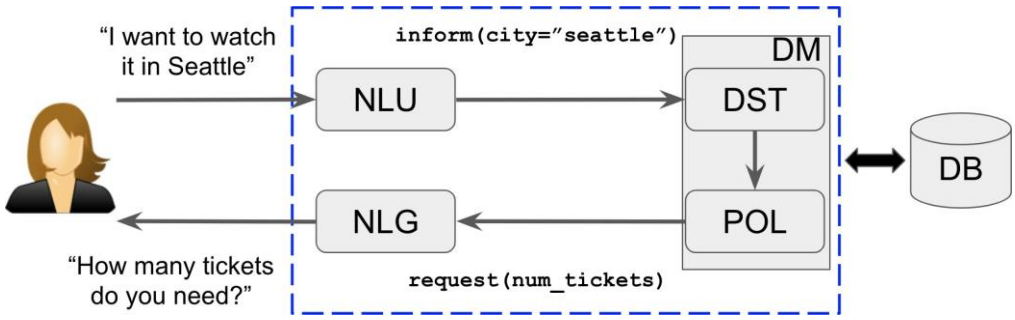


Рисунок 2.6 Архітектура для багатокрокових цілеспрямованих діалогів. Вона складається з наступних модулів: NLU (Natural Language Understanding), DM (Dialogue Manager) і NLG (Generation Natural Language). DM містить два підмодулі: DST (Dialogue State Tracker) і POL (Dialogue Policy). Система діалогу (позначена пунктирним прямокутником) може мати доступ до зовнішньої бази даних (БД)

Розуміння природної мови (NLU): Цей модуль приймає необроблене висловлювання користувача в якості вхідних даних і перетворює його в семантичну форму актів діалогу.

Менеджер діалогу (DM): Цей модуль є центральним контролером системи діалогу. Він часто має підмодуль відстеження стану діалогу (DST), який відповідає за відстеження поточного стану діалогу. Інший підмодуль, який відповідає за вибір стратегії, спирається на внутрішній стан, що наданий DST для вибору дії. Зауважимо, що дія може бути відповіддю користувачеві або деякою операцією з базами даних (наприклад, пошук певної інформації).

Генерація природних мов (NLG): Якщо менеджер діалогу вирішить відповісти користувачеві, цей модуль перетворить цю дію(часто акт діалогу) на природну мову.

2.3.1 Менеджер діалогу

Тут розглянуто невелику частину традиційних підходів з точки зору теоретичних рішень, прийнятих нами в даній роботі.

Раніше розмову розглядали як проблему прийняття рішень. Хоча цей підхід і перспективний, але він припускав, що стан діалогу може приймати лише кінцеве число можливих значень і є повністю спостережуваним (тобто ідеально підходить для DST). Але це припущення часто порушується в реальних додатках, враховуючи неоднозначність у висловлюваннях користувачів і неминучі помилки в NLU.

Для обробки невизначеності, властивої діалоговим системам, Рой та ін. (2000), а також Вільямс і Янг (2007) запропонували використовувати частково спостережуваний процес прийняття рішень Маркова (POMDP) як принципову математичну основу для моделювання та оптимізації систем діалогу [16, 17]. Ідея полягає в тому, щоб сприймати висловлювання користувачів в якості спостережень для підтримки апостеріорного розподілу неспостережуваного стану діалогу; цей розподіл іноді називають "belief state" [15]. Оскільки точна оптимізація в POMDP є обчислювально нерозв'язною, автори вивчали методи апроксимації та альтернативні представлення.

Іншим важливим обмеженням традиційних підходів є те, що кожен модуль на рисунку 2.6 часто оптимізується окремо. Отже, коли система не працює належним чином, може бути важко вирішити проблему «присвоєння кредиту», а саме, визначити, який компонент системи викликає небажану реакцію системи та потребує поліпшення [4]. Справді, як стверджував МакТейр (2002), “ключ до успішної системи діалогу - це інтеграція цих компонентів у робочу систему”. Недавнє поєднання навчання з підкріпленням з диференційованими нейронними моделями

дозволяє оптимізувати систему діалогу від початку до кінця, що потенційно призводить до вищої якості розмови.

2.3.2Модуль обробки природньої мови

Модуль NLU приймає висловлювання користувача в якості вхідних даних і виконує три завдання: виявлення домену, визначення намірів і тегування слотів. Приклад виводу для цих завдань наведено на рисунку 2.7. Як правило, приймається потоковий підхід, при якому ці завдання вирішуються один за одним. Точність та оцінка F1 є двома найпоширенішими показниками, які використовуються для оцінки якості прогнозування моделі. NLU є етапом попередньої обробки для подальших модулів у системах діалогу, якість яких значно впливає на загальну якість системи.

Серед них перші дві задачі часто формулюються як проблеми класифікації, які повинна визначити домен та намір (з попередньо визначеного набору класів) на основі поточного висловлювання користувача. Нейронні підходи до багатокласової класифікації використовуються в новітній літературі і показують кращі результати за традиційні статистичні методи, такі як : умовні випадкові поля і метод опорних векторів. Особливо ефективним є використання стандартних рекурентних нейронних мереж. Для коротких речень, де інформація повинна бути виведена з контексту, було запропоновано використовувати також згорткові нейронні мережі.

Більш складне завдання тегування слотів часто трактується як класифікація послідовностей, де класифікатор прогнозує семантичні мітки класу для підпослідовностей вхідного висловлювання. На рисунку 2.7 показаний приклад висловлювання ATIS (Airline Travel Information

System) у форматі Inside-Outside-Beginning (IOB, де для кожного слова модель передбачений семантичний тег.

W	find	recent	comedies	by	james	cameron
	↓	↓	↓	↓	↓	↓
S	O	B-date	B-genre	O	B-dir	I-dir
D	movies					
I	find_movie					

Рисунок 2.7 - Приклад виводу NLU, де висловлювання (W) використовується для прогнозування домену (D), наміру (I) і тегування слота (S). Використовується представлення IOB.

Модель, яка показана на рисунку 2.8, використовує два набори LSTM клітин, що застосовуються до вхідної послідовності (пряма) і зворотної послідовності (назад).

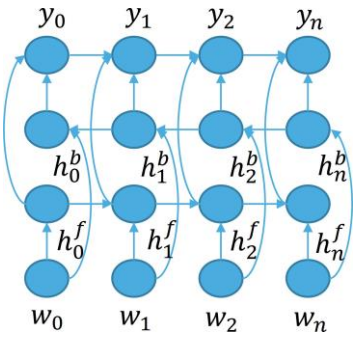


Рисунок 2.8: Модель bLSTM для спільної оптимізації в NLU.

Складені приховані шари прямого і зворотного LSTM використовуються як вхідні дані для іншої нейронної мережі для обчислення вихідної послідовності. Математично операції прямої частини bLSTM визначаються наступним набором рівнянь:

$$i_t = g(W_{wi}W_t + W_{hi}h_{t-1}),$$
$$f_t = g\left(W_{wf}W_t + W_{hf}h_{t-1}\right),$$
$$o_t = g(W_{wo}W_t + W_{ho}h_{t-1}),$$

$$\hat{c}_t = \tanh(W_{wc}W_t + W_{hc}h_{t-1}),$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \hat{c}_t,$$
$$h_t = o_t \odot \tanh(c_t),$$

де h_{t-1} - прихований шар;

W_y- параметри, що навчаються;

$g()$ - сигмоподібна функція.

Як і в стандартних LSTM, i_t, f_t і o_t є вхідним, забутим і вихідним затворами, відповідно. Зворотна частина аналогічна до вхідної.

Для прогнозування тегів, як показано на рисунку 2.7, вхідний w_t часто є унітарним вектором моделі. Вихід при введенні w_t прогнозується згідно з наступним розподілом p_t:

$$p_t = softmax\ (W_{hy}^{(f)}h_t^{(t)} + W_{hy}^{(b)}h_t^{(b)}),$$

де верхні індекси *(f)* і *(b)* позначають передню і зворотну частини bLSTM, відповідно.

Для таких завдань, як : класифікація доменів та намірів, вихідні дані прогнозуються в кінці вхідної послідовності, і можуть використовуватися більш прості архітектури.

У багатьох ситуаціях одне висловлювання може бути неоднозначним або йому бракує всієї необхідної інформації. Очікується, що контексти, які містять інформацію з попередніх висловлювань, допоможуть поліпшити точність моделі. Хорі (2015) розглядав історію бесіди як довгу послідовність слів із змінами ролей (словами від користувача в порівнянні зі словами від системи), і запропонували варіант LSTM з ролевими залежними шарами [18]. Чен (2016) вивчав моделі побудовані на мережах з пам’яттю, які вивчають, якої частини контекстної інформації слід

дотримуватися, коли роблять моделі для тегування. Обидві моделі забезпечили більшу точність, ніж безконтекстні моделі [19].

Хоча три завдання NLU часто вивчаються окремо, існують переваги для спільного їх вирішення (подібно до багатозадачного навчання, що вимагає менше маркованих даних).

2.4 Визначення якості моделі та моделювання користувача

Оцінювання є важливою темою дослідження для систем діалогу. Існують різні підходи, включаючи підходи на основі корпусів, моделювання користувачів, дослідження користувачів, фактичне дослідження користувача тощо.

2.4.1 Метрики

Хоча окремі компоненти в системі діалогу часто можуть бути оптимізовані за допомогою більш чітко визначених метрик, таких як : точність, прецизійність ,оцінки F1 і BLEU, оцінка всієї системи діалогу вимагає більш цілісного погляду і є більш складним завданням. У межах навчання з підкріпленням це означає, що функція винагороди повинна враховувати численні аспекти якості діалогу. На практиці, функція винагороди часто є зваженою лінійною комбінацією підмножини наступних метрик.

Перший клас метрик вимірює успішність виконання завдання. Найбільш поширеним вибором є успішність завдання - частка діалогів, які успішно вирішують проблему користувача (покупка правильних квитків

на кіно, пошук відповідних ресторанів тощо). Фактично, винагорода, що відповідає цій метриці, дорівнює 0 для кожного ходу, за винятком останнього, де вона є +1 для успішного діалогу і -1 в іншому.

Другий клас вимірює витрати, завдані під час діалогу, такі як втрачений час.

Крім того, інші аспекти якості діалогу також можуть бути закодовані у функцію винагороди, хоча це і є відносно недостатньо дослідженим напрямком. У контексті чат-ботів, узгодженість, різноманітність і особисті стилі використовувалися для того, щоб призводити до більш людських подібних діалогів. Вони також можуть бути корисними для діалогів, орієнтованих на виконання конкретних цілей.

2.4.2 Оцінка на основі моделювання

Як правило, алгоритм RL повинен взаємодіяти з користувачем для навчання. Проте запуск RL на найманих або реальних користувачах може бути досить дорогим. Природним способом обійти цю проблему є створення імітованого користувача, за допомогою якого алгоритм RL може взаємодіяти практично без витрат. По суті, змодельований користувач намагається імітувати те, що робить реальний користувач у розмові: він відстежує стан діалогу і спілкується з системою діалогу RL.

Існує багато різних аспектів, що дозволяють класифікувати симулятор користувача, наприклад, детермінований чи стохастичний, на основі контенту чи на основі співпраці, статичні чи нестатичні цілі користувачів під час розмов. Тут виділяємо такі два аспекта:

1). В області деталізації симулятор користувача може працювати або на рівні діалогових дій (також відомому як рівень намірів), або на рівні висловлювань.

2). На рівні методології, симулятор користувача може бути реалізований з використанням підходу, що ґрунтується на правилах, або на основі модельного підходу який передбачає навчання на реальному корпусі текстів.

Agenda-Based симуляція. Кожне моделювання діалогу починається з випадково сформованої мети користувача, яка невідома менеджеру діалогів. Загалом, ціль користувача складається з двох частин: інформаційні слоти, які містять декілька значень, які служать обмеженнями, що користувач хоче накласти на діалог; слоти запиту - слоти, значення яких спочатку невідомі користувачеві і будуть заповнені під час розмови [9]. На рисунку 2.7 показаний приклад цільової задачі користувача в домені бронювання квитків на фільм, в якому користувач намагається придбати 3 квитки на завтра для перегляду фільму «Бетмен».

```
{
  request_slots:
  {
    ticket: UNK
    theater: UNK
    start_time: UNK
  },
  inform_slots:
  {
    number_of_people: 3
    date: tomorrow
    movie_name: batman vs. superman
  }
}
```

Рисунок 2.7 - Приклад цілі користувача в домені бронювання квитка на фільм

Крім того, щоб зробити ціль користувача більш реалістичною, додаються обмеження, специфічні для домену, так що певні слоти повинні з'являтися в цілях користувача. Наприклад, має сенс вимагати від користувача знати кількість квитків, які вона бажає в домені фільму.

Під час діалогу імітований користувач підтримує стекову структуру даних, що називається agenda користувача. Кожен запис у agenda відповідає очікуваному наміру, якого користувач прагне досягти, і їх пріоритети неявно визначаються правилом FILO. Іншими словам , agenda забезпечує зручний спосіб кодування історії розмови та «стану розуму» користувача. Моделювання користувача зводиться до того, щоб підтримувати agenda після кожного кроку діалогу, коли з'являється більше інформації. Для встановлення параметрів у процесі оновлення стека можна використовувати механізми машинного навчання або визначені експертом правила.

Симуляція на основі моделі. Інший підхід до побудови симуляторів користувача повністю базується на даних. Подібно до підходу, що базується на agenda, симуляція також починається із випадково сформованими цілями та обмеженнями користувача. Вони фіксуються під час розмови.

На кожному кроці модель користувача приймає на вхід послідовність контекстів, зібраних до цього в розмові, і виводить наступну дію. Зокрема, контекст на кожному кроці розмови складається з:

a) останньої дії машини,

b) невідповідність між інформацією, що доступна машині, та цілі користувача,

c) статус обмеження;

d) статус запиту.

У цих контекстах LSTM або інші послідовні моделі використовуються для виведення наступного висловлювання користувача. Модель можна вивчити з корпусів діалогу між людьми. На практиці часто можна отримати якісні результати, поєднуючи методи, які засновані на правилах і на основі моделей.

2.4.3 BLEU

BLEU (Bilingual Evaluation Understudy – оцінка розроблена для порівняння запропонованих варіантів перекладу тексту. Не зважаючи на це, вона може бути використана для оцінки тексту, що генерується для вирішення цілої низки завдань, пов'язаних з обробкою природної мови.

BLEU є показником для оцінки генерованого речення відносно базового. Ідеальний збіг дає 1,0 бал, тоді як ідеальна невідповідність - 0.0.

Алгоритм пропонує 5 незаперечних переваг:

1) Швидкий та незатратний розрахунок;
2) Легкий для розуміння;
3) Не залежить від вибору мови;
4) Тісно пов’язаний з оцінкою людини;
5) Отримав широке розповсюдження.

Підхід працює шляхом підрахунку кількості збігів n-грамів у перекладі-кандидаті та n-грамів в базовому текст. Порівняння проводиться незалежно від порядку слів.

Висновки

У цьому розділі було розглянуто математичні основи побудови діалогових систем. Розглянуто діалог як процес прийняття рішень, який можна представити у вигляді Марковських процесів. Було описано необхідні основи машинного навчання у контексті задач NLP, розглянуто спільні та відмінні сторони навчання з підкріпленням та навчання з вчителем. Наведено опис необхідної для побудови чат-бота інформації про глибоке навчання. Зокрема

розглянуто необхідні шари нейронних мереж та показано як їх можна застосовувати на прикладах задачі ранжування та генерації текстів.

Було розглянуто в навчання з підкріпленням. Зокрема наведена інформація про Q-навчання із застосуванням experience replay, що дозволяє зробити систему більш стабільною в процесі навчання. А також Policy gradient з алгоритмом Reinforce, що дозволяє оптимізувати стратегію дії діалогового агента безпосередньо.

Було визначено та описано використання ключових елементів архітектур діалогових систем. Розглянуто менеджер діалогу як ключову сутність майбутнього чат-боту, що повинен містити в собі реалізацію алгоритмів для прийняття рішень. А також розглянуто модуль природньої обробки мови, який виявляє намір, домен та займається тегуванням слотів, використовуючи формат IOB та bLTSM архітектуру нейронних мереж.

В кінці розглянуто можливі метрики для діалогових систем. Також введено та описано роботу і структуру симулятора користувача з agenda модулюванням, який буде заміняти середовище у процесі навчання та тестування чат-боту. В силу обмежених ресурсів в роботі використовується контроль якості лише в процесі навчання агента діалогової системи. А саме будемо відстежувати кількість кроків, що необхідні системі для досягнення визначених симулятором користувача цілей, нагороду, яку агент буде отримувати в процесі навчання, та успішність діалогу, тобто відсоток діалогів, що закінчились виконанням цілей користувача за задану кількість кроків.

РОЗДІЛ 3
ПОБУДОВА ДІАЛОГОВОЇ СИСТЕМИ У ВИГЛЯДІ ЧАТ-БОТУ З ВИКОРИСТАННЯМ МЕТОДІВ ГЛИБОКОГО НАВЧАННЯ З ПІДКРІПЛЕННЯМ

3.1 Обґрунтування вибору платформи та мови реалізації

Мова програмування Python є найбільш популярною для машинного навчання та data science. Елегантний синтаксис Python, динамічна обробка типів, сумісність з багатьма іншими мовами, кількість бібліотек для аналізу даних та побудови моделей а також те, що це інтерпретована мова, роблять її ідеальною для написання скриптів та швидкої розробки прикладних програм у багатьох галузях на більшості платформ. Саме тому для вирішення поставленої задачі було обрано мову програмування Python.

В роботі розрахунки виконувались на платформі Microsoft Windows 10 в середовищах розробки PyCharm та Google Colaboratory .

3.2 Дані

База даних квитків – це набір квитків у кіно з різними атрибутами або слотами. На рисунку3.1 наведено кілька елементів :

```
{
    "0": {
        "city": "hamilton",
        "theater": "manville 12 plex",
        "zip": "08835",
        "critic_rating": "good",
        "genre": "comedy",
        "state": "nj",
        "starttime": "10:30am",
        "date": "tomorrow",
        "moviename": "zootopia"
    },
    "287": {
        "date": "tonight",
        "city": "dallas",
        "theater": "alamo draft house",
        "moviename": "deadpool",
        "starttime": "10:15pm"
    },
    "330": {
        "city": "Monroe",
        "theater": "regal barkley village stadium 16",
        "zip": "98272",
        "distanceconstraints": "near me",
        "state": "wa",
        "starttime": "around 8pm",
        "date": "tonight",
        "moviename": "kung fu panda 3"
    },
    "976": {
        "city": "carbondale",
        "theater": "amc showplace carbondale 8",
        "distanceconstraints": "in your area",
        "genre": "thriller",
        "state": "illinois",
        "other": "before dinner",
        "starttime": "right now",
        "date": "Tuesday",
        "moviename": "the witch"
    },
}
```


Рисунок 3.1 - Приклад записів в базі даних з квитками

Він організований як словник з ключами, які є індексом квитка, а значення - також як словники, що містять інформацію про фільм, який

представляє квиток. Як можна бачити, не всі квитки мають однакові наявні атрибути.

База даних словників: інша база даних містить словник, де ключі - це різні слоти, які можуть бути в квитку, а значення - списки можливих значень для кожного слота. На рисунку 3.2 декілька різних елементів (кількість значень зменшена):

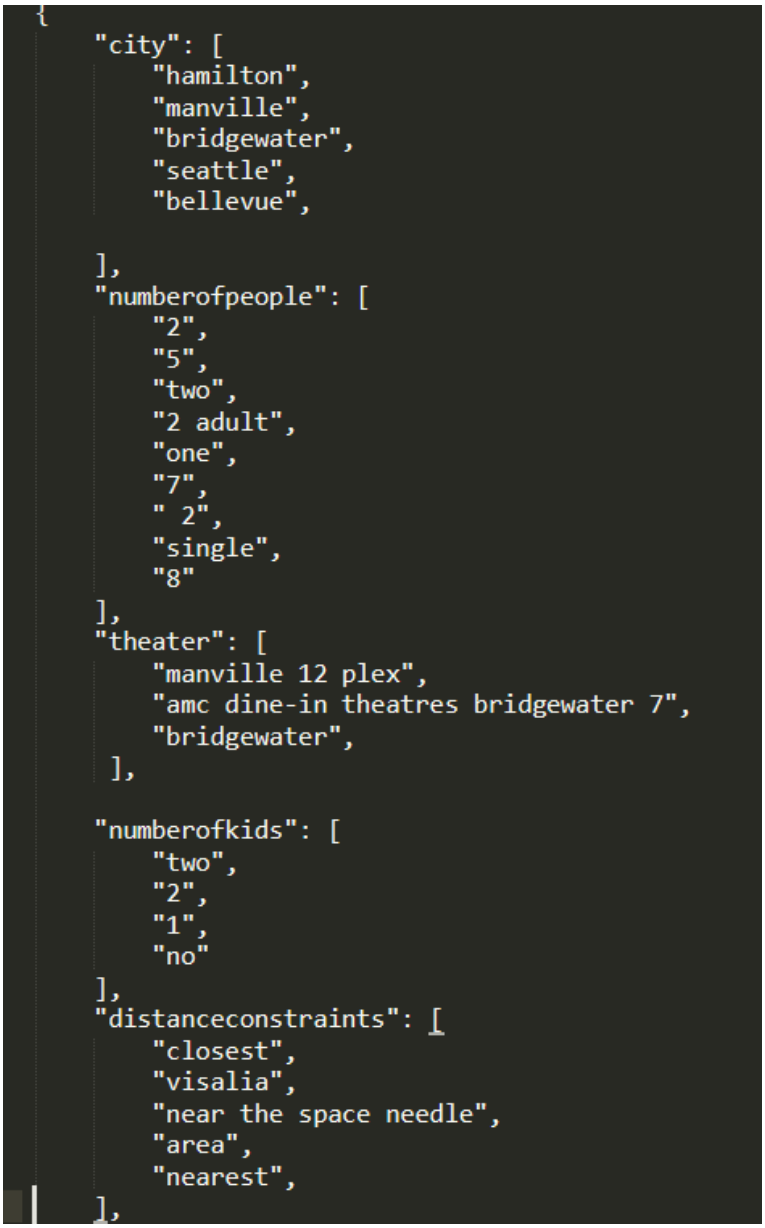


Рисунок 3.2 - Приклад записів в базі даних словників

База даних зі списком цілей користувача: ми маємо цілі користувача як список словників, які містять слоти запитів і повідомлень для кожної мети. Приклад записів наведено на малюнку 3.3

```
{'diaact': 'request',
 'inform_slots': {'city': 'seattle',
                  'date': 'tomorrow',
                  'moviename': 'deadpool',
                  'numberofpeople': '2',
                  'starttime': '9:00 pm',
                  'theater': 'amc pacific place 11 theater'},
 'request_slots': {}}
{'diaact': 'request',
 'inform_slots': {'city': 'portland',
                  'moviename': 'star wars',
                  'numberofpeople': '5',
                  'state': 'oregon'},
 'request_slots': {'date': 'UNK', 'starttime': 'UNK', 'theater': 'UNK'}}
{'diaact': 'request',
 'inform_slots': {'moviename': 'zootopia', 'numberofpeople': 'two'},
 'request_slots': {'date': 'UNK', 'starttime': 'UNK', 'theater': 'UNK'}}
```

Рисунок 3.3 - Приклад цілей користувача

Акти діалогу – семантичний фрейм, що використовується для внутрішнього представлення речення, подається на вхід до агента та симулятора користувача. Акт діалогу представлений у формі словника де ключами є намір, відомі та необхідні слоти. На рисунку 3.4 зображено приклад актів діалогу

```
{'inform_slots': {'city': 'seattle'},
 'intent': 'request',
 'request_slots': {'starttime': 'UNK', 'theater': 'UNK'}}
{'inform_slots': {'moviename': 'the witch'},
 'intent': 'inform',
 'request_slots': {}}
{'inform_slots': {}, 'intent': 'done', 'request_slots': {}}
{'inform_slots': {}, 'intent': 'request', 'request_slots': {'moviename': 'UNK'}}
{'inform_slots': {}, 'intent': 'thanks', 'request_slots': {'theater': 'UNK'}}
```

Рисунок 3.4 - Акти діалогу

В таблиці 3.1 представлений список всіх можливих намірів та значень слотів в актах діалогу

Таблиця 3.1 Наміри та значення слотів

Ключ	Можливі значення
Намір	request, inform, deny, confirm question,confirm answer, greeting, closing, not sure, multiple choice, thanks, welcome, match found
Слот	actor, actress, city, closing, critic rating,date, description, distanceconstraints, greeting, implicit value, movie series, moviename, mpaa rating, numberofpeople, numberofkids, taskcomplete, other, price, seating, starttime, state, theater,theater chain, video format, zip, result, ticket, mc list

Наведемо пояснення до найбільш популярних намірів:

a) Inform: забезпечує обмеження у вигляді інформаційних слотів;

b) Request : запит на заповнення слотів конкретними значеннями;

c) Thanks: використовується лише користувачем; він просто вказує агентові, що той зробив щось якісно або що користувач готовий закінчити розмову;

d) Match Found: вказує користувачу на те, що він має збіг, який, на його думку, виконає поставлене перед ним завдання;

e) Reject: використовується тільки користувачем; використовується тільки у відповідь на дії агента, що відправив match found, і вказує на те, що збіг не відповідає його обмеженням;

f) Done: агент використовує це, щоб закрити бесіду і перевірити, чи виконано поточну мету, користувача використовує цей намір автоматично, коли діалог триває велику кількість кроків;

3.3 Модель для навчання чат-бота

На рисунку 3.2 зображено обрану архітектуру для навчання чат-бота на основі навчання з підкріпленням [8].Спочатку симулюється висловлювання

користувача, потім навчання з підкріпленням використовується для тренування всіх компонентів разом.

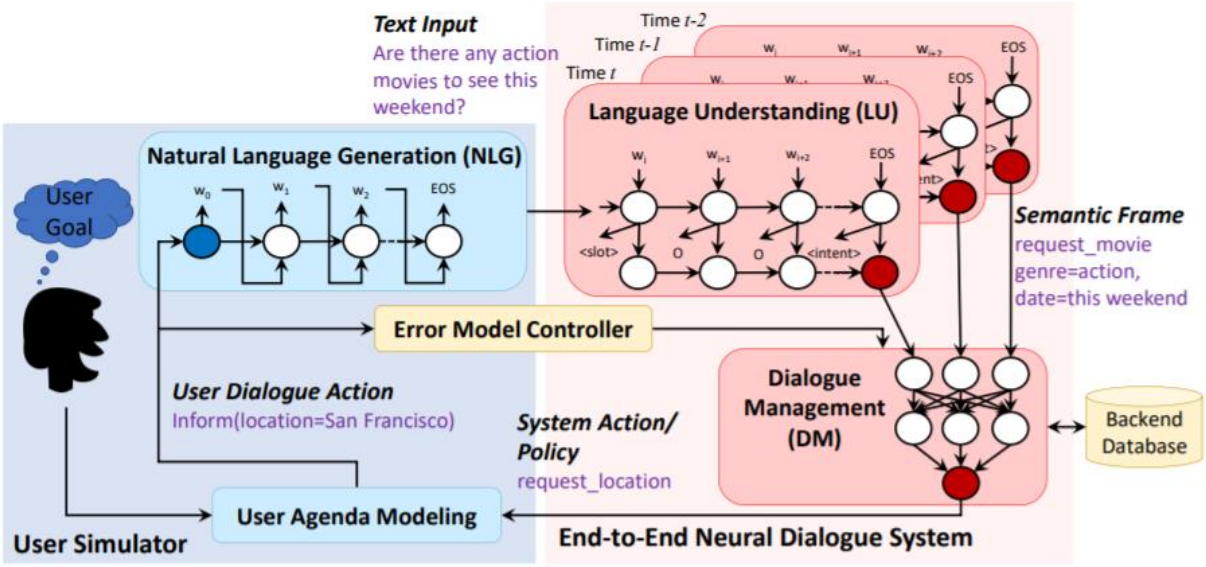


Рисунок 3.2 - Обрана архітектура діалогової системи.

Вона включає в себе симулятор користувача (ліву частину) і нейронну діалогову систему (права частина). В частині стимулювання користувача, використовується agenda модель, яка на основі актів діалогу застосовується для контролю обміну фразами відповідно до згенерованої цілі користувача, для забезпечення того, щоб користувач поводив себе послідовно і цілеспрямовано. Модуль NLG використовується для створення текстів на природній мові, що відповідають актам діалогу користувача. У нейронній діалоговій системі вхідне речення (розпізнаване висловлювання або введений текст) проходить через модуль LU і стає відповідним семантичним фреймом (який є низькорівневим представленням реального висловлювання, він може бути оброблений агентом), а DM, що включає в себе state tracker (трекер стану) і модуль для визначення стратегії, збирає та накоплює семантику з кожного висловлювання, відстежує стан діалогу під час розмови та створює наступну системну дію.

3.3.1 Діалогова система

3.3.1.1. Модуль розуміння мови (LU)

Головне завдання LU - автоматично класифікувати домен запиту користувача разом з наміром і заповнити набір слотів, щоб сформувати семантичний фрейм. Популярний формат IOB (in-out-begin) використовується для представлення тегів слотів, як показано на малюнку2.7(минулий розділ).

$$x = w_1, \dots, w_n, < EOS >$$
$$y = s_1, \dots, s_n, i_m$$

де x - вхідна послідовність слів,

y містить асоційовані слоти *sk*, та намір речення *i*.

Компонент LU реалізується за допомогою єдиного LSTM, який одночасно виконує прогнозування намірів і заповнення слотів *y = LSTM(x).*

Метою LU є максимізація умовної ймовірності y (слоти і наміри) за умови x (послідовність слів):

$$p(y|x) = (\prod_{i}^{n} p(s_i\ |w_{1,\dots,w_i}))p\ (i_m|y)$$

Ваги моделі LSTM навчаються, використовуючи алгоритм зворотного поширення помилки для максимізації умовної ймовірності міток навчального набору. Прогнозований набір тегів є конкатенированим набором тегів формату IOB і тегу для намірів. Таким чином, цю модель можна навчити, використовуючи всі доступні акти діалогу та парні висловлювання в нашому маркованому наборі даних за допомогою навчання з вчителем.

3.3.1.2 Діалоговий менеджер (DM)

Символічний вихід LU передається DM у формі акту діалогу (або іншого семантичного фрейму). Класична DM включає два етапи(модулі) : відстеження стану діалогу (state tracking) та вибору стратегії (policy learning).

Відстеження стану діалогу: враховуючи сигнальний вихід LU, наприклад запит *(moviename; genre = action; date = this weekend)*, трекер виконує три основні функції: формує символічний фрейм для взаємодії з базою даних для отримання наявних результатів; оновлює стан трекера на основі доступних результатів з бази даних і останніх дій користувача діалогу; підготовує представлення стану s_t для передачі до етапу вибору стратегії.

Вибір стратегії: представлення стану для вибору стратегії включає в себе дії користувача (наприклад *request(moviename; genre=action; date=this weekend)*), останню дію агента *(request(location)*), доступні записи з бази даних, інформація поточний крок діалогу, історія кроків діалогу і т.п. Агент на основі цих даних приймає рішення про вибір наступної дії, використовуючи глибоку Q-мережу.

3.3.2 Моделювання користувача

Для того, щоб здійснювати комплексне навчання для всіх модулів запропонованої системи діалогу, потрібен симулятор користувача для автоматичної і природної взаємодії з системою діалогу. В обраній архітектурі спочатку генерується ціль користувача. Агент не знає цілі користувача, але намагається допомогти користувачеві виконати її в ході розмови. Отже, весь обмін реченнями навколо цієї мети є неявним. Цілі користувача складаються з двох частин: *inform_slot* зі значеннями, які є обмеженнями від

користувача(тобто уже відомою йому інформацією), і *request slots* для слотів, про значення яких користувач не має інформації, але хоче отримати значення від агента під час розмови . Цілі користувача генеруються за допомогою набору помічених розмовних даних.

3.3.2.1 Agenta моделювання

Під час діалогу симулятор користувача зберігає стек (agenta) , де користувальницький стан s_u включений до agenta A і цілі t. Мета складається з обмежень C і запиту R. На кожному етапі часу t симулятор користувача генерує наступну дію користувача a_{ut} на основі поточного стану s_{ut} і останнього дії агента $a_{m,t-1}$, а потім оновлює поточний стан $s_{ju.}$

3.3.2.2 Генерація природних мов (NLG):

Враховуючи акти діалогу користувача, модуль NLG генерує текст на англійській мові. Щоб контролювати якість моделювання користувача з урахуванням обмежень в промаркованих даних, застосовується гібридний підхід, що включає шаблонну NLG модель та модель де NLG навчається на маркованому наборі даних разом seq_to_seq мережею. Вона приймає акти діалогу і генерує ескіз речення, при цьому залишеє заповнювачі(порожні забронювані місця), за допомогою декодер LSTM. Потім виконується сканування з обробкою для заміни заповнювачів слотів на їх фактичні значення. У LSTM-декодері застосовується променевий пошук, який ітеративно розглядає найкращі підречення [8].

У цій гібридній моделі, якщо акт діалогу користувача можна знайти в попередньо визначених шаблонах речень, застосовується NLG на основі шаблону; в іншому випадку висловлювання генерується на основі моделі NLG. Цей гібридний підхід дозволяє розробнику системи діалогу легко удосконалювати NLG, надаючи шаблони речень, з якими погано справляються навчена модель.

3.3.3 Навчання агента

Розглянемо алгоритм навчання агента для обраної архітектури. На рисунку 3.5 зображена діаграма, яка являє собою один крок в циклі навчання агента (для спрощення ігнорується модулі NLU та NLG). Маємо 4 основні частини цієї системи - агент dqn_agent, трекер стану діалогу state_tracker, користувач user sim. (або симулятор користувача) і контролер помилок моделі emc. Опишемо основні етапи навчання:

1) Отримати поточний стан, який еквівалентний попередньому стану (пункт 6 на рисунку 3.6) або початковому стану, якщо це початок епізоду, і надіслати його як вхід до методу дії агента state_to_action.

2) Отримати дію агента та надіслати її до методу update в трекері стану, він оновлює свою власну історію поточної бесіди в цьому методі, а також оновлює дію агента з інформацією про запит до бази

3) Отримати поточний стан, який еквівалентний попередньому стану (пункт 6 на рисунку 3.6) або початковому стану, якщо це початок епізоду, і надіслати його як вхід до методу дії агента state_to_action

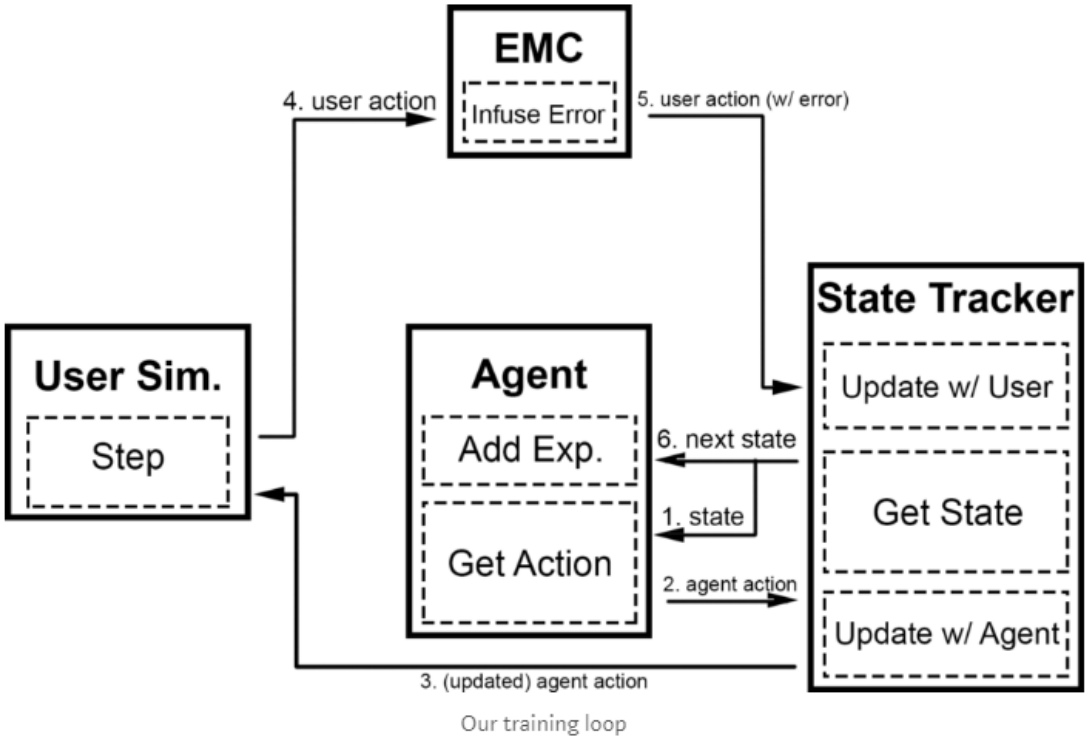


Рисунок 3.3 - Діаграма спрощеного алгоритму навчання агента

4) Відповідь користувача передається до emc.

5) Дія користувача з помилкою надсилається як вхід до методу update в трекері стану для дій користувача, який просто зберігає інформацію в своїй історії, але не оновлює і не доповнює дії користувача

6) Нарешті, наступний стан виводиться з методу Get_state_for_agent в трекері стану.

Важливо відзначити, що, як і в будь-якому DQN-агенті, буфер пам'яті чатково заповнюється на стадії «warm-up». На відміну від багатьох застосувань для DQN в іграх агент не приймає випадкових дій на цій стадії. Замість цього під час розігріву використовується спрощений алгоритм на основі правил.

3.4 Діаграма класів програмного продукту

На рисунку 3.4 представлена діаграма класів чат-бота

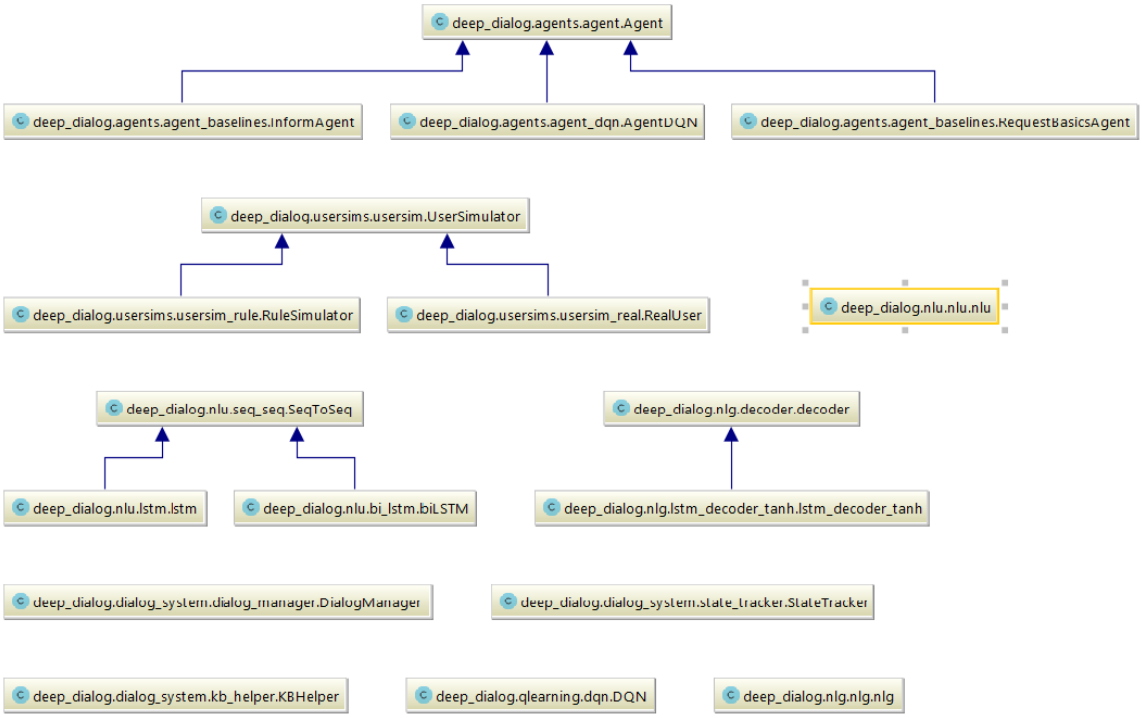


Рисунок 3.4 Діаграма класів чат-бота

Розглянемо деякі класи більш детально

На рисунку 3.5 зображено uml діаграму AgentDQN класу, в якому реалізований вибір стратегії для діалогової системи

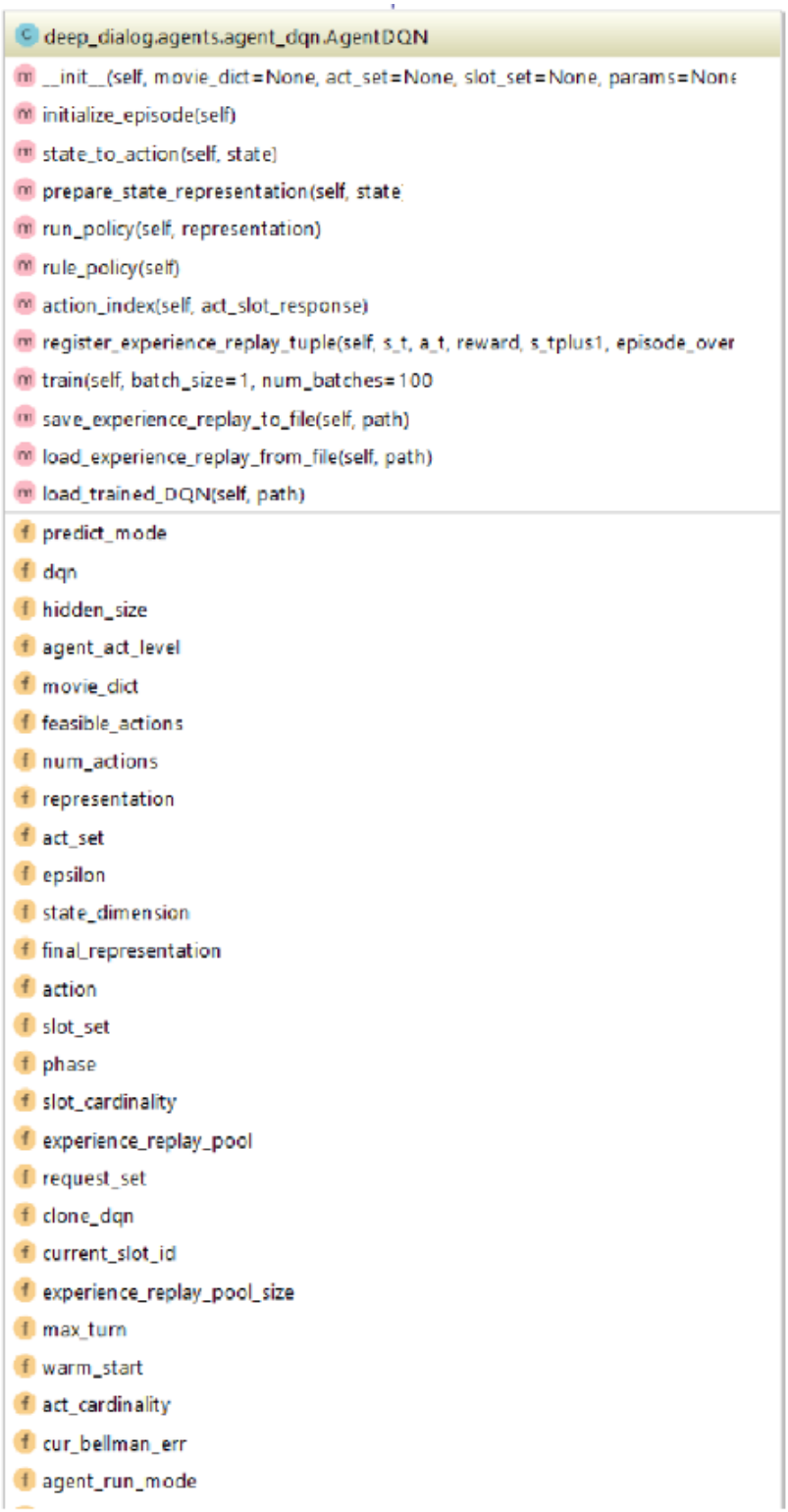


Рисунок 3.5 - Uml діаграма класу AgentDQN

В ньому реалізовані такі методи та функції:

1) initialize_episode(self) - ініціалізує новий епізод. Ця функція викликається кожного разу, коли запускається новий епізод діалогу.

2) train(self, batch_size=1, num_batches=100) – для тренування DQN з experience replay.

3) state_to_action(self, state) – приймає інформацію про стан діалогу, описаний в пункті 3.3.1.2, запускає необхідні функції для навчання та вибору необхідної стратегії, яку потім повертає

4) register_experience_replay_tuple(self, s_t, a_t, reward, s_tplus1, episode_over)- реєструє зворотний зв'язок з середовища, який зберігається як майбутні навчальні дані;

5) run_policy(self, representation) – запускає методи вибору стратегії, які реалізовані в класі DQN

6) prepare_state_representation(self, state)- створює необхідне представлення для кожного стану.

На рисунку 3.6 зображено UML діаграму StateTracker класу, який зберігає записи про заповнення слотів запитів і заповнює інформаційні слоти.

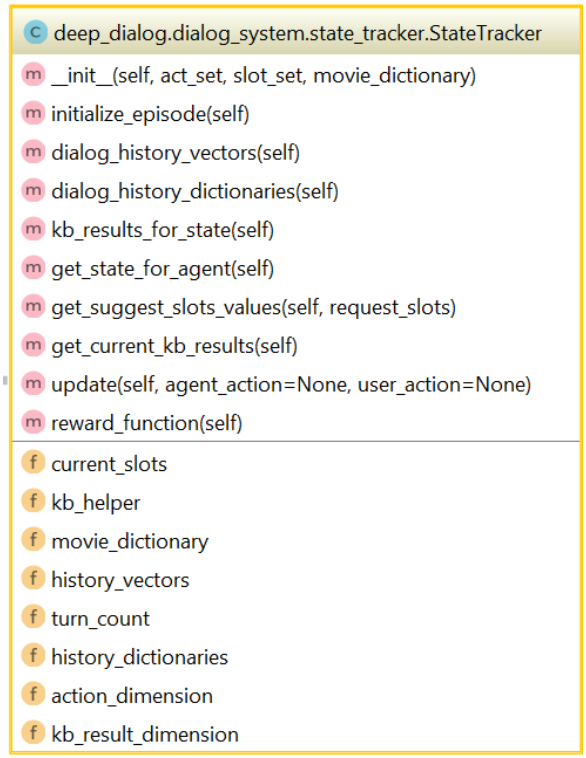


Рисунок 3.6 - Uml діаграма класу StateTracker

В ньому реалізовані такі методи та функції:

1) kb_results_for_state(self)- повертає інформацію про відповідні записи з бази даних на основі відомих слотів за допомогою методів класу KBHelper

2) dialog_history_dictionaries(self) - повертає історію діалогу у вигляді словника.

3) get_suggest_slots_values(self, request_slots) - отримує запропоновані значення для слотів запиту.

4) update(self, agent_action=None, user_action=None) - оновлює стан на основі останньої дії

На рисунку 3.7 зображено UML діаграму DialogManager класу, який опосередковує взаємодію між агентом і клієнтом.

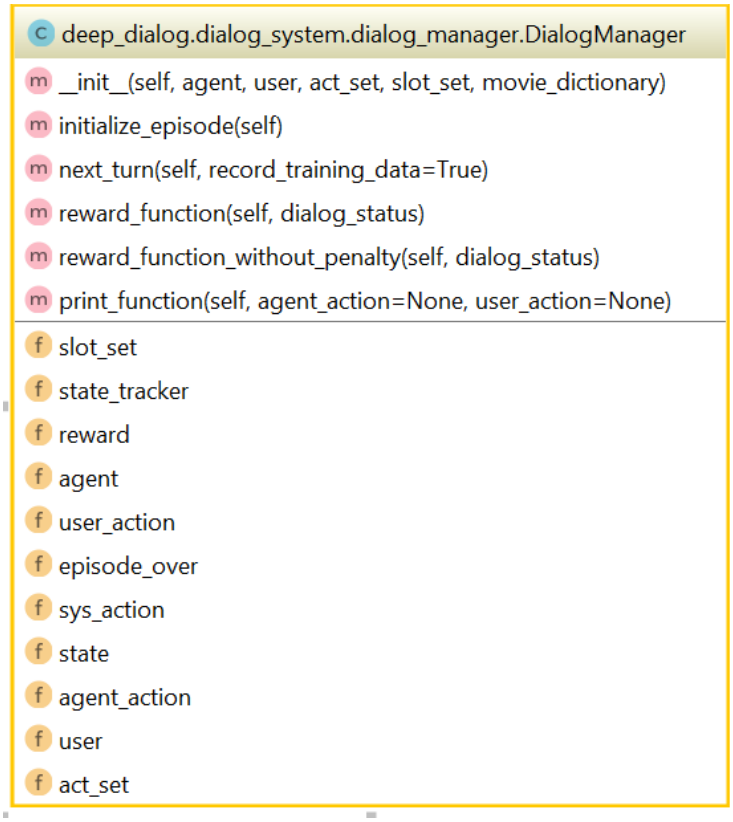


Рисунок 3.7 Uml діаграма класу DialogManager

В ньому реалізовані такі методи та функції:

1) next_turn(self, record_training_data=True)- ця функція ініціює кожний наступний інформацією між агентом і користувачем, тобто фактично реалізує алгоритм описаний в пункті 3.3.3

2) reward_function(self, dialog_status)- функція винагороди, що обчилюється на основі даних в змінній dialog_status.

3) print_function(self, agent_action=None, user_action=None)- функція виводу діалогу та його результатів.

На рисунку 3.8 зображено UML діаграму RuleSimulator класу, який реалізовує логіку симулятора користувача для взаємодії з агентом.

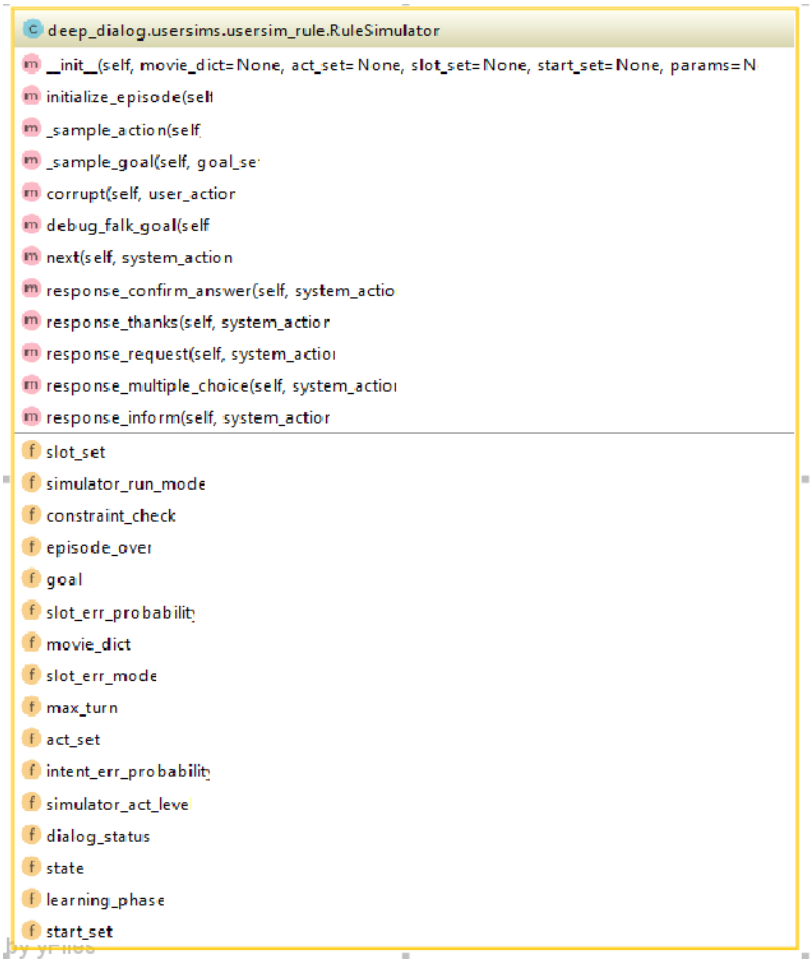


Рисунок 3.8 - Uml діаграма класу RuleSimulator

В ньому реалізовані такі методи та функції:

1) _sample_action(self)- випадково обирає початкову дію на основі цілі користувача

2) next(self, system_action) - генерує наступну дію користувача на основі останньої дії системи

3) response_thanks(self, system_action)- допоміжна функція для генерації відповіді після вхідного повідомлення з подякою.

3.5 Результати роботи та їх аналіз

Було реалізовано чат-бот з використанням вищеописаної архітектури, який демонструє роботу діалогової системи та симуляції користувача у вигляді їх діалогу. Для реалізації інтерфейсу було вирішено створити бота в месенджері Telegram та розгорнути чат-бот за допомогою сервісу Heroku, для забезпечення безперервного доступу та можливості використання програми з будь-якого гаджету. Для демонстрації роботи чат-бота достатньо вбити в пошук в Telegram ‘@itry_bot’ та натиснути кнопку start, що продемонстровано на рисунку 3.9. Для запуску діалогу між діалоговою системою та симулятором користувача необхідно відправити чат-бота команду ‘/try’. Бот у відповідь відправить відразу згенеровану ціль для симуляції користувача для контролю якості виконання діалогу та сам діалог, що продемонстровано на рисунку 3.10. У випадку задовільної роботи діалогової системи (тобто задовільнення цілей користувача менше ніж за 40 кроків діалогу) бот відправить повідомлення про успішний діалог, у зворотньому про провалений.

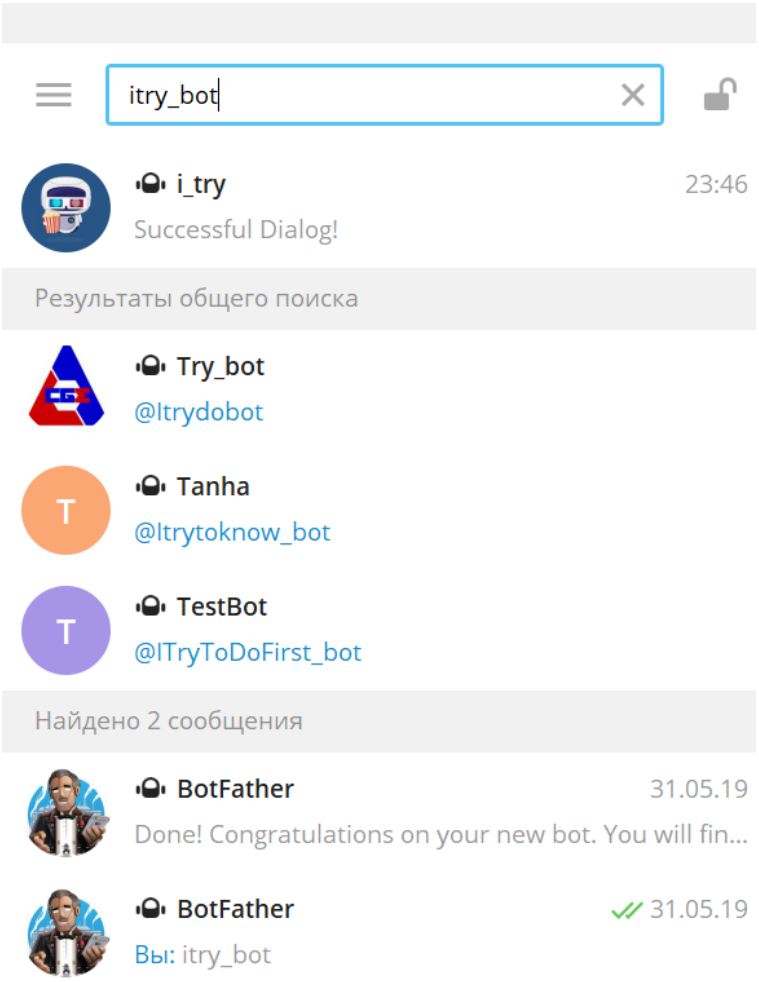


Рисунок 3.9 - Пошук бота в Telegram

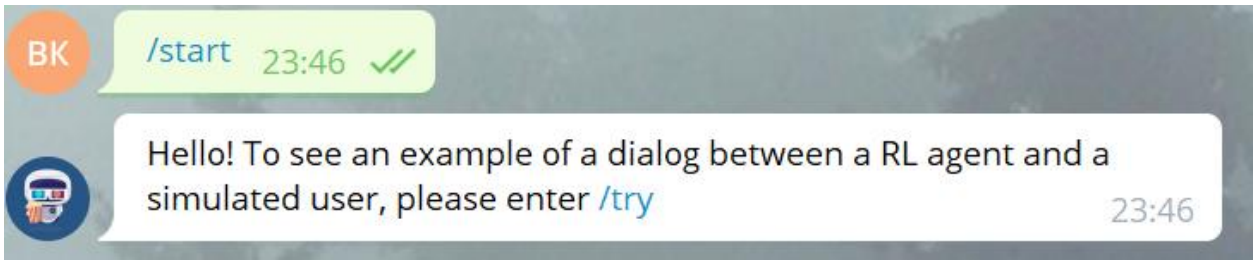


Рисунок 3.10 - Запуск бота

На рисунку 3.11 зображений опис бота

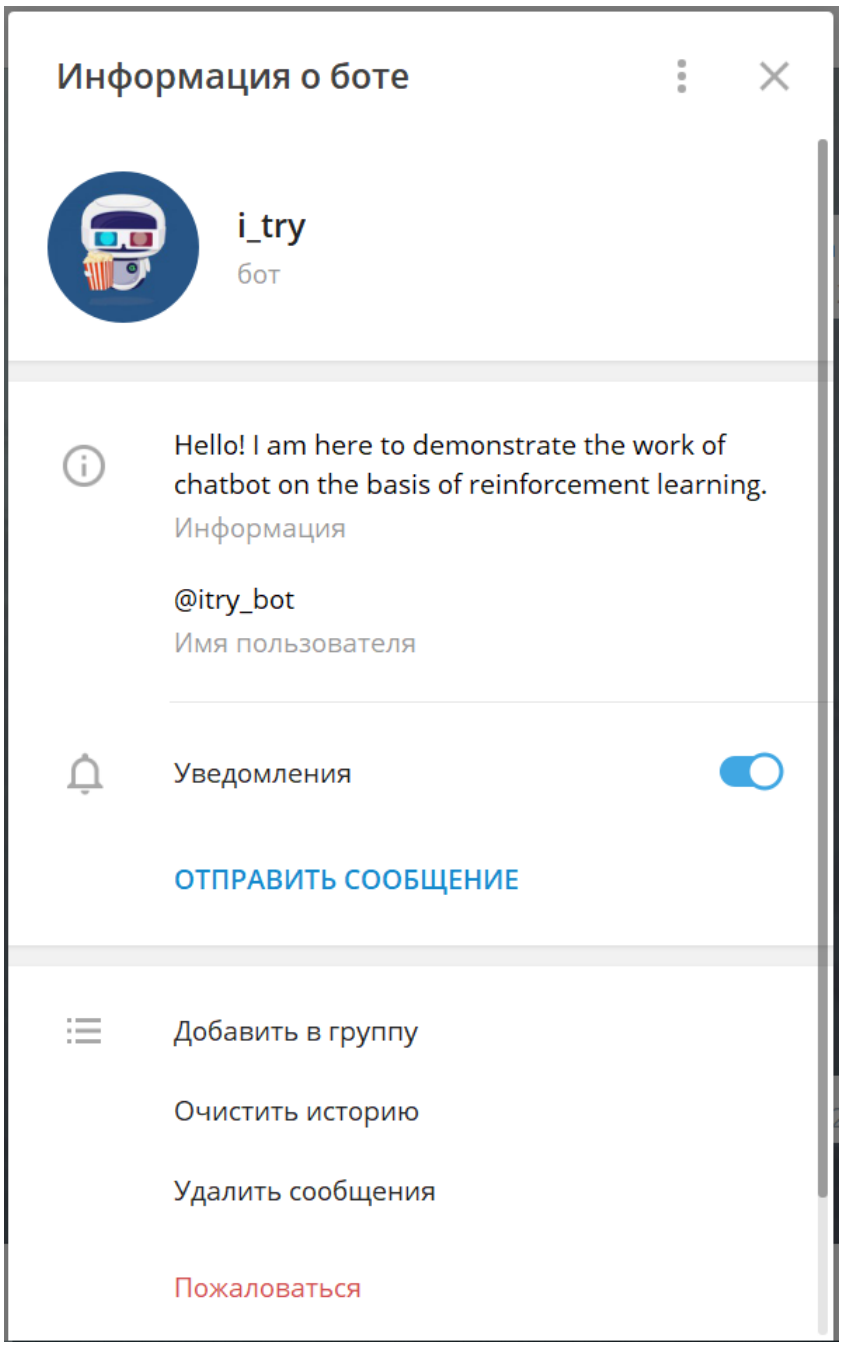


Рисунок 3.11 - Опис бота

На рисунках 3.12 та 3.13 зображено вдалий приклад діалогу та відповідно невдалий.

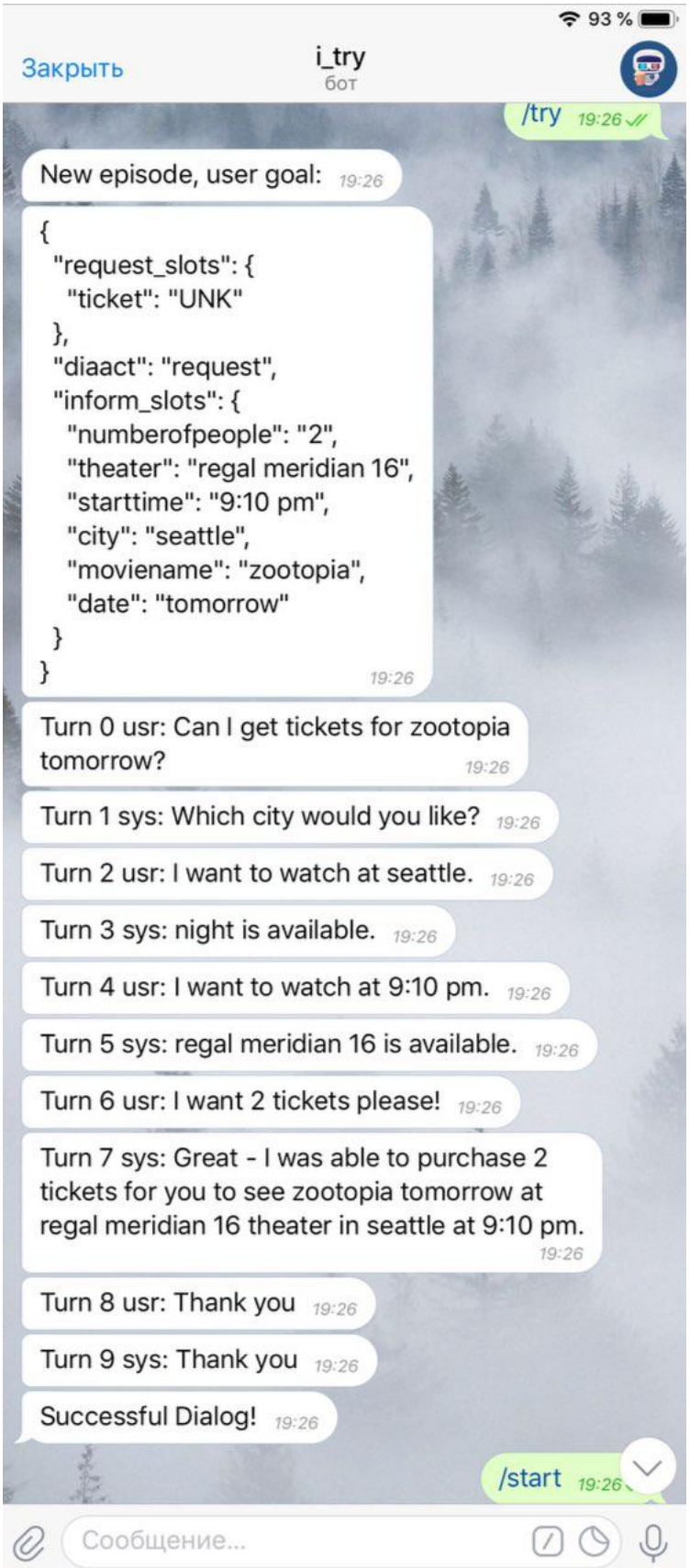


Рисунок 3.12 - Вдалий діалог

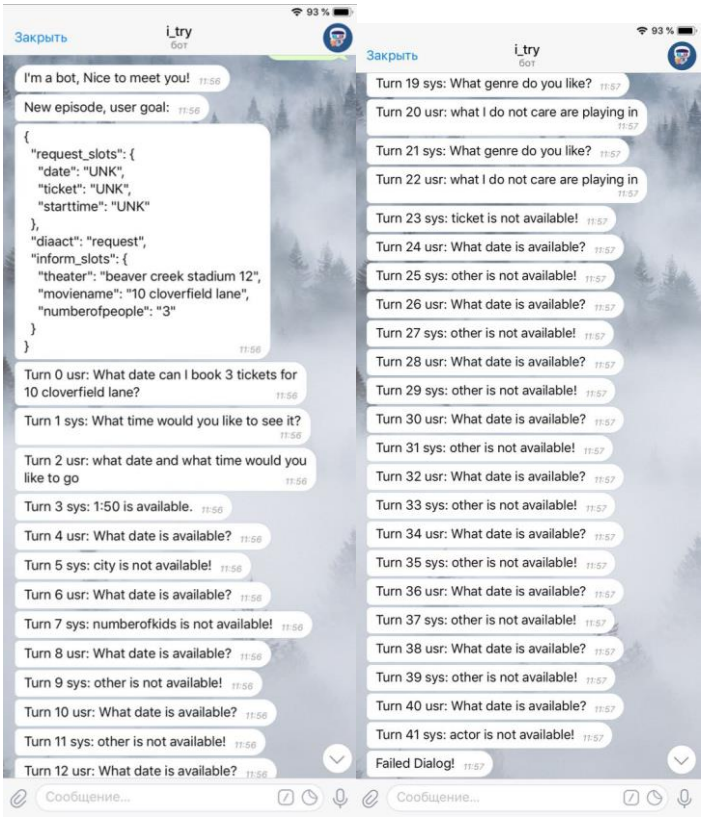


Рисунок 3.13 - Приклад невдалого діалогу

На рисунку 3.14 зображено приклад успішного діалогу в процесі навчання

```
Turn 0 usr: Can I get tickets for zootopia at regal meridian 16?
Turn 1 sys: What movie are you interested in?
Turn 2 usr: I want to watch zootopia.
Turn 3 sys: What time would you like to see it?
Turn 4 usr: I want to watch at 9:10 pm.
Turn 5 sys: Which city would you like?
Turn 6 usr: I want to watch at seattle.
Turn 7 sys: What date would you like to watch it?
Turn 8 usr: I want to set it up tomorrow
Turn 9 sys: Which theater would you like?
Turn 10 usr: I want to watch at regal meridian 16.
Turn 11 sys: How many tickets do you need?
Turn 12 usr: I want 2 tickets please!
Turn 13 sys: Great - I was able to purchase 2 tickets for you to see zootopia tomorrow at regal meridian 16 theater in seattle at 9:10 pm.
Turn 14 usr: Thank you
Turn 15 sys: Thank you
warm_start simulation episode 0: Success
```

Рисунок 3.14 – Приклад вдалого діалогу в процесі навчання

Під час навчання контролювались три метрики : успішність діалогу, кількість кроків для досягнення цілей, нагорода отримана агентом.

На рисунку 3.15 зображена графік для успішності діалогу без використання модулів NLU та NLG. На осі OX показується номер епохи навчання, на осі OУ відсоток успішних діалогів.

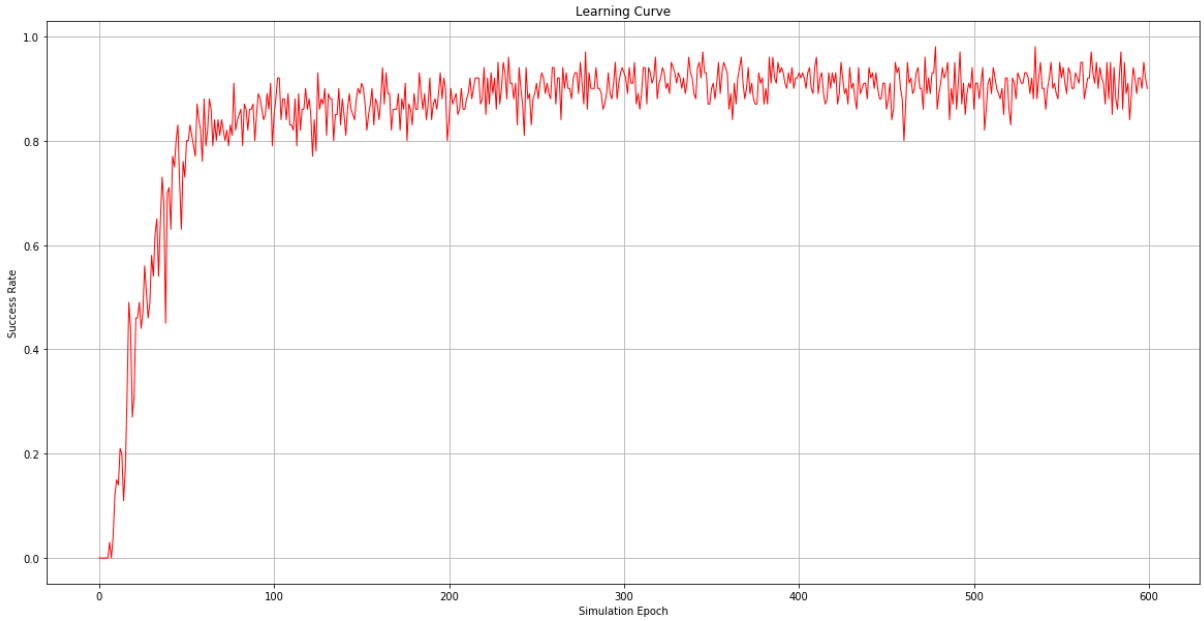


Рисунок 3.15 - Графік залежності успішності діалогу в залежності від епохи

На рисунку 3.16 зображена графік середньої кількості кроків, необхідних для виконання цілі користувача без використання модулів NLU та NLG. На осі OX показується номер епохи навчання, на осі OУ середня кількість кроків.

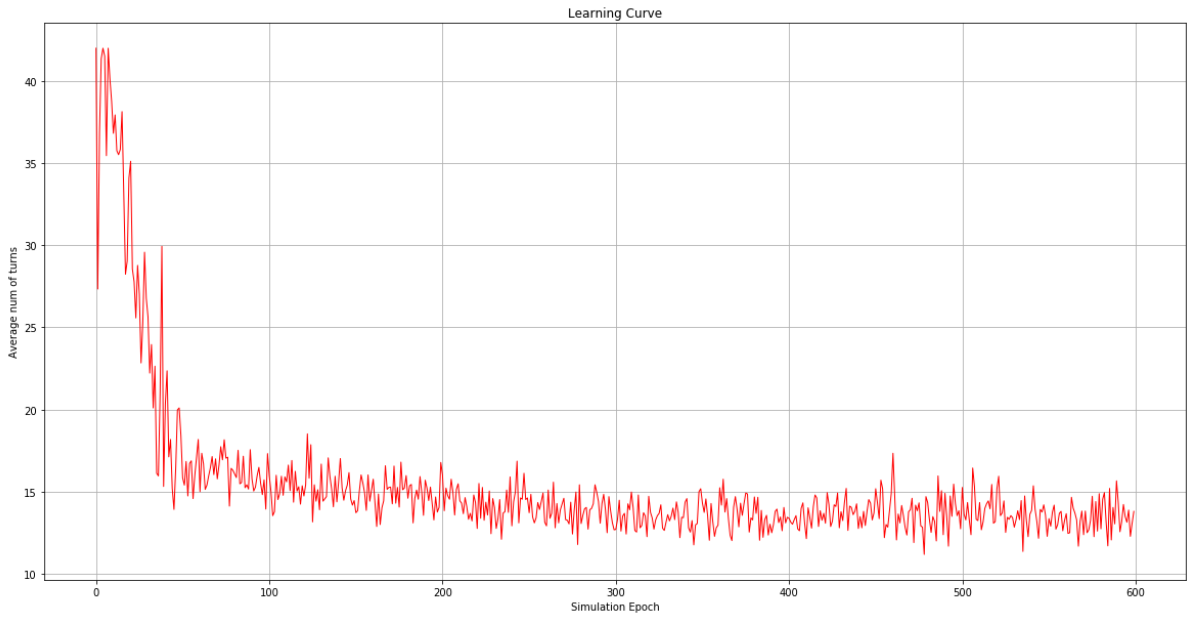


Рисунок 3.16 - Графік залежності необхідної кількості кроків у залежності від епохи

На рисунку 3.17 зображена графік середньої нагороди агента у процесі навчання без використання модулів NLU та NLG. На осі OX показується номер епохи навчання, на осі OУ середня кількість кроків.

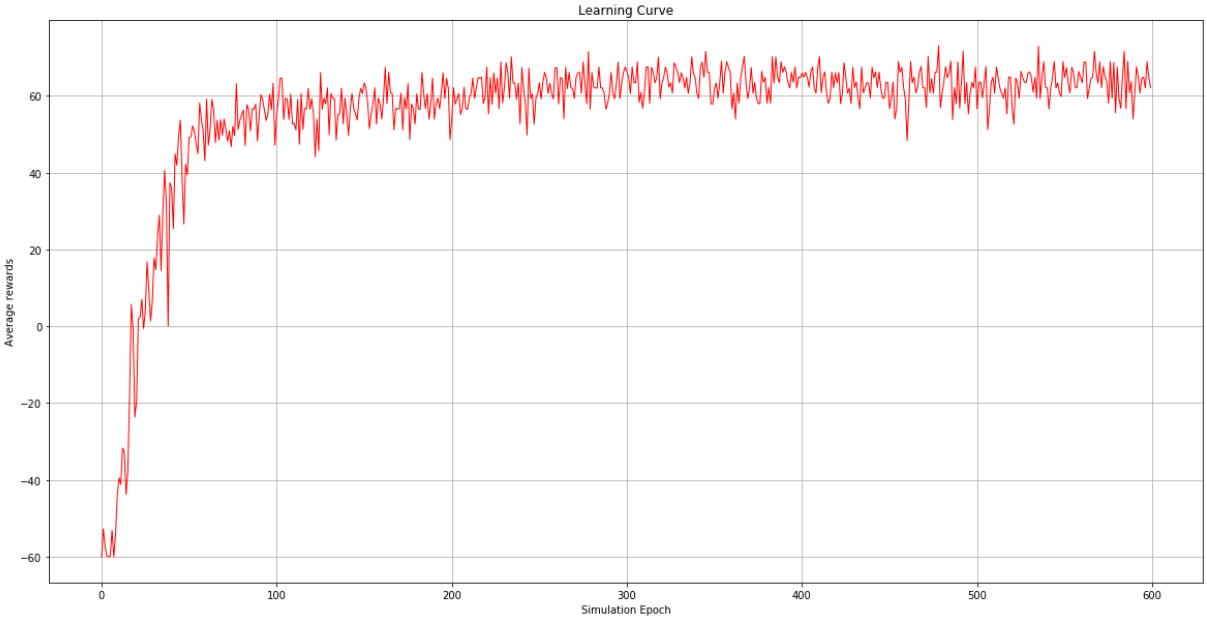


Рисунок 3.17 - Графік середньої винагороди в залежності від епохи

На рисунку 3.18 зображена графік для успішності діалогу з використанням модулів NLU та NLG. На осі OX показується номер епохи навчання, на осі OУ відсоток успішних діалогів.

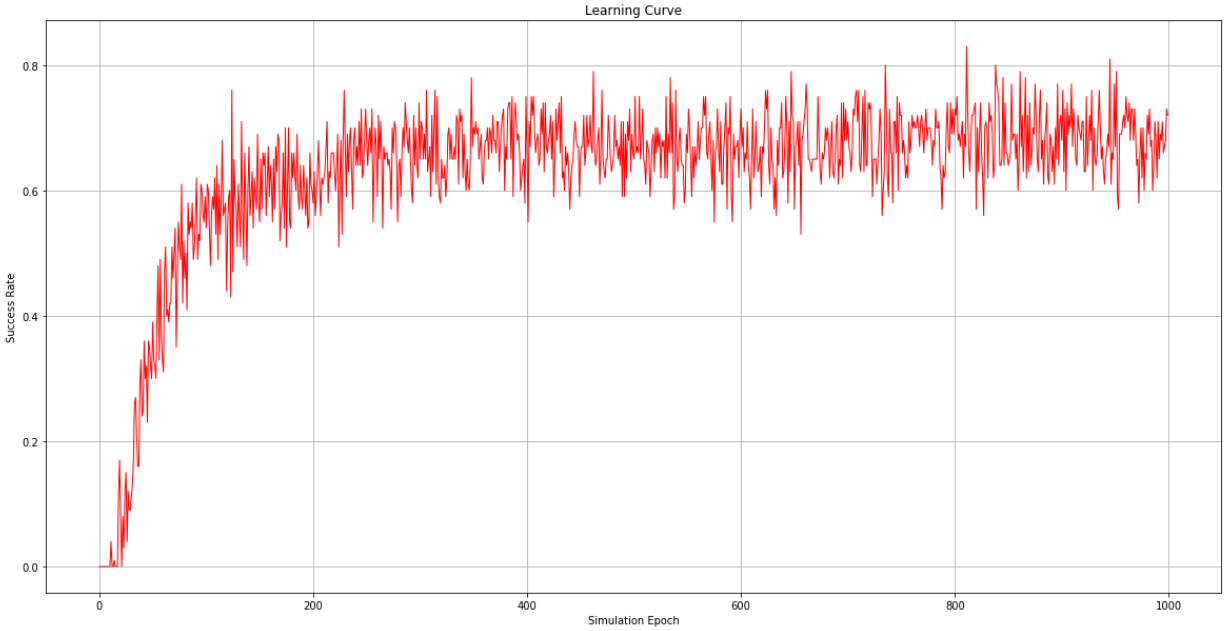


Рисунок 3.18 - Графік залежності успішності діалогу в залежності від епохи

На рисунку 3.19 зображена графік середньої кількості кроків, необхідних для виконання цілі користувача з використанням модулів NLU та NLG. На осі OX показується номер епохи навчання, на осі OУ середня кількість кроків.

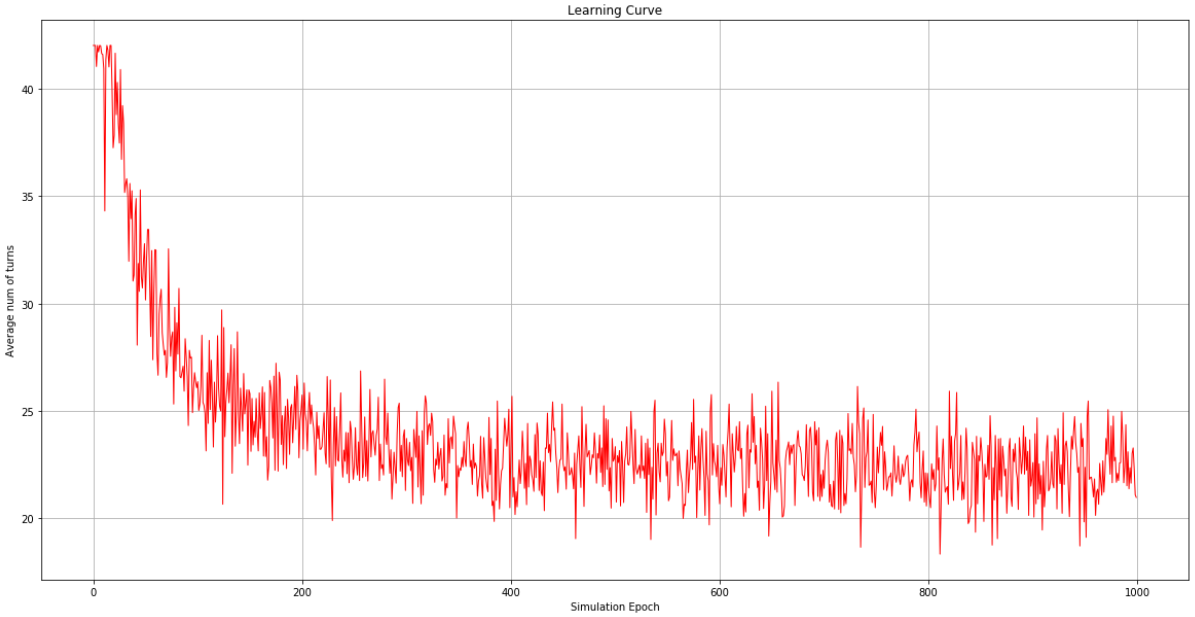


Рисунок 3.19 - Графік залежності необхідної кількості кроків у залежності від епохи

На рисунку 3.20 зображена графік середньої нагороди агента у процесі навчання з використанням модулів NLU та NLG. На осі OX показується номер епохи навчання, на осі OУ середня кількість кроків.

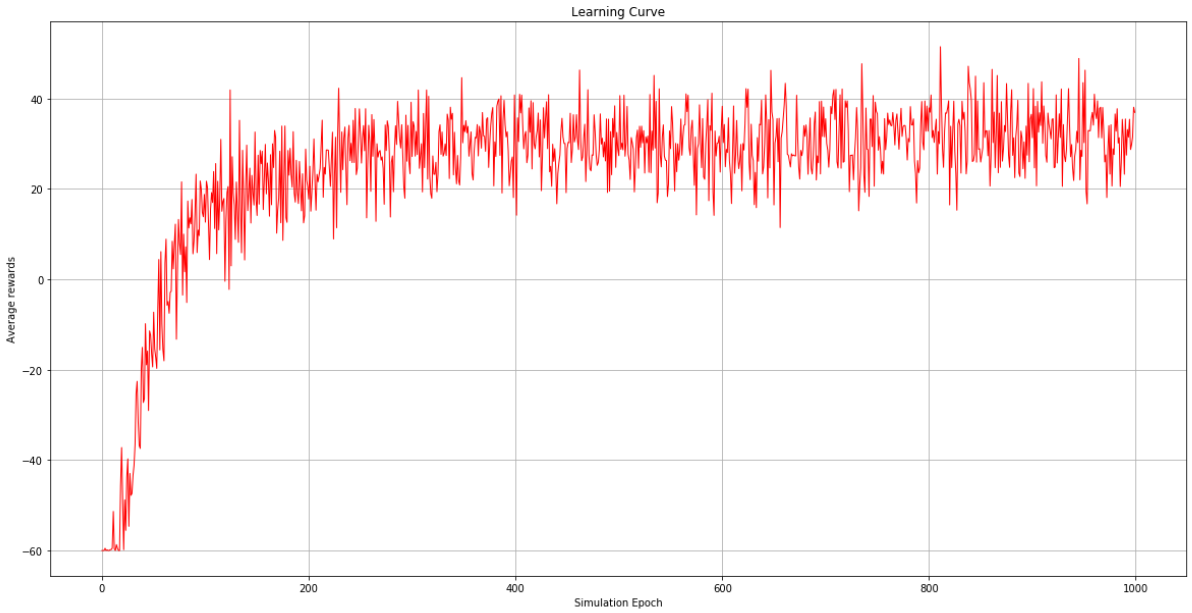


Рисунок 3.20 - Графік середньої винагороди в залежності від епохи

Як можна помітити з рисунків 3.15 - 3.20, значення по всіх метриках погіршуються при використанні модулів NLU та NLG. Для виконання цілей, що задані симулятором користувача, діалоговій системі необхідно близько 14 кроків в спрощеному режимі та близько 22 в повному, що є задовільним результатом. Система демонструє високий відсоток успішно закінчених сесій (90% у спрощеному та 75% у повному режимах). Можна відмітити, що значення нагороди значно корелює зі значеннями інших метрик.

Висновки

У цьому розділі були описані дані, що необхідні для використання чат-бота та внутрішні структури, які використовуються для обміну інформацією між різними модулями програми. А також наведені зразки цих структур та даних.

Також було проведено аналіз обраної архітектури з описом її модулів. Зокрема розглянуто взаємодію між діалоговою системою, в якій діє агент, та середовищем, яке представлене у вигляді симуляції користувача. Описано обмін інформацією у вигляді семантичних фреймів. У підрозділі 3.3.3 було послідовно описаний спрощений процес навчання агента в даній архітектурі.

Була розглянута UML діаграма побудованого чат-бота та проведено опис ключових класів та методів. Наведено результати роботи чат-бота у вигляді діалогу між діалоговою системою та симуляцією користувача. Надано опис з тим, як можна самостійно протестувати робота чат-бота в месенджері Telegram, а також проілюстровані графіки залежності значень обраних трьох метрик від епохи навчання.

РОЗДІЛ 4
ФУНКЦІОНАЛЬНО ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

Вступ

В даному розділі проводиться аналіз варіантів реалізації модулю з метою вибору оптимальної, з економічної точки зору. А саме проводиться функціонально-вартісний аналіз (ФВА).

Функціонально-вартісний аналіз — це метод комплексного техніко-економічного дослідження об’єкта з метою розвитку його корисних функцій при оптимальному співвідношенні між їхньою значимістю для споживача і витратами на їхнє здійснення. Є одним з основних методів оцінки вартості науково-дослідної роботи, оскільки ФВА враховує як технічну оцінку продукту, що розробляється, так і економічну частину розробки. Крім того, даний метод дозволяє вибрати оптимальний варіант розв’язання задачі, як з погляду розробника, так і з точки зору покупця. Також він дозволяє оптимізувати витрати й час виконання робіт.

Мета ФВА полягає у забезпеченні правильного розподілу ресурсів, виділених на виробництво продукції або надання послуг, на прямі та непрямі витрати. У даному випадку – аналізу функцій програмного продукту й виявлення усіх витрат на реалізацію цих функцій.

Фактично цей метод працює за таким алгоритмом:

1) визначається послідовність функцій, необхідних для виробництва продукту. Спочатку всі можливі, потім вони розподіляються по двом групам: ті, що впливають на вартість продукту і ті, що не впливають. На цьому ж етапі оптимізується сама послідовність скороченням кроків, що не впливають на цінність і відповідно витрат.

2) для кожної функції визначаються повні річні витрати й кількість робочих часів.

3) для кожної функції на основі оцінок попереднього пункту визначається кількісна характеристика джерел витрат.

4) після того, як для кожної функції будуть визначені їх джерела витрат, проводиться кінцевий розрахунок витрат на виробництво продукту.

4.1 Постановка задачі техніко-економічного аналізу

У роботі застосовується метод ФВА для проведення техніко-економічний аналізу розробки чатботу. Оскільки основні проектні рішення стосуються всієї системи, кожна окрема підсистема має їм задовольняти.

Відповідно цьому варто обирати і систему показників якості програмного продукту.

Технічні вимоги до продукту наступні:

1) програмний продукт повинен функціонувати на персональних комп’ютерах із стандартним набором компонент;

2) забезпечувати високу швидкість обробки великих об’ємів даних у реальному часі;

3) забезпечувати зручність і простоту взаємодії з користувачем або з розробником програмного забезпечення у випадку використовування його як модуля;

4) передбачати мінімальні витрати на впровадження програмного продукту.

4.2 Обґрунтування функцій програмного продукту

Головна функція F0– розробка програмного продукту, який аналізує

процес за вхідними даними та будує його модель для подальшого прогнозування.

Виходячи з конкретної мети, можна виділити наступні основні функції ПП:

1) F1 – вибір мови програмування;
2) F2 – Вибір фреймворка машинного навчання;
3) F3 – інтерфейс користувача.

Кожна з основних функцій може мати декілька варіантів реалізації.

Функція F1:

a) мова програмування Python
b) мова програмування Java;
c) мова програмування C#, .NET 4.5.

Функція F2:

a) фреймворк TensorFlow
b) фреймворк Bulrap.
c) фреймворк Accord.NET Framework.

Функція F3:

a) інтерфейс користувача,інтегрований в Telegram;
b) інтерфейс користувача, створений за технологією JavaFx;
c) інтерфейс користувача, створений за технологією .NET Framework.

4.3 Варіанти реалізації основних функцій

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рисунок 4.1). На основі цієї карти побудовано позитивно-негативну матрицю варіантів основних функцій (таблиця 4.1).

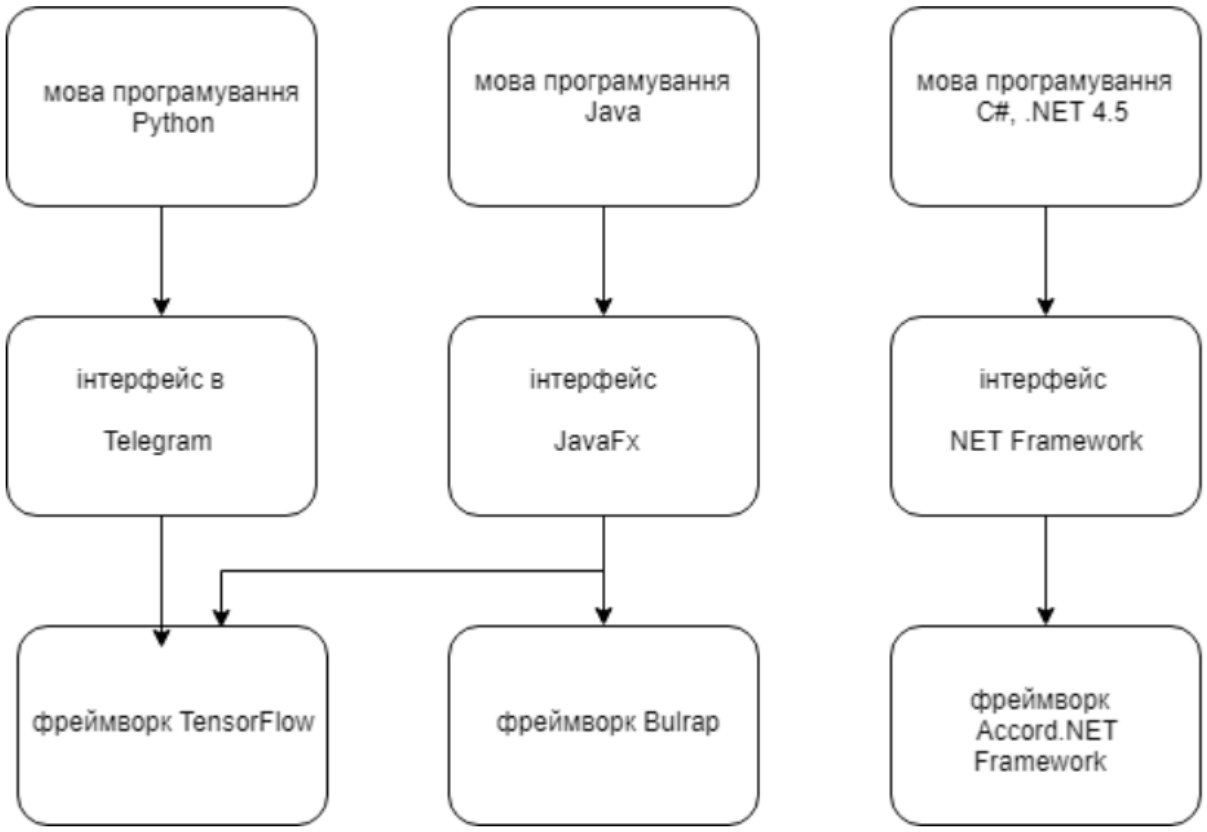


Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображує всі можливі комбінації варіантів реалізації функцій, які складають повну множину варіантів ПП.

Таблиця 4.1 – Позитивно-негативна матриця

Основні функції	**Варіанти реалізації**	**Переваги**	**Недоліки**
F1	*А*	Код швидко виконується, кросплатформений	Для запуску програми потрібно поставити чимало бібліотек
	Б	Простий запуск на різних платформах, швидкодія	Велика кількість коду
	В	Займає менше часу при написанні коду	Не кр осплатформений
F2	А	велика швидкодія, детальна документація, точність	Складна настройка

Продовження таблиці 4.1– Позитивно-негативна матриця

	Б	Зручна настройка, Простота запуску	Заплутаний у використанні,
	В	Простий у створенні.	Відсутність кросплатформеності, мало документації
F3	*А*	Дуже легкий у створенні, велика швидкодія	Необхідний сервер
	Б	Стабільний у використанні	Мала швидкодія
	В	Швидкий, легкий у створенні	Не кросплатформенний

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція F1:

Оскільки для задачі потрібна швидкодія і кросплатформеність вибираємо варіанти а) і б).

Функція F2:

Оскільки нам потрібна велика швидкодія а також точність беремо варіанта).

Функція F3:

Інтерфейс користувача не відіграє велику роль у даному програмному продукту, тому вважаємо варіанти а) та б) гідними розгляду. Варіант в) відкинутий через вибір мов програмування.

Таким чином, будемо розглядати такі варіанти реалізації ПП:

1) F1а – F2а – F3а
2) F1б – F2б – F3а

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.4 Обґрунтування системи параметрів ПП

4.4.1 Опис параметрів

На підставі даних про основні функції, що повинен реалізувати програмний продукт, вимог до нього, визначаються основні параметри виробу, що будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

1) *X1* – Об’єм Використаної пам’яті;
2) *X2* – Загруженість процесора;
3) *X3* – Швидкість відклику інтерфейсу;
4) *X4* – Час обробки даних.

X1: Відображає об’єм пам’яті в оперативній пам’яті персонального комп’ютера, необхідний для збереження та обробки даних під час виконання програми.

X2: Відображає Загруженість процесора під час виконання програми.

X3: Відображає швидкість відклику інтерфейса

X4: Показує Час обробки даних обраним фреймворком

4.4.2Кількісна оцінка параметрів

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію ПП як показано у табл. 4.2

Таблиця 4.2 – Основні параметри ПП

Назва Параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Об’єм пам’яті для збереження даних	X1	Мб	128	64	16
Загруженість процесора	X2	%	100	40	5
Швидкість відклику інтерфейсу	X3	мс	1000	300	20
Час обробки даних	X4	с	2000	1000	400

За даними таблиці 4.2 будуються графічні характеристики параметрів – (рисунок 4.2 – риунок. 4.5).

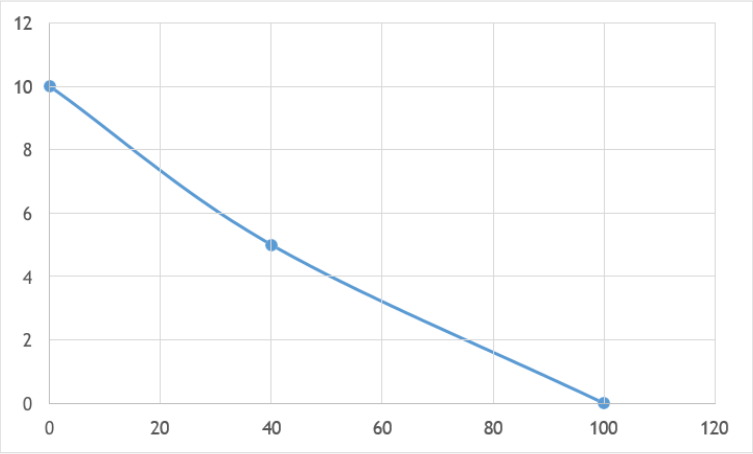


Рисунок 4.3 – X2, Загруженість процессора

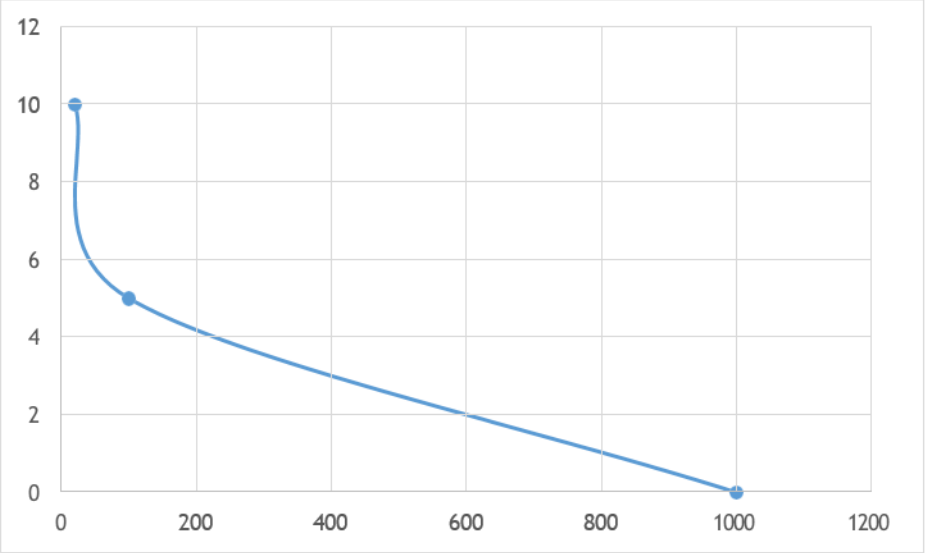


Рисунок 4.4 – X3, Швидкість відклику інтерфейсу

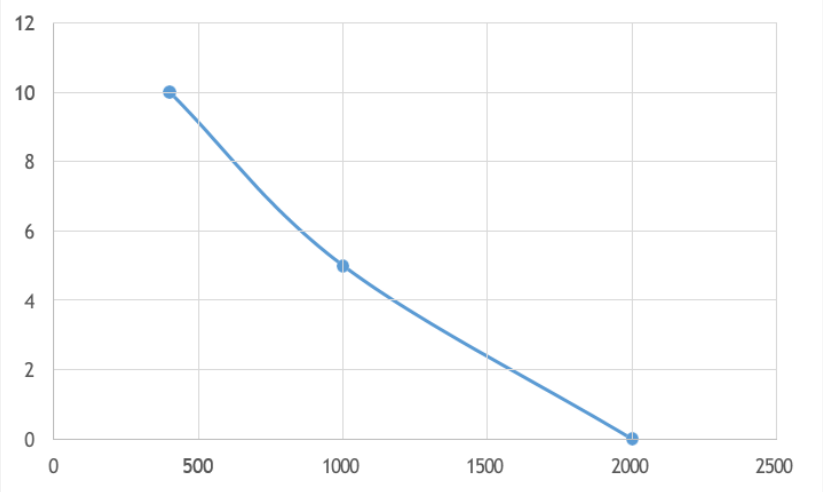


Рисунок 4.5 – X4, Час обробки даних

4.4.2 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка

програмного продукту, який дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

1) визначення рівня значимості параметра шляхом присвоєння різних рангів;

2) перевірку придатності експертних оцінок для подальшого використання;

3) визначення оцінки попарного пріоритету параметрів;

4) обробку результатів та визначення коефіцієнту значимості. Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 – Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Об’єм пам’яті для збереження даних	Мб	1	2	1	1	1	1	2	9	-8,5	72.25
X2	Загруженість процесора	%	2	1	2	2	2	3	1	13	-4,5	20.25
X3	Швидкість відклику інтерфейсу	мс	4	3	3	4	4	4	3	25	7,5	56.25
X4	Час обробки даних	с	3	4	4	3	3	2	4	23	5.5	30.25
	Разом		10	10	10	10	10	10	10	70	0	179

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70, \quad (4.1)$$

де N – число експертів, n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17.5 \quad (4.2)$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T \quad (4.3)$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення

$$S = \sum_{i=1}^{N} \Delta_i^2 = 179 \quad (4.4)$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3-n)} = \frac{12 \cdot 179}{7^2(4^3-4)} = 0.72 > W_{\text{норм}} = 0{,}67 \quad (4.5)$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.4.

Таблиця 4.4 – Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	<	>	<	<	<	<	>	<	0,5
X1 і X3	<	<	<	<	<	<	<	<	0,5
X1 і X4	<	<	<	<	<	<	<	<	0,5
X2 і X3	<	<	<	<	<	<	<	<	0,5
X2 і X4	<	<	<	<	<	>	<	<	0,5
X3 і X4	>	<	<	>	>	>	<	>	1,5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1.5, \text{при Xi} > \text{Xj} \\ 1.0, \text{при Xi} = \text{Xj} \\ 0.5, \text{при Xi} < \text{Xj} \end{cases} \quad (4.6)$$

З отриманих числових оцінок переваги складемо матрицю A=$\| a_{ij} \|$.Для кожного параметра зробимо розрахунок вагомості $K_{ві}$ за наступними формулами:

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i}, \quad (4.7)$$

де $b_i = \sum_{i=1}^{N} a_{ij}$.

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%).

На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \text{де } b_i' = \sum_{i=1}^{N} a_{ij} b_j. \quad (4.8)$$

Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 – Розрахунок вагомості параметрів

Параметри x_i	Параметри x_i				Перша ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$	b_i^2	$K_{\text{ві}}^2$
X1	1,0	0,5	0,5	0,5	2,5	0,156	9,25	0,156	34,125	0,157
X2	1,5	1,0	0,5	0,5	3,5	0,218	12,25	0,207	44,875	0,207
X3	1,5	1,5	1,0	1,5	5,5	0,343	21,25	0,360	77,875	0,360
X4	1,5	1,5	0,5	1,0	4,5	0,281	16,25	0,275	59,125	0,273
Всього:					16	1	98	1	1	59

4.4 Аналіз рівня якості варіантів реалізації функції

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів *X2* (Загруженість процесора) та *X1*(Об'єм пам'яті для збереження даних) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра *X3 (*Швидкість відклику інтерфейсу) обрано не найгіршим (не максимальним).

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо. Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^{n} K_{\text{в}i,j} B_{i,j}, \tag{4.9}$$

де n – кількість параметрів;

$K_{\text{в}i}$– коефіцієнт вагомості i–го параметра;

B_i – оцінка i–го параметра в балах.

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Параметри, що беруть участь у реалізації функцій	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1(X1, X2)	А	X1	50	6,7	0,157	1,0519
		X2	30	6,9	0,207	1,4283
	Б	X1	90	3,2	0,157	0,5024
		X2	50	4,8	0,207	0,9936
F2(X3)	А	X3	200	6,4	0,360	2,304
	Б		190	6,5	0,360	2,34
F3(X4)	А	X4	1200	4,6	0,360	1,656

За даними з таблиці 4.6 за формулою

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}], \quad (4.10)$$

визначаємо рівень якості кожного з варіантів:

КК1 = 1,0519 + 1,4283 + 2,304 + 1,656 = 6,4402

КК2 = 0,5024 + 0,9936 + 2,34 + 1,656 = 5,492

Як видно з розрахунків, кращим є перший варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.5 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1) Розробка архітектури класів та їх логіки;
2) Реалізація графічного інтерфейсу
3) Тестування програмного функціоналу;

Варіант І також має завдання - вивчення фреймворка TensorFlow

Варіант ІІ також має завдання вивчення фреймворка Bulrap

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань. Загальна трудомісткість обчислюється як

$$T_O = T_P \cdot K_П \cdot K_{СК} \cdot K_M \cdot K_{СТ} \cdot K_{СТ.М}, \quad (4.11)$$

де T_P – трудомісткість розробки ПП;

$K_П$ – поправочний коефіцієнт;

$K_{СК}$ – коефіцієнт на складність вхідної інформації;

K_M – коефіцієнт рівня мови програмування;

$K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 2, трудомісткість дорівнює: T_P =36 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $K_П$ = 1.51.

Поправочний коефіцієнт, який враховує складність контролю вхідної та

вихідної інформації для всіх семи завдань рівний 1: $К_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ}$ = 0.8. Тоді, за формулою 4.1, загальна трудомісткість програмування першого завдання дорівнює:

$$Т_1 = 36 \cdot 1.51 \cdot 0.8 = 43.49 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто

$Т_Р$ =19 людино-днів, $К_П$ =0.9,$К_{СК}$ = 1,$К_{СТ}$=0.8:

$$Т_2 = 19 \cdot 0.8 \cdot 0.9 = 13.68 \text{ людино-днів.}$$

Для третього завдання (використовується алгоритм третьої групи складності, степінь новизни А), тобто $Т_Р$ =27 людино-днів, $К_П$ =1.26 ,$К_{СК}$ = 1,$К_{СТ}$=0.8:

$$Т_3 = 27 \cdot 0.8 \cdot 1.26 = 27.22 \text{ людино-днів.}$$

Для четвертого завдання (використовується алгоритм третьої групи складності, степінь новизни А), тобто $Т_Р$ =27 людино-днів, $К_П$ =1.26,$К_{СК}$ = 1, $К_{СТ}$=1:

$$Т_5 = 27 \cdot 1.51 = 34.02 \text{ людино-днів.}$$

Для п’ятого завдання (використовується алгоритм першої групи складності, степінь новизни Б), тобто $Т_Р$ =64 людино-днів, $К_П$ =1,021,$К_{СК}$ = 1, $К_{СТ}$ =1:$Т_4$ = 64 · 1.021= 65.344 людино-днів.

Складаємо трудомісткість відповідних завдань для кожного з обраних

варіантів реалізації програми, щоб отримати їх трудомісткість:

$$T_I = (43.49 + 13.68 + 27.22 + 34.02) \cdot 8 = 947.28 \text{ людино-годин};$$

$$T_{II} = (43.49 + 13.68 + 27.22 + 65.344) \cdot 8 = 1197.87 \text{ людино-годин};$$

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 9000 грн., один аналітик з окладом 6000 грн . Визначимо зарплату за годину за формулою:

$$\text{СЧ} = \frac{\text{М}}{T_m \cdot t} \text{ грн.}, \quad (4.12)$$

де М – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

$$\text{СЧ} = \frac{9000 + 9000 + 6000}{3 \cdot 21 \cdot 8} = 47.6 \text{ грн.}$$

Тоді, розрахуємо заробітну плату за формулою

$$\text{СЗП} = \text{С}_\text{ч} \cdot \text{Т}_i \cdot \text{КД}, \quad (4.13)$$

де $\text{С}_\text{ч}$ – величина погодинної оплати праці програміста;

Т_i – трудомісткість відповідного завдання;

$\text{К}_\text{Д}$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $\text{С}_\text{ЗП} = 47.6 \cdot 947.28 \cdot 1.2 = 54108.63$ грн.

II. $\text{С}_\text{ЗП} = 47.6 \cdot 1197.87 \cdot 1.2 = 68422.33$ грн.

Відрахування на всі види соціального страхування становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 54108.63 * 0.22 = 11903.9$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 68422.33 * 0.22 = 15052.91$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 9000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 9000 \cdot 0,2 = 21600 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г \cdot (1 + К_З) = 21600 \cdot (1 + 0.2) = 25920 \text{ грн.}$$

Відрахування на єдиний соціальний внесок:

$$С_{ВІД} = С_{ЗП} \cdot 0.22 = 25920 * 0,22 = 5702.4 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 25000 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.15 \cdot 0.25 \cdot 25000 = 7187.5 \text{ грн.},$$

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$– річна норма амортизації;

$Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.15 \cdot 25000 \cdot 0.05 = 1437.5$ грн.,

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 8 – 16) \cdot 8 \cdot 0.9 = 1706.4$$

де $Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t –кількість робочих годин в день;

$К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1706{,}4 \cdot 0{,}75 \cdot 2{,}7515 = 3521.37 \text{грн.},$$

де $N_С$ – середньо-споживча потужність приладу;

$К_З$– коефіцієнтом зайнятості приладу;

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0.67 = 25000 \cdot 0{,}67 = 16750 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$СЕКС = СЗП + СВІД + СА + СР + СЕЛ + СН$$

$$С_{ЕКС} = 25920 + 5702.4 + 7187.5 + 1437.5 + 3521.37 + 16750 = 60518.77 \text{грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М\text{-}Г} = С_{ЕКС}/ Т_{ЕФ} = 60518.77/1706.4 = 35.46 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$С_М = С_{М\text{-}Г} \cdot Т$$

I. $С_М = 35.46 * 947.28 = 33590.55$грн.;

II. $С_М = 35.46 * 1197.87 = 42476.47$грн.;

Накладні витрати складають 67% від заробітної плати:

$$С_Н = С_{ЗП} \cdot 0{,}67$$

I. $С_Н = 54108.63* 0{,}67 = 36252.78$ грн.;

II. $С_Н = 68422.33 * 0{,}67 = 45842.96$ грн.;

Отже, вартість розробки ПП за варіантами становить:

$$СПП = СЗП+ СВІД+ СМ +СН$$

I. $С_{ПП} = 54108.63+ 11903.9+ 33590.55+ 36252.78 = 135855.86$ грн.;

II. $С_{ПП} = 68422.33 + 15052.91 + 42476.47+ 45842.96 = 171794.67$

4.6 Вибір кращого варіанта ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$КТЕРj = ККj/СФj,$$

$$К_{ТЕР1} = 6,4402 / 135855.86 = 0,47 \cdot 10^{-4};$$

$$К_{ТЕР2} = 5,492 / 171794.67 = 0.32 \cdot 10^{-4};$$

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1} = 0.47 \cdot 10^{-4}$.

Висновки

В даному розділі проведено повний функціонально-вартісний аналіз ПП, який було розроблено в рамках дипломного проекту. Процес аналізу можна умовно розділити на дві частини.

В першій з них проведено дослідження ПП з технічної точки зору: було визначено основні функції ПП та сформовано множину варіантів їх реалізації; на основі обчислених значень параметрів, а також експертних оцінок їх важливості було обчислено коефіцієнт технічного рівня, який і дав змогу визначити оптимальну з технічної точки зору альтернативу реалізації функцій ПП.

Другу частину ФВА присвячено вибору із альтернативних варіантів реалізації найбільш економічно обґрунтованого. Порівняння запропонованих варіантів реалізації в рамках даної частини виконувалось за коефіцієнтом ефективності, для обчислення якого були обчислені такі допоміжні параметри, як трудомісткість, витрати на заробітну плату, накладні витрати.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмногокомплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня

якості $К_{ТЕР} = 0,47 \cdot 10^{-4}$.

Цей варіант реалізації програмного продукту має такі параметри:

a) мова програмування – Python;

b) фреймворк TensorFlow

c) інтерфейс користувача, інтегрований в Telegram.

Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, непоганий функціонал і швидкодію.

ВИСНОВКИ

Попри очевидно зручну форму використання, чат-боти довгий час залишалися в ніші дослідницьких та розважальних програмних продуктів, яскравим прикладом яким можуть бути системи ELIZA(1966) та MITSUKU(з 2005). Однак стрімкий розвиток NLU технологій та глибокого навчання дозволив чат-ботам знайти широку область для застосування, зокрема в автоматизації сфери обслуговування та стати актуальною предметною областю.

Варіативність підходів до створення чат-ботів та їхня інтуїтивно зрозуміла концепція взаємодії користувача з програмою у майбутньому дозволить стати ще більш популярною цю область, а можливо перетворить її в головний інтерфейс для програмного забезпечення майбутнього. Уже доступні якісні приклади чат-ботів від провідних компаній світу, такі як : Siri від Apple та Alexa від Amazon, а також рішення для зручного створення користувацьких чат-ботів, таких як Luis від Microsft.

Однією з можливостей подальшого вдосконалення чат-ботів є викостання навчання з підкріпленням, яке останній час почало стрімко розвиватись та популяризуватись. Воно надає можливість навчати моделі в більш природній та еволюційно правильний спосіб ніж традиційні методи, що застосовувались до задач машинного навчання. Саме навчання з підкріпленням дозволяє навчати всі модулі в архітекторах чат-ботів одночасно, уникаючи накопичення помилок. Діалогові системи створені за допомогою навчання з підкріпленням показують кращу стійкість до помилок, гнучкість та відтворюваність.

У даній роботі побудувано працюючий варіант чат-бота на основі навчання з підкріпленням, який демонструє роботу системи замовлення квитків в кінотеатрах. Також нам вдалося побудувати симулятора користувача, який відіграв роль середовища для агента який здатний

завершувати 89% відсотків діалогів успіхів. Досліджено, проаналізовано і побудовано обраний тип та архітектуру чат-бота, пояснено його роботу та внутрішню структуру.

Список використаних джерел

1. CS234 : Reinforcement Learning. URL : http://web.stanford.edu/ class/ cs234/index.html

2. Саттон Р.С, Э. Г. Барто Обучение с подкреплением. Москва : Лаборатория знаний 2011. 399с.

3. Maxim Lapan Deep Reinforcement Learning Hands-On. Birmingham : Packt Publishing Ltd. 2018. 549p.

4. Jianfeng Gao, Michel Galley, Lihong Li Neural Approaches to Conversational AI. URL : https://arxiv.org/abs/1809.08267.

5. Jaromír Janisch Let’s make a DQN : Full DQN . URL: https://jaromiru.com/2016/10/21/lets-make-a-dqn-full-dqn.

6. Ketakee Nimavat, Tushar Champaneria Chatbots: An overview. Types, Architecture, Tools and Future Possibilities. Url : https://www.researchgate.net/ publication/320307269_Chatbots_An_overview_Types_Architecture_Tools_and_ Future_Possibilities

7. Dipanjan Sarkar A Comprehensive Hands-on Guide to Transfer Learning with Real-World Applications in DeepLearning. URL : https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a.

8. Xiujun Li, Yun-Nung Chen, Lihong Li, Jianfeng Gao, Asli Celikyilmaz End-to-End Task-Completion Neural Dialogue Systems. URL: https://arxiv.org/abs/1703.01008.

9. Xiujun Li, Zachary C. Lipton, Bhuwan Dhingra, Lihong Li, Jianfeng Gao, Yun-Nung Chen A User Simulator for Task-Completion Dialogues. URL: https://arxiv.org/abs/1612.05688.

10. Mastering Chess and Shogi by Self-Play with a GeneralReinforcementLearning Algorithm, David Silver, Thomas Hubert, Julian Schrittwieser, and others. URL : https://arxiv.org/abs/1712.01815.

11. VC Ramesh Unsupervised Deep Learning for Dialog Chatbots. URL : https://www.linkedin.com/pulse/unsupervised-deep-learning-dialog-chatbots-vc-ramesh/

12. G. Neff, P. Nagy Automation, Algorithms, and Politics| Talking to Bots: Symbiotic Agency and the Case of Tay. URL : https://ijoc.org/index.php/ijoc/article/view/6277/1804

13. S. A. Abdul-Kader, J. Woods, Survey on Chatbot Design Techniques in Speech Conversation Systems. URL : https://thesai.org/Downloads/ Volume6No7/Paper_12-Survey_on_Chatbot_Design_Techniques_in_Speech_ Conversation_ Systems.pdf

14. Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. URL : http://www-anw.cs.umass.edu/~barto/ courses/cs687/williams92simple.pdf

15. Kotti, M., Diakoloukas, V., Papangelis, A., Lagoudakis, M., Stylianou, Y. A case study on the importance of belief state representation for dialogue policy management. URL : https://www.isca-speech.org/archive/Interspeech_2018/pdfs/ 1293.pdf

16. Roy, N., Pineau, J., and Thrun, S. (2000). Spoken dialogue management using probabilistic reasoning. URL : https://dl.acm.org/citation.cfm?id=1075231

17. Young, S. J., Gašić, M., Keizer, S., Mairesse, F., Schatzmann, J., Thomson, B., and Yu, K. The Hidden Information State model: A practical framework for POMDP-based spoken dialogue management. URL : https://hal.archives-ouvertes.fr/hal-00598186/document

18. Hori, C., Hori, T., Watanabe, S., and Hershey, J. R. Context sensitive spoken language understanding using role dependent LSTM layers. URL : https://www.semanticscholar.org/paper/Context-Sensitive-and-Role-Dependent-Spoken-Using-Hori-Hori/dd3ba828dbbb17cf478f6840a37954f6ebc81770

19. Chen, Y.-N., Hakkani-Tür, D., Tur, G., Gao, J., and Deng, L. End-to-end memory networks with knowledge carryover for multi-turn spoken language

understanding. URL : https://www.microsoft.com/en-us/research/publication/contextualslu

ДОДАТОК А
Код програмного продукту

```
class DialogManager:
    """ A dialog manager to mediate the interaction between an agent and a
customer """

    def __init__(self, agent, user, act_set, slot_set, movie_dictionary):
        self.agent = agent
        self.user = user
        self.act_set = act_set
        self.slot_set = slot_set
        self.state_tracker = StateTracker(act_set, slot_set,
movie_dictionary)
        self.user_action = None
        self.reward = 0
        self.episode_over = False

    def initialize_episode(self):
        """ Refresh state for new dialog """

        self.reward = 0
        self.episode_over = False
        self.state_tracker.initialize_episode()
        self.user_action = self.user.initialize_episode()

        self.state_tracker.update(user_action = self.user_action)
        #print self.user_action
        if dialog_config.run_mode < 3:
            print ("New episode, user goal:")
            print json.dumps(self.user.goal, indent=2)
        self.print_function(user_action = self.user_action)

        self.agent.initialize_episode()

    def next_turn(self, record_training_data=True):
        """ This function initiates each subsequent exchange between agent
and user (agent first) """


########################################################################
        #   CALL AGENT TO TAKE HER TURN

########################################################################
        self.state = self.state_tracker.get_state_for_agent()
        self.agent_action = self.agent.state_to_action(self.state)


########################################################################
        #   Register AGENT action with the state_tracker

########################################################################
        self.state_tracker.update(agent_action=self.agent_action)

        self.agent.add_nl_to_action(self.agent_action) # add NL to Agent
Dia_Act
        self.print_function(agent_action =
```

```
self.agent_action['act_slot_response'])
        self.reward = self.state_tracker.reward_function()

################################################################################
        #   CALL USER TO TAKE HER TURN

################################################################################
        self.sys_action = self.state_tracker.dialog_history_dictionaries()[-
1]
        self.user_action, self.episode_over, dialog_status =
self.user.next(self.sys_action)
        #self.reward = self.reward_function(dialog_status)


################################################################################
        #   Update state tracker with latest user action

################################################################################
        if self.episode_over != True:
            self.state_tracker.update(user_action = self.user_action)
            self.print_function(user_action = self.user_action)


################################################################################
        #  Inform agent of the outcome for this timestep (s_t, a_t, r,
s_{t+1}, episode_over)

################################################################################
        if record_training_data:
            self.agent.register_experience_replay_tuple(self.state,
self.agent_action, self.reward, self.state_tracker.get_state_for_agent(),
self.episode_over)

        return (self.episode_over, self.reward)


    def reward_function(self, dialog_status):
        """ Reward Function 1: a reward function based on the dialog_status
"""
        if dialog_status == dialog_config.FAILED_DIALOG:
            reward = -self.user.max_turn #10
        elif dialog_status == dialog_config.SUCCESS_DIALOG:
            reward = 2*self.user.max_turn #20
        else:
            reward = -1
        return reward

    def reward_function_without_penalty(self, dialog_status):
        """ Reward Function 2: a reward function without penalty on per turn
and failure dialog """
        if dialog_status == dialog_config.FAILED_DIALOG:
            reward = 0
        elif dialog_status == dialog_config.SUCCESS_DIALOG:
            reward = 2*self.user.max_turn
        else:
            reward = 0
        return reward


    def print_function(self, agent_action=None, user_action=None):
        """ Print Function """

        if agent_action:
            if dialog_config.run_mode == 0:
```

```
                if self.agent.__class__.__name__ != 'AgentCmd':
                    print ("Turn %d sys: %s" % (agent_action['turn'],
agent_action['nl']))
            elif dialog_config.run_mode == 1:
                if self.agent.__class__.__name__ != 'AgentCmd':
                    print("Turn %d sys: %s, inform_slots: %s, request slots:
%s" % (agent_action['turn'], agent_action['diaact'],
agent_action['inform_slots'], agent_action['request_slots']))
            elif dialog_config.run_mode == 2: # debug mode
                print("Turn %d sys: %s, inform_slots: %s, request slots: %s"
% (agent_action['turn'], agent_action['diaact'],
agent_action['inform_slots'], agent_action['request_slots']))
                print ("Turn %d sys: %s" % (agent_action['turn'],
agent_action['nl']))

            if dialog_config.auto_suggest == 1:
                print('(Suggested Values: %s)' %
(self.state_tracker.get_suggest_slots_values(agent_action['request_slots'])))
        elif user_action:
            if dialog_config.run_mode == 0:
                print ("Turn %d usr: %s" % (user_action['turn'],
user_action['nl']))
            elif dialog_config.run_mode == 1:
                print ("Turn %s usr: %s, inform_slots: %s, request_slots: %s"
% (user_action['turn'], user_action['diaact'], user_action['inform_slots'],
user_action['request_slots']))
            elif dialog_config.run_mode == 2: # debug mode, show both
                print ("Turn %d usr: %s, inform_slots: %s, request_slots: %s"
% (user_action['turn'], user_action['diaact'], user_action['inform_slots'],
user_action['request_slots']))
                print ("Turn %d usr: %s" % (user_action['turn'],
user_action['nl']))

            if self.agent.__class__.__name__ == 'AgentCmd': # command line
agent
                user_request_slots = user_action['request_slots']
                if 'ticket'in user_request_slots.keys(): del
user_request_slots['ticket']
                if len(user_request_slots) > 0:
                    possible_values =
self.state_tracker.get_suggest_slots_values(user_action['request_slots'])
                    for slot in possible_values.keys():
                        if len(possible_values[slot]) > 0:
                            print('(Suggested Values: %s: %s)' % (slot,
possible_values[slot]))
                        elif len(possible_values[slot]) == 0:
                            print('(Suggested Values: there is no available
%s)' % (slot))
                else:
                    kb_results = self.state_tracker.get_current_kb_results()
                    print ('(Number of movies in KB satisfying current
constraints: %s)' % len(kb_results))
```

```
class AgentDQN(Agent):
    def __init__(self, movie_dict=None, act_set=None, slot_set=None,
params=None):
        self.movie_dict = movie_dict
```

```
        self.act_set = act_set
        self.slot_set = slot_set
        self.act_cardinality = len(act_set.keys())
        self.slot_cardinality = len(slot_set.keys())

        self.feasible_actions = dialog_config.feasible_actions
        self.num_actions = len(self.feasible_actions)

        self.epsilon = params['epsilon']
        self.agent_run_mode = params['agent_run_mode']
        self.agent_act_level = params['agent_act_level']
        self.experience_replay_pool = [] #experience replay pool <s_t, a_t,
r_t, s_t+1>

        self.experience_replay_pool_size =
params.get('experience_replay_pool_size', 1000)
        self.hidden_size = params.get('dqn_hidden_size', 60)
        self.gamma = params.get('gamma', 0.9)
        self.predict_mode = params.get('predict_mode', False)
        self.warm_start = params.get('warm_start', 0)

        self.max_turn = params['max_turn'] + 4
        self.state_dimension = 2 * self.act_cardinality + 7 *
self.slot_cardinality + 3 + self.max_turn

        self.dqn = DQN(self.state_dimension, self.hidden_size,
self.num_actions)
        self.clone_dqn = copy.deepcopy(self.dqn)

        self.cur_bellman_err = 0

        # Prediction Mode: load trained DQN model
        if params['trained_model_path'] != None:
            self.dqn.model =
copy.deepcopy(self.load_trained_DQN(params['trained_model_path']))
            self.clone_dqn = copy.deepcopy(self.dqn)
            self.predict_mode = True
            self.warm_start = 2


    def initialize_episode(self):
        """ Initialize a new episode. This function is called every time a
new episode is run. """

        self.current_slot_id = 0
        self.phase = 0
        self.request_set = ['moviename', 'starttime', 'city', 'date',
'theater', 'numberofpeople']


    def state_to_action(self, state):
        """ DQN: Input state, output action """

        self.representation = self.prepare_state_representation(state)
        self.action = self.run_policy(self.representation)
        act_slot_response = copy.deepcopy(self.feasible_actions[self.action])
        return {'act_slot_response': act_slot_response,
'act_slot_value_response': None}


    def prepare_state_representation(self, state):
        """ Create the representation for each state """

        user_action = state['user_action']
```

```
        current_slots = state['current_slots']
        kb_results_dict = state['kb_results_dict']
        agent_last = state['agent_action']


################################################################################
        #    Create one-hot of acts to represent the current user action

################################################################################
        user_act_rep =  np.zeros((1, self.act_cardinality))
        user_act_rep[0,self.act_set[user_action['diaact']]] = 1.0


################################################################################
        #     Create bag of inform slots representation to represent the
current user action

################################################################################
        user_inform_slots_rep = np.zeros((1, self.slot_cardinality))
        for slot in user_action['inform_slots'].keys():
            user_inform_slots_rep[0,self.slot_set[slot]] = 1.0


################################################################################
        #    Create bag of request slots representation to represent the
current user action

################################################################################
        user_request_slots_rep = np.zeros((1, self.slot_cardinality))
        for slot in user_action['request_slots'].keys():
            user_request_slots_rep[0, self.slot_set[slot]] = 1.0


################################################################################
        #    Creat bag of filled_in slots based on the current_slots

################################################################################
        current_slots_rep = np.zeros((1, self.slot_cardinality))
        for slot in current_slots['inform_slots']:
            current_slots_rep[0, self.slot_set[slot]] = 1.0


################################################################################
        #    Encode last agent act

################################################################################
        agent_act_rep = np.zeros((1,self.act_cardinality))
        if agent_last:
            agent_act_rep[0, self.act_set[agent_last['diaact']]] = 1.0


################################################################################
        #    Encode last agent inform slots

################################################################################
        agent_inform_slots_rep = np.zeros((1, self.slot_cardinality))
        if agent_last:
            for slot in agent_last['inform_slots'].keys():
                agent_inform_slots_rep[0,self.slot_set[slot]] = 1.0


################################################################################
        #    Encode last agent request slots
```

```
########################################################################
        agent_request_slots_rep = np.zeros((1, self.slot_cardinality))
        if agent_last:
            for slot in agent_last['request_slots'].keys():
                agent_request_slots_rep[0,self.slot_set[slot]] = 1.0

        turn_rep = np.zeros((1,1)) + state['turn'] / 10.


########################################################################
        #  One-hot representation of the turn count?

########################################################################
        turn_onehot_rep = np.zeros((1, self.max_turn))
        turn_onehot_rep[0, state['turn']] = 1.0


########################################################################
        #   Representation of KB results (scaled counts)

########################################################################
        kb_count_rep = np.zeros((1, self.slot_cardinality + 1)) +
kb_results_dict['matching_all_constraints'] / 100.
        for slot in kb_results_dict:
            if slot in self.slot_set:
                kb_count_rep[0, self.slot_set[slot]] = kb_results_dict[slot]
/ 100.


########################################################################
        #   Representation of KB results (binary)

########################################################################
        kb_binary_rep = np.zeros((1, self.slot_cardinality + 1)) + np.sum(
kb_results_dict['matching_all_constraints'] > 0.)
        for slot in kb_results_dict:
            if slot in self.slot_set:
                kb_binary_rep[0, self.slot_set[slot]] = np.sum(
kb_results_dict[slot] > 0.)

        self.final_representation = np.hstack([user_act_rep,
user_inform_slots_rep, user_request_slots_rep, agent_act_rep,
agent_inform_slots_rep, agent_request_slots_rep, current_slots_rep, turn_rep,
turn_onehot_rep, kb_binary_rep, kb_count_rep])
        return self.final_representation

    def run_policy(self, representation):
        """ epsilon-greedy policy """

        if random.random() < self.epsilon:
            return random.randint(0, self.num_actions - 1)
        else:
            if self.warm_start == 1:
                if len(self.experience_replay_pool) >
self.experience_replay_pool_size:
                    self.warm_start = 2
                return self.rule_policy()
            else:
                return self.dqn.predict(representation, {},
predict_model=True)

    def rule_policy(self):
        """ Rule Policy """
```

```
        if self.current_slot_id < len(self.request_set):
            slot = self.request_set[self.current_slot_id]
            self.current_slot_id += 1

            act_slot_response = {}
            act_slot_response['diaact'] = "request"
            act_slot_response['inform_slots'] = {}
            act_slot_response['request_slots'] = {slot: "UNK"}
        elif self.phase == 0:
            act_slot_response = {'diaact': "inform", 'inform_slots':
{'taskcomplete': "PLACEHOLDER"}, 'request_slots': {} }
            self.phase += 1
        elif self.phase == 1:
            act_slot_response = {'diaact': "thanks", 'inform_slots': {},
'request_slots': {} }

        return self.action_index(act_slot_response)

    def action_index(self, act_slot_response):
        """ Return the index of action """

        for (i, action) in enumerate(self.feasible_actions):
            if act_slot_response == action:
                return i
        print act_slot_response
        raise Exception("action index not found")
        return None


    def register_experience_replay_tuple(self, s_t, a_t, reward, s_tplus1,
episode_over):
        """ Register feedback from the environment, to be stored as future
training data """

        state_t_rep = self.prepare_state_representation(s_t)
        action_t = self.action
        reward_t = reward
        state_tplus1_rep = self.prepare_state_representation(s_tplus1)
        training_example = (state_t_rep, action_t, reward_t,
state_tplus1_rep, episode_over)

        if self.predict_mode == False: # Training Mode
            if self.warm_start == 1:
                self.experience_replay_pool.append(training_example)
        else: # Prediction Mode
            self.experience_replay_pool.append(training_example)

    def train(self, batch_size=1, num_batches=100):
        """ Train DQN with experience replay """

        for iter_batch in range(num_batches):
            self.cur_bellman_err = 0
            for iter in range(len(self.experience_replay_pool)/(batch_size)):
                batch = [random.choice(self.experience_replay_pool) for i in
xrange(batch_size)]
                batch_struct = self.dqn.singleBatch(batch, {'gamma':
self.gamma}, self.clone_dqn)
                self.cur_bellman_err += batch_struct['cost']['total_cost']

            print ("cur bellman err %.4f, experience replay pool %s" %
(float(self.cur_bellman_err)/len(self.experience_replay_pool),
len(self.experience_replay_pool)))
```

```
################################################################################
###
    #    Debug Functions

################################################################################
###
    def save_experience_replay_to_file(self, path):
        """ Save the experience replay pool to a file """

        try:
            pickle.dump(self.experience_replay_pool, open(path, "wb"))
            print 'saved model in %s' % (path, )
        except Exception, e:
            print 'Error: Writing model fails: %s' % (path, )
            print e

    def load_experience_replay_from_file(self, path):
        """ Load the experience replay pool from a file"""

        self.experience_replay_pool = pickle.load(open(path, 'rb'))


    def load_trained_DQN(self, path):
        """ Load the trained DQN from a file """

        trained_file = pickle.load(open(path, 'rb'))
        model = trained_file['model']

        print "trained DQN Parameters:", json.dumps(trained_file['params'],
indent=2)
        return model




class StateTracker:
    """ The state tracker maintains a record of which request slots are
filled and which inform slots are filled """

    def __init__(self, act_set, slot_set, movie_dictionary):
        """ constructor for statetracker takes movie knowledge base and
initializes a new episode

        Arguments:
        act_set                 --  The set of all acts availavle
        slot_set                --  The total set of available slots
        movie_dictionary        --  A representation of all the available
movies. Generally this object is accessed via the KBHelper class

        Class Variables:
        history_vectors         --  A record of the current dialog so far in
vector format (act-slot, but no values)
        history_dictionaries    --  A record of the current dialog in
dictionary format
        current_slots           --  A dictionary that keeps a running record
of which slots are filled current_slots['inform_slots'] and which are
requested current_slots['request_slots'] (but not filed)
        action_dimension        --  # TODO indicates the dimensionality of
the vector representaiton of the action
        kb_result_dimension     --  A single integer denoting the dimension
of the kb_results features.
```

```
        turn_count              --  A running count of which turn we are at
in the present dialog
        """
        self.movie_dictionary = movie_dictionary
        self.initialize_episode()
        self.history_vectors = None
        self.history_dictionaries = None
        self.current_slots = None
        self.action_dimension = 10      # TODO REPLACE WITH REAL VALUE
        self.kb_result_dimension = 10   # TODO  REPLACE WITH REAL VALUE
        self.turn_count = 0
        self.kb_helper = KBHelper(movie_dictionary)


    def initialize_episode(self):
        """ Initialize a new episode (dialog), flush the current state and
tracked slots """

        self.action_dimension = 10
        self.history_vectors = np.zeros((1, self.action_dimension))
        self.history_dictionaries = []
        self.turn_count = 0
        self.current_slots = {}

        self.current_slots['inform_slots'] = {}
        self.current_slots['request_slots'] = {}
        self.current_slots['proposed_slots'] = {}
        self.current_slots['agent_request_slots'] = {}


    def dialog_history_vectors(self):
        """ Return the dialog history (both user and agent actions) in vector
representation """
        return self.history_vectors


    def dialog_history_dictionaries(self):
        """  Return the dictionary representation of the dialog history
(includes values) """
        return self.history_dictionaries


    def kb_results_for_state(self):
        """ Return the information about the database results based on the
currently informed slots """

#############################################################################
        # TODO Calculate results based on current informed slots

#############################################################################
        kb_results =
self.kb_helper.database_results_for_agent(self.current_slots) # replace this
with something less ridiculous
        # TODO turn results into vector (from dictionary)
        results = np.zeros((0, self.kb_result_dimension))
        return results


    def get_state_for_agent(self):
        """ Get the state representatons to send to agent """
        #state = {'user_action': self.history_dictionaries[-1],
'current_slots': self.current_slots, 'kb_results':
self.kb_results_for_state()}
        state = {'user_action': self.history_dictionaries[-1],
```

```
'current_slots': self.current_slots, #'kb_results':
self.kb_results_for_state(),

'kb_results_dict':self.kb_helper.database_results_for_agent(self.current_slot
s), 'turn': self.turn_count, 'history': self.history_dictionaries,
                 'agent_action': self.history_dictionaries[-2] if
len(self.history_dictionaries) > 1 else None}
        return copy.deepcopy(state)

    def get_suggest_slots_values(self, request_slots):
        """ Get the suggested values for request slots """

        suggest_slot_vals = {}
        if len(request_slots) > 0:
            suggest_slot_vals =
self.kb_helper.suggest_slot_values(request_slots, self.current_slots)

        return suggest_slot_vals

    def get_current_kb_results(self):
        """ get the kb_results for current state """
        kb_results =
self.kb_helper.available_results_from_kb(self.current_slots)
        return kb_results


    def update(self, agent_action=None, user_action=None):
        """ Update the state based on the latest action """


########################################################################
        #  Make sure that the function was called properly

########################################################################
        assert(not (user_action and agent_action))
        assert(user_action or agent_action)


########################################################################
        #   Update state to reflect a new action by the agent.

########################################################################
        if agent_action:


############################################################
            #   Handles the act_slot response (with values needing to be
filled)

############################################################
            if agent_action['act_slot_response']:
                response = copy.deepcopy(agent_action['act_slot_response'])

                inform_slots =
self.kb_helper.fill_inform_slots(response['inform_slots'],
self.current_slots) # TODO this doesn't actually work yet, remove this
warning when kb_helper is functional
                agent_action_values = {'turn': self.turn_count, 'speaker':
"agent", 'diaact': response['diaact'], 'inform_slots': inform_slots,
'request_slots':response['request_slots']}

                agent_action['act_slot_response'].update({'diaact':
response['diaact'], 'inform_slots': inform_slots,
'request_slots':response['request_slots'], 'turn':self.turn_count})
```

```
            elif agent_action['act_slot_value_response']:
                agent_action_values =
copy.deepcopy(agent_action['act_slot_value_response'])
                # print("Updating state based on act_slot_value action from
agent")
                agent_action_values['turn'] = self.turn_count
                agent_action_values['speaker'] = "agent"


############################################################################
            #   This code should execute regardless of which kind of agent
produced action

############################################################################
            for slot in agent_action_values['inform_slots'].keys():
                self.current_slots['proposed_slots'][slot] =
agent_action_values['inform_slots'][slot]
                self.current_slots['inform_slots'][slot] =
agent_action_values['inform_slots'][slot] # add into inform_slots
                if slot in self.current_slots['request_slots'].keys():
                    del self.current_slots['request_slots'][slot]

            for slot in agent_action_values['request_slots'].keys():
                if slot not in self.current_slots['agent_request_slots']:
                    self.current_slots['agent_request_slots'][slot] = "UNK"

            self.history_dictionaries.append(agent_action_values)
            current_agent_vector = np.ones((1, self.action_dimension))
            self.history_vectors = np.vstack([self.history_vectors,
current_agent_vector])


############################################################################
        #   Update the state to reflect a new action by the user

############################################################################
        elif user_action:
            print user_action

############################################################################
            #   Update the current slots

############################################################################
            for slot in user_action['inform_slots'].keys():
                self.current_slots['inform_slots'][slot] =
user_action['inform_slots'][slot]
                if slot in self.current_slots['request_slots'].keys():
                    del self.current_slots['request_slots'][slot]

            for slot in user_action['request_slots'].keys():
                if slot not in self.current_slots['request_slots']:
                    self.current_slots['request_slots'][slot] = "UNK"

            self.history_vectors = np.vstack([self.history_vectors,
np.zeros((1,self.action_dimension))])
            new_move = {'turn': self.turn_count, 'speaker': "user",
'request_slots': user_action['request_slots'], 'inform_slots':
user_action['inform_slots'], 'diaact': user_action['diaact']}
            self.history_dictionaries.append(copy.deepcopy(new_move))


############################################################################
        #   This should never happen if the asserts passed
```

```
##########################################################################
        else:
            pass


##########################################################################
        #   This code should execute after update code regardless of what
kind of action (agent/user)

##########################################################################

        self.turn_count += 1

    def reward_function(self):
        """ Reward Function 1: a reward function based on the dialog_status
"""
        reward = 1
        user_action = self.history_dictionaries[-2]
        agt_action = self.history_dictionaries[-1]
        aa = 1

        return reward
```

ДОДАТОК Б
Ілюстративний матеріал

Чат-бот на основі навчання з підкріпленням

Автор: студент групи КА-55
Котирло Віталій Володимирович
Керівник: Недашківська Надія Іванівна

Дипломна робота

- Об'єкт дослідження – чат-бот – програма, яка може підтримувати дискусію чи розмову з людиною.
- Предмет дослідження – методи і алгоритми навчання з підкріпленням для побудови діалогової системи у вигляді чат-боту.
- Мета роботи – побудувати чат-бот з використанням методів глибокого навчання з підкріпленням.

Актуальність

- Необхідність автоматизації сфери обслуговування
- Миттєва реакція на запит
- Доповнення до тенденції використання веб-додатків
- Працює незалежно від людського оператора
- Можливість поступової адаптації
- Месенджери зробили даний інтерфейс інтуїтивно зрозумілим
- Заміна функціоналу повноцінного веб чи мобільного додатку

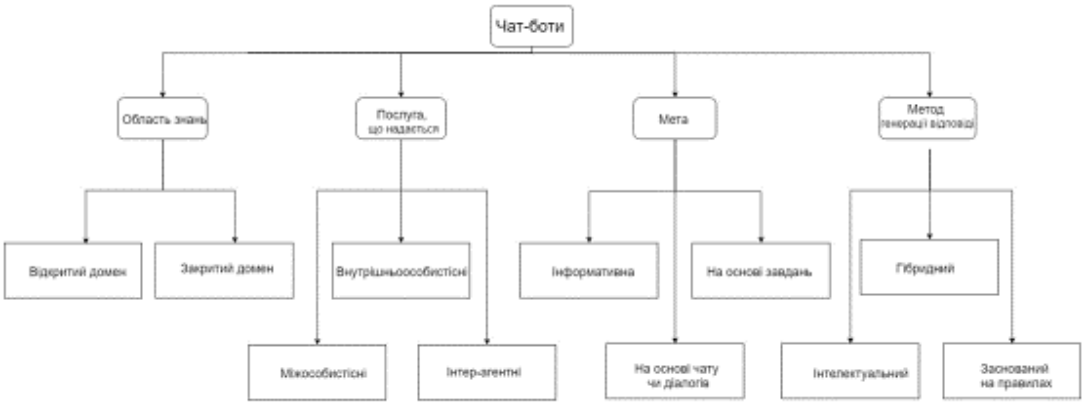
3

Постановка задачі

- Розглянути теоретичну область
- Проаналізувати існуючі засоби розробки чат-ботів та інструментарій
- Розробити діалогову систему у вигляді чат-бота для бронювання квитків в кінотеатр
- Обрати необхідну архітектуру та тип чат-бота для розв'язку заданої задачі
- Інтегрувати методи навчання з підкріпленням

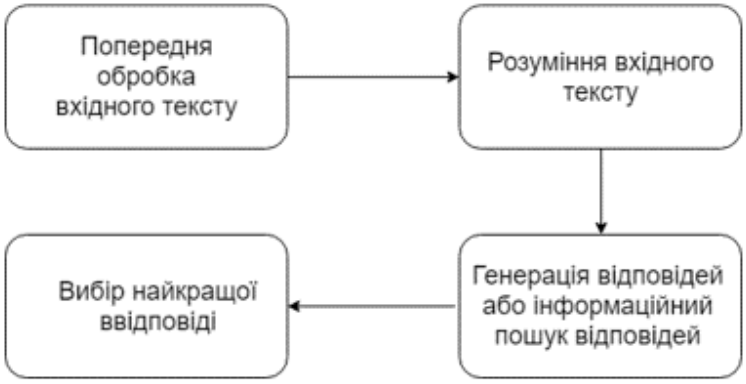
4

Класифікація чат-ботів



5

Загальна схема чат-бота



6

Моделі чат-ботів

Моделі на основі пошуку

- засновані на правилах або які отримують відповіді з заздалегідь визначеного набору відповідей
- якщо всі можливі відповіді можна уявити, то модель на основі пошуку працює досить якісно
- системи бронювання, системи FAQ

Генеративні моделі

- підтримувати ієрархічну структуру базових значень мови
- використовуються, коли доступна велика кількість даних і система може бути навчена на них
- використовують алгоритми NLP і NLU для обробки вхідних даних і генерування пропозицій, нейронні мережі, LSTM, SVM, моделі послідовностей

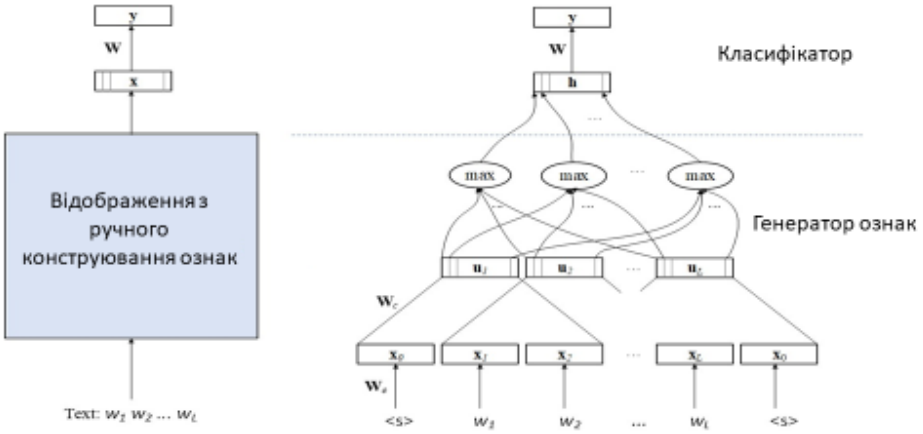
7

Порівняльний аналіз готових рішень

Номер	Платформа	Класифікація намірів	Вилучення суб'єктів	Аналіз настрою	Вилучення ключових слів	Базованість на правилах	Мова
1	Recast. AI	+	+	+	-	+	3 мови: англійська, французька та іспанська мають всі функціональні можливості, 17 інших мають обмежені можливості вилучення об'єктів і класифікацію намірів. Один агент може підтримувати кілька мов одночасно.
2	Wit.ai	+	+	+	-	+	50 мов доступні або 39 з яких знаходяться в бета-версії
3	Api.ai	+	+	-	-	-	Підтримуються 15 мов. З кожним запитом, мовний тег повинен бути відправлений. Один агент підтримує одну умову, без можливості зміни
4	Chatfuel	+-	-	-	-	+	Підтримується більше 40 мов
5	Pandora Bots	+	+	-	+	+	Багатомовна
6	Microsoft Bots, LUIS	+	+	+	+	-	Більше 30 мов доступні
7	Google Natural language API	-	+	+	+	-	Підтримка 9 мов, виявлення настроїв суб'єктів і класифікація тексту доступні тільки для англійської мови
8	IBM Watson	-	+	+	+	-	Різні послуги доступні для багатьох мов в тому числі виявлення ключових слів та об'єктів. Англійська та іспанська мають найбільший пакет послуг

8

Схема класичного машинного навчання і глибокого навчання



9

Навчання з підкріпленням

10

Можливість застосування навчання з підкріпленням

1. Діалог можна сформувати як процес прийняття рішень
2. Прийняття рішень може бути представлене у вигляді математичних структур варіантів дій над Марковськими процесами прийняття рішень
3. Якщо ми розглядаємо кожен варіант як дію, то до системи можна застосовувати методи навчання з підкріпленням
4. На кожному кроці агент спостерігає за поточним станом і вибирає дію відповідно до лінії стратегії

11

Моделювання взаємодії агента та середовища

Дискретно-марковський процес прийняття рішень, або MDP, описаний п'ятіркою $M = (S, A, P, R, \gamma)$:

- S є, можливо, нескінченним набором станів, в яких може перебувати середовище;
- A - це, можливо, нескінченний набір дій, які агент може прийняти в стані;
- $P(s` | s, a)$ дає ймовірність переходу середовища в новий стан s` після дії a, що приймається в стані s;
- $R(s, a)$ - середня винагорода, що негайно отримується агентом після виконання дії a в стані s;
- $\gamma \in (0, 1)$ - коефіцієнт дисконтування.

Процес може бути записаний як траєкторія $(s1, a1, r1, ...)$, яка генерується наступним чином: на етапі $t = 1, 2, . . .$,

12

Агент RL

- Метою є максимізація довгострокової винагороди шляхом прийняття оптимальних дій
- З огляду на стратегію π, значення стану s є середньою дисконтованою довгостроковою винагородою від π

$$V^{\pi}(s) := \mathbb{E}[r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots | s_1 = s, a_i \sim \pi(s_i), \forall i \geq 1]$$

13

Q-learning

$$Q^{\pi}(s,a) := \mathbb{E}[r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots | s_1 = s, a_1 = a, a_i \sim \pi(s_i), \forall i > 1]$$

Оптимальна стратегія може бути відразу знайдена, якщо доступна оптимальна Q-функція

$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

припускаємо, що Q-функція має задану параметричну форму, параметризовану деяким вектором ϑ.

$$\theta \leftarrow \theta + \alpha \left(r + \gamma \max_{a'} Q(s', a'; \theta) - Q(s, a; \theta) \right) \nabla_{\theta} Q(s, a; \theta)$$

14

Policy Gradient

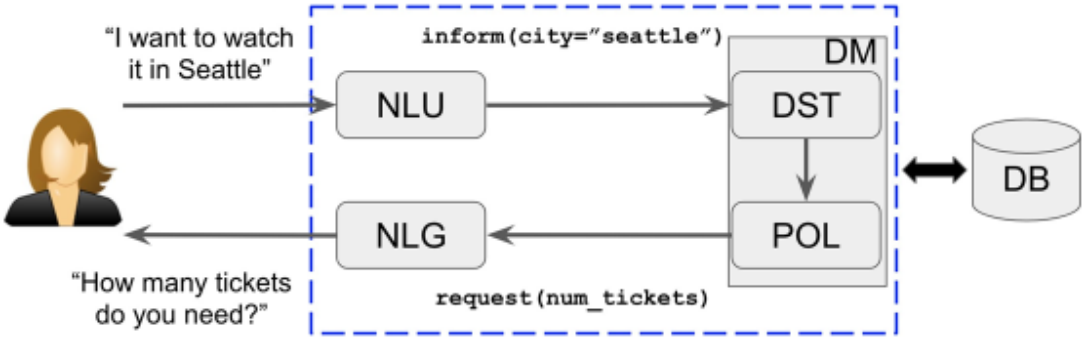
- Оптимізуємо стратегію безпосередньо, без необхідності в Q-функції
- Стратегія безпосередньо параметризована по ϑ, а $\pi(s; \vartheta)$ є розподілом над діями
- Стратегія оцінюється за середньою довгостроковою винагородою, що потрапляє в траєкторію довжини H

$$H, \tau = (s_1, a_1, r_1, \ldots, s_H, a_H, r_H) \qquad J(\theta) := \mathbb{E}\left[\sum_{t=1}^{H} \gamma^{t-1} r_t | a_t \sim \pi(s_t; \theta)\right]$$

- Якщо можна оцінити градієнт $\nabla_\theta J$ по траєкторіям, то можна застосувати метод SGD, щоб максимізувати J

15

Архітектура та модулі чат-бота



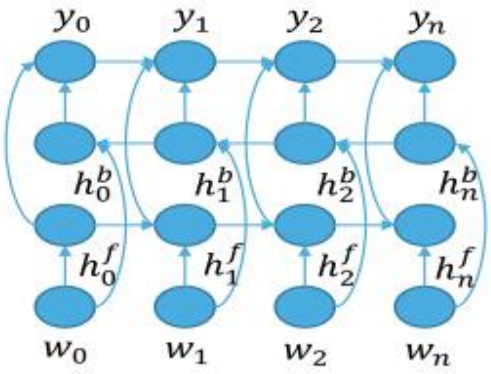
16

Модуль обробки природньої мови NLU

Приклад виводу NLU, де висловлювання (W) використовується для прогнозування домену (D), наміру (I) і тегування слота (S).

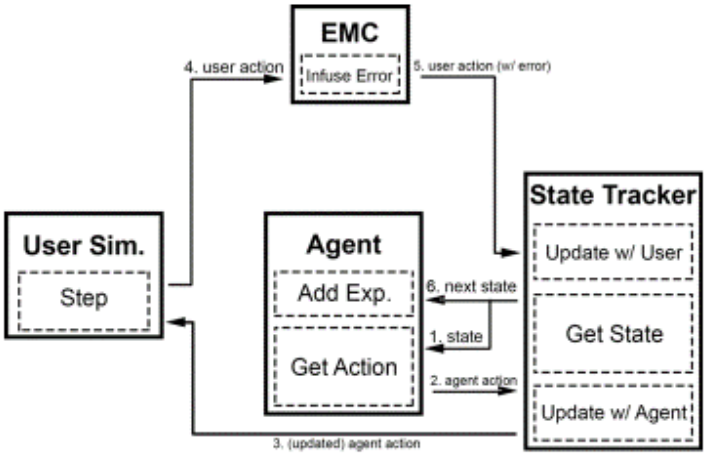
W	find	recent	comedies	by	james	cameron
	↓	↓	↓	↓	↓	↓
S	O	B-date	B-genre	O	B-dir	I-dir
D	movies					
I	find_movie					

Модель bLSTM



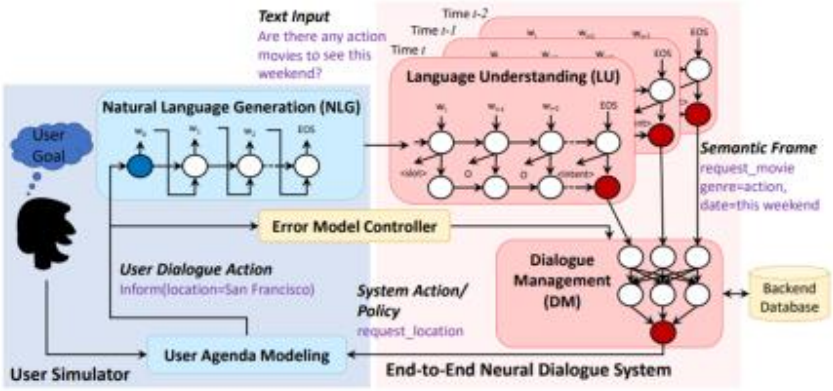
17

Запропонований алгоритм тренування агента



18

Обрана архітектура чат-бота



19

Семантичні фрейми

Акти діалогу

```
{
    'intent': 'request',
    'inform_slots':
    {
        'city': 'seattle'
    },
    'request_slots':
    {
        'theater': 'UNK',
        'starttime': 'UNK'
    }
}
```

Цілі користувача

```
{
  request_slots:
  {
    ticket: UNK
    theater: UNK
    start_time: UNK
  },
  inform_slots:
  {
    number_of_people: 3
    date: tomorrow
    movie_name: batman vs. superman
  }
}
```

20

Слоти та наміри

Наміри

```
request
inform
confirm_question
confirm_answer
greeting
closing
multiple_choice
thanks
welcome
deny
not_sure
```

Слоти

```
actor
actress
city
closing
critic_rating
date
description
distanceconstraints
genre
greeting
implicit_value
movie_series
moviename
mpaa_rating
numberofpeople
numberofkids
taskcomplete
other
price
seating
starttime
state
theater
theater_chain
video_format
zip
result
ticket
mc_list
```

21

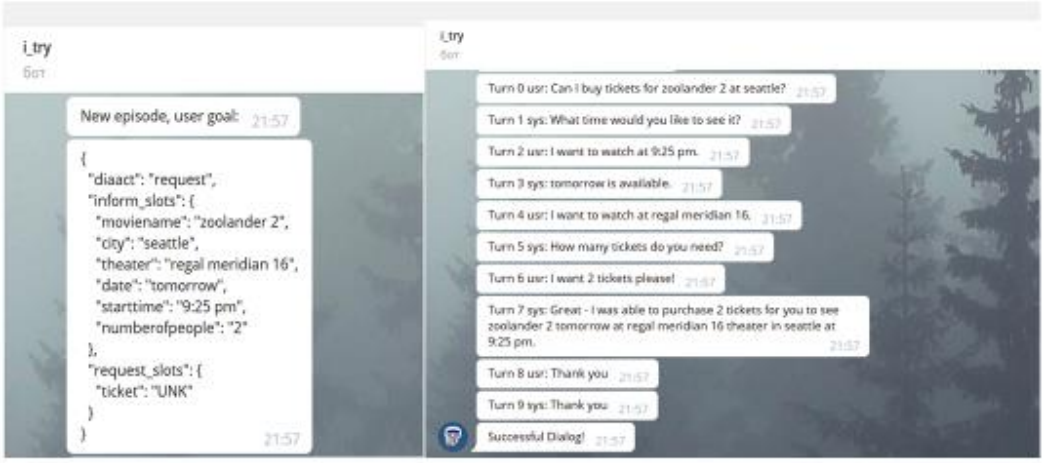
Інтеграція розробленої системи в Telegram бот

@itry_bot

https://telegram.me/itry_bot

22

Приклад вдалої роботи чатботу



23

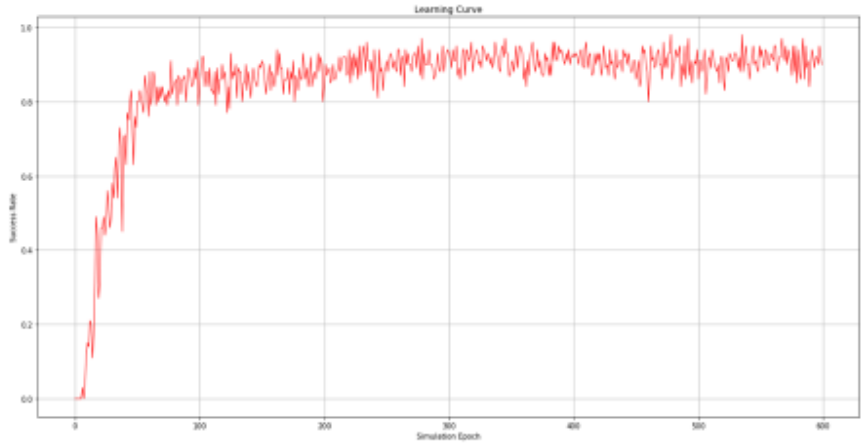
Приклад невдалої роботи чатботу

```
New episode, user goal:
{
  "request_slots": {
    "ticket": "UNK",
    "theater": "UNK",
    "starttime": "UNK"
  },
  "diaact": "request",
  "inform_slots": {
    "date": "this friday",
    "numberofpeople": "4",
    "moviename": "star wars"
  }
}
```

```
Turn 0 usr: What is the start time for star wars this friday?
Turn 1 sys: 12:05pm is available.
Turn 2 usr: Can I get 4 tickets?
Turn 3 sys: Do you have any constraint for the price?
Turn 4 usr: i will be a date in mind
Turn 5 sys: description is not available!
Turn 6 usr: Could you help me to book the tickets?
Turn 7 sys: description is not available!
Turn 8 usr: Could you help me to book the tickets?
Turn 9 sys: description is not available!
Turn 10 usr: Could you help me to book the tickets?
Turn 11 sys: description is not available!
Turn 12 usr: Could you help me to book the tickets?
Turn 13 sys: description is not available!
Turn 14 usr: Could you help me to book the tickets?
Turn 15 sys: description is not available!
Turn 16 usr: Could you help me to book the tickets?
Turn 17 sys: description is not available!
Turn 18 usr: Could you help me to book the tickets?
Turn 19 sys: description is not available!
```

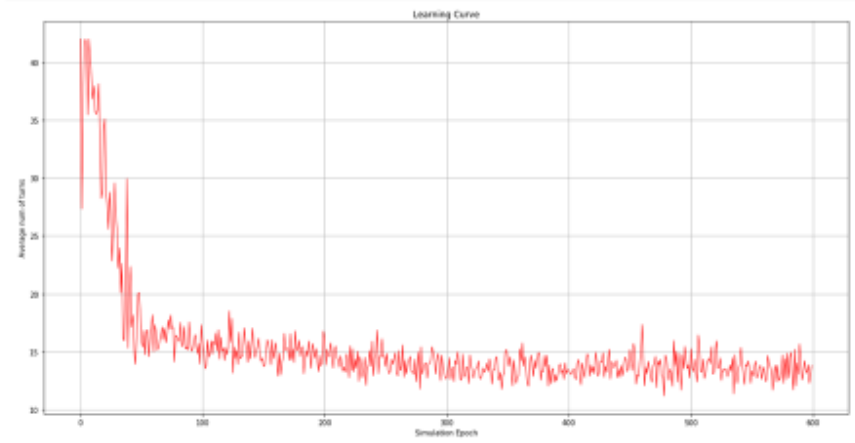
24

Графік успішності діалогів



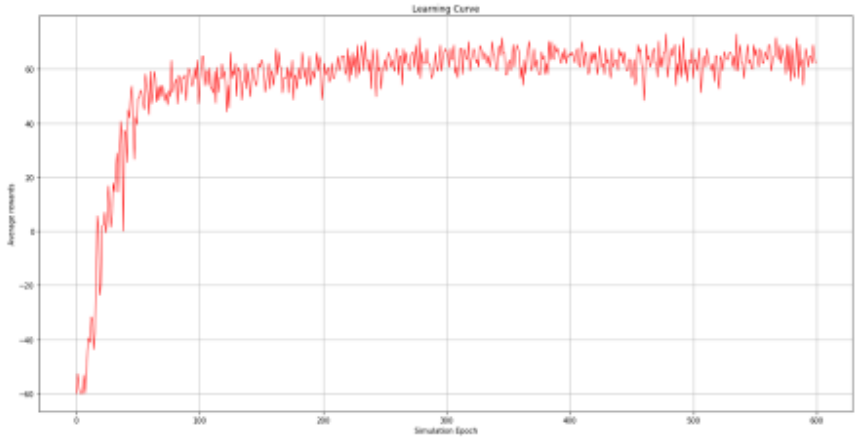
25

Графік середньої кількості кроків в діалозі



26

Графік середньої винагороди



27

Результати

- Розглянуто теоретичні основи побудови чат-ботів, Марківські моделі та методи навчання з підкріпленням
- Виконано аналіз існуючих архітектур та типів чат-ботів
- Виконано порівняльний аналіз існуючого інструментарію
- Обрано необхідну архітектуру та тип чат-бота для розв'язання задачі розробки діалогової системи для бронювання квитків в кінотеатр
- Розроблено модулі діалогової системи у вигляді чат-бота для бронювання квитків в кінотеатр

28