ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ СЕМЕНА КУЗНЕЦЯ

ФАКУЛЬТЕТ ЕКОНОМІЧНОЇ ІНФОРМАТИКИ

КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ

Пояснювальна записка

до дипломного проєкту

бакалавра

на тему: «Розроблення модуля для створення та використання діалогової моделі на основі бібліотеки Tensorflow»

Виконав: студент другого року навчання за скороченою програмою,
групи 6.04.121.012.18.1,
спеціальності 121
«Інженерія програмного забезпечення»
Яковлєв Д. О.

Керівник: к.т.н., проф. кафедри ІС ХНЕУ
Щербаков О. В.

Харків – 2020 рік

РЕФЕРАТ

Пояснювальна записка: 50 с., 26 рис., 32 джерела, 2 табл., 1 додаток.

Об'єкт дослідження: діалогова модель на основі бібліотеки Tensorflow.

Мета роботи: Розроблення модуля для створення та використання діалогової моделі на основі бібліотеки Tensorflow.

Методи дослідження. При вирішенні поставленого завдання використовувалися наукові досягнення в областях штучного інтелекту і нейронних мереж.

Значення отриманих результатів полягає у проведенні аналізу та виявленні недоліків у навчанні моделі ввести діалог, а також в реалізації нейронної діалогової моделі на основі використання нейронної мережі.

Практична цінність результатів полягає в створенні програмних модулів, програмного продукту, які дозволяють оцінити переваги машинного навчання на базі нейронних мереж.

Область застосування. Розроблена діалогова система може застосовуватися для вирішення широкого спектру завдань, зокрема, для машинного навчання, ведення діалогу зі штучним інтелектом.

Значення роботи та висновки. Удосконалена методика дозволяє проектувати діалогові системи зі значним скороченням як матеріальних витрат, так і тимчасових.

Прогнози щодо розвитку досліджень. Розробити універсальні модулі, які можуть бути використані для підтримки проектування діалогових систем з різних програмних платформ. Розробити комплекс програмних засобів і призначений для користувача інтерфейс для графічного представлення результатів, ведення діалогу зі штучним інтелектом.

Список ключових слів: НЕЙРОННА МЕРЕЖА, РЕКУРЕНТНА НЕЙРОННА МЕРЕЖА, ШТУЧНИЙ ІНТЕЛЕКТ, ДІАЛОГОВА МОДЕЛЬ, PYTHON, TENSORFLOW.

ABSTRACT

Explanatory note: 50 p., 26 img., 32 source, 2 tab., 1 annex.

Object of study: a dialog model at the Tensorflow Library.

The purpose of the project: Developed a software module to create and use a dialog model using Tensorflow library.

Methods of research. In addressing the task used scientific advances in the areas of artificial intelligence and neural networks.

The value of the results obtained is to carry out the analysis and identify the disadvantages of teaching the model to have the dialog, as well as in the implementation of the neural dialogue model based on the use of the neural network.

The practical value of work is to create software modules, software products that allow us to assess the benefits of machine learning based on neural networks.

The scope. The developed dialogue system can be used to solve a wide range of tasks, in particular, for machine learning, dialogue with artificial intelligence.

The value of the work and conclusions. The advanced technique allows designing dialog systems with significant reductions both material costs and temporary.

Projections on development research. Develop universal program modules that can be used to support the design of divine systems from various software platforms. Develop a set of software tools and a user interface for graphic presentation of results, conducting a file with artificial intelligence.

Key words list: NEURAL NETWORK, RECURRENT NEURAL NETWORK, ARTIFICIAL INTELLIGENCE, DIALOGUE MODEL, PYTHON, TENSORFLOW.

ЗМІСТ

ПЕРЕЛІК ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

RNN (англ. Recurrent neural network) — вид нейронних мереж, де зв'язки між елементами утворюють спрямовану послідовність;

LSTM (англ. Long short-term memory) — це архітектура рекурентних нейронних мереж;

GRU (англ. Gated recurrent units) — це архітектура рекурентних нейронних мереж;

API (англ. Application programming interface) — це інтерфейс програмування, інтерфейс створення додатків;

ШІ — штучний інтелект;

ПЗ — програмне забезпечення;

ОС — операційна система;

Чат-бот — віртуальний співрозмовник, програма, яка створена для імітації поведінки людини при спілкуванні з одним або декількома співрозмовниками;

Месенджер — клієнтська програмна система миттєвих повідомлень.

ВСТУП

Моделювання розмови — важливе завдання в розумінні природної мови і машинного інтелекту. Хоча й існують інші підходи моделювання, але вони часто обмежуються конкретними сферами (наприклад, замовлення авіаквитків). У цій роботі досліджується підхід моделювання, який використовує недавно запропоновану структуру перетворення послідовності (sequence to sequence). Ця структура може передбачати необхідне речення, враховуючи попереднє речення або висловлювання в розмові. Суть цієї структури полягає в тому, що її можна навчити end-to-end. Тому ця структура може генерувати розмови, враховуючи великий набір даних. Вона здатна витягати знання як з набору даних, специфічною для області розмови, так і з великого, загального набору даних. Наприклад в наборі даних довідкової служби IT, модель може знайти рішення технічної проблеми за допомогою діалогу.

Метою даної бакалаврської проєкту є реалізація нейронної діалогової моделі за допомогою бібліотеки машинного навчання Tensorflow.

Для досягнення поставленої мети в роботі сформульовані і вирішені такі завдання:

1. Проведення аналізу та виявлення недоліків існуючого підходу до розробки діалогової моделі.

2. Реалізація нейронної діалогової моделі на основі використання нейронної мережі і бібліотеки машинного навчання Tensorflow.

Об’єкт дослідження – діалогова модель.

Ідея дослідження полягає в реалізації нейронної діалогової моделі на мові Python за допомогою бібліотеки Tensorflow.

Методи дослідження. При вирішенні поставленого завдання використовувалися наукові досягнення в областях обробки природної мови, машинного навчання і нейронних мережах.

Очікувані результати:

1. Сформований аналіз підходу до розробки діалогових систем, а також виявлення недоліків;

2. Реалізація нейронної діалогової моделі на основі використання нейронної мережі і бібліотеки машинного навчання Tensorflow.

Наукова новизна отриманих результатів полягає в реалізації нейронної діалогової моделі з використанням бібліотеки машинного навчання Tensorflow.

Практичне значення отриманих результатів полягає в реалізації нейронної діалогової моделі за допомогою бібліотеки машинного навчання Tensorflow, що дозволяють оцінити переваги проектування діалогових систем.

Робота складається з вступу, трьох розділів і висновків. Містить 61 сторінку друкованого тексту, в тому числі 50 сторінок тексту основної частини з 26 рисунками, списку з 32 використаних джерел з найменуваннями на сторінках, додатку на 11 сторінках.

1 АНАЛІЗ СТАНУ ПИТАННЯ «ОСОБЛИВОСТІ ВЕДЕННЯ БЕСІДИ З ВІРТУАЛЬНИМ СПІВРОЗМОВНИКОМ»

1.1 Визначення ШІ

На сучасному етапі розвитку штучний інтелект досяг рівня самовдосконалення, що безпосередньо відбивається в алгоритмізації і оптимізації процесів управління складними системами, а також самоорганізації процесів в автоматизованих системах управління. Штучний інтелект на початку XXI століття вже досяг такого рівня розвитку, що за допомогою сучасних суперкомп'ютерів, потужність яких вимірюється сотнями терафлопів, може самостійно вирішувати алгоритмічні, математичні, програмні завдання настільки високого рівня, що при цьому удосконалює заданий раніше початковий програмний код, алгоритм до рівня самостійного прийняття рішень, оптимізації процесів управління. Подібного роду завдання ставилися і раніше, але потужність обчислювальних машин не дозволяла обробляти великі обсяги даних з настільки потужною структурою алгоритмів. В силу своїх можливостей алгоритмізація і оптимізація процесів і явищ в XXI столітті досягла величезних масштабів в управлінні складними об'єктами, системами з удосконаленням штучного інтелекту. Сучасні системи об'єктів управління стали самонавчальними, самовдосконалюються за рахунок впровадження штучного інтелекту. В даний час навіть сучасні суперкомп'ютери за своїми фізичними ресурсами поки за більшістю параметрів поступаються людському мозку, перевершуючи його лише по швидкості обчислень. Проте, прогрес в області штучного інтелекту вражає, так як розвиток комп'ютерних технологій йде дуже швидкими темпами. І можна припустити, що в доступному для огляду майбутньому комп'ютери зрівняються за своїми можливостями з людським мозком і перевершать його [2].

Відображення в XXI столітті розвитку штучного інтелекту, його вдосконалення все більше знаходить рішення в нейронних мережах. Якщо наблизити поняття штучного інтелекту до людського фактору, то можна знайти визначення нейронних мереж. Нейронні мережі будуються за принципом організації та функціонування біологічних нейронних мереж, тобто мереж нервових клітин живого організму. Поняття біологічних нейронних мереж виникло при вивченні процесів, що протікають в мозку. Так спроба моделювання біологічних нейронних мереж привела до розробки алгоритмів, де отримані моделі на основі відомих і розроблених алгоритмів стали використовувати в задачах ідентифікації об'єктів управління для практичних цілей. Нейронні мережі являють собою систему багатозв'язних між собою процесорів — штучних

нейронів, які є досить простими за своєю структурою. Подібні процесори нейронної мережі обробляють як сигнали, що знаходяться на них, так і сигнали, надіслані ними на інші процесори нейронної мережі. Будучи з'єднаними в досить велику мережу з керованою взаємодією, такі локально прості процесори разом здатні виконувати досить складні завдання.

Нейронні мережі є основним напрямком по вивченню можливості моделювання природного інтелекту за допомогою комп'ютерних алгоритмів. Варто відзначити, що нейронні мережі не програмуються, а навчаються. Можливість навчання — одне з головних переваг нейронних мереж перед алгоритмами. Технічно навчання полягає в знаходженні коефіцієнтів зв'язків між нейронами. В процесі навчання нейронна мережа здатна виявляти складні залежності між вхідними даними і вихідними, а також виконувати узагальнення. Нейронні мережі — в принципі високо паралельні пристрої, що дозволяють революційно прискорити обробку інформації. Нейронні мережі можна реалізувати в якості програмного забезпечення для звичайного комп'ютера (тоді прощавайте швидкісні переваги паралельності), або як апаратне забезпечення: аналогове, цифрове або змішане. Найвища швидкість і гнучкість досягається на спеціальних комп'ютерах — нейрокомп'ютерах з використанням відповідного програмного забезпечення [3].

Перш ніж нейронна мережа стане здатна що-небудь обробляти, її слід навчити. По суті, поняття «тренінг» ближче до істини, але історично склалося вживати термін «навчання». Ненавчена нейронна мережа не має навіть рефлексів, тобто на будь-які зовнішні впливи її реакція буде хаотичною. Навчання складається в багаторазовому пред'явленні характерних прикладів до тих пір, поки нейронна мережа на свій вихід не стане видавати бажаний відгук. Залежно від призначення нейронної мережі задається певний бажаний відгук, наприклад, номер класу для класифікації зображень. У загальному випадку бажаний відгук задається «вчителем». У разі передбачення (точніше буде сказати прогнозування) «вчитель» формує вхідні дані, вибирає число кроків передбачення на виході і відфільтровує непотрібні компоненти в вихідної тимчасової послідовності, щоб задати більш відповідний бажаний відгук. Наприклад, іноді заздалегідь відомо, що високочастотні флуктуації передбачати немає необхідності, тому їх фільтрують і отримують бажаний відгук у вигляді згладженої вихідної послідовності.

Різниця між реальним виходом нейронної мережі та бажаним відгуком називається помилкою. Величина і знак цієї помилки служать для самоадаптації вагових коефіцієнтів, які на старті навчання задаються випадковими числами. Механізм процесу навчання полягає в цілеспрямованому змінюванні

(самоадаптації) вагових коефіцієнтів (іноді і деяких додаткових параметрів) і називається правилом або алгоритмом навчання. Критерієм навченості є планомірне, асимптотичне зменшення середньоквадратичної помилки, тобто головне, щоб вона зменшувалася з кожним новим повтором якогось прикладу. Навчання нейронних мереж завершується, коли середньоквадратична помилка досягне деякої наперед заданої величини або коли нейронна мережа почне правильно (з необхідною точністю) обробляти дані з окремого тестового набору характерних прикладів.

Сама по собі ідея самоорганізації архітектури нейронних мереж не нова. Однак, на все свій час. Саме зараз кількість досліджень переходить в якість результатів, і саме зараз впровадження нейронних мереж розгортається широким фронтом [5].

У розвитку ШІ намагалися відшукати загальні методи вирішення широкого класу задач. Але розробка програмних засобів виявилася занадто трудомістким заняттям, що не принесло вагомих результатів в дослідженні і розробці ШІ. У період 60-х років почалося зародження евристичного програмування, що призвело до скорочення кількості операцій перебору в просторі пошуку. Це правило теоретично не обґрунтовується, але дозволяє скоротити кількість операцій в просторі пошуку. Друге «народження» ШІ пережив в 80-х роках. В Японії був створений нейрокомп'ютер, в якому обмеження по пам'яті і швидкодії були практично зняті. З'явилися паралельні комп'ютери з великою кількістю процесорів. Основна область застосування нейрокомп'ютерів полягала в розпізнаванні образів. В даний час використовуються три підходи до створення нейромереж: апаратний, програмний і гібридний — поєднання апаратного з програмним. На закінчення варто відзначити, що ШІ має за своєю структурою якийсь алгоритм послідовності дій прийнятих рішень. Однак ШІ обмежується вибором прийняття рішень, тоді як людина не обмежується в більшості ситуацій, а, отже, людський фактор не настільки передбачуваний як ШІ. Передбачення людського фактора не може дати настільки хороший прогноз поведінки людини, в той час як прогнозування ШІ можливо з високим ступенем ймовірності [4].

1.2 Віртуальний співрозмовник

Віртуальний співрозмовник (англ. Chatbot) — це комп'ютерна програма, яка створена для імітації мовної поведінки людини при спілкуванні з одним або декількома співрозмовниками. По відношенню до віртуальних співрозмовників вживається також назва програма-співрозмовник.

Одним з перших віртуальних співрозмовників була програма Еліза (рис. 1.1), створена в 1966 році Джозефом Вейзенбаумом[19]. Еліза пародіювала мовну поведінку психотерапевта, реалізуючи техніку активного слухання, перепитуючи користувача і використовуючи фрази типу «Будь ласка, продовжуйте» [21].

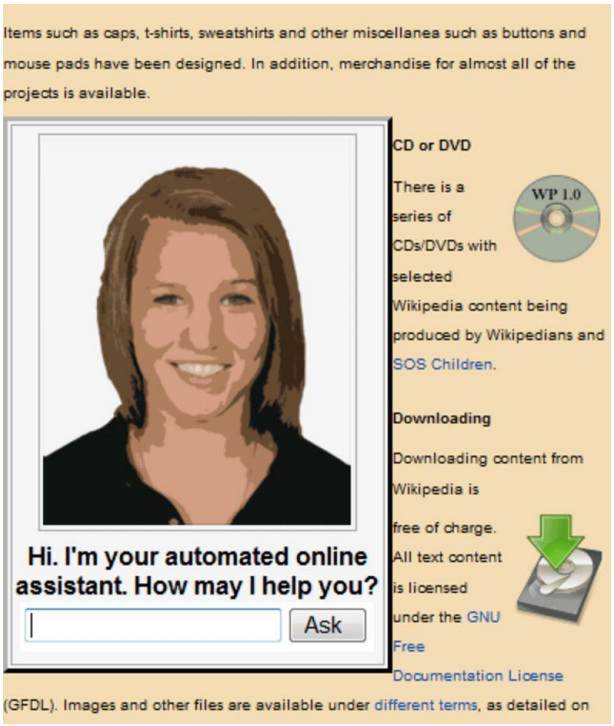

Рисунок 1.1 — Одна з перших програм-співрозмовників Еліза

Передбачається, що ідеальна програма-співрозмовник повинна пройти тест Тьюринга [9]. Проводяться щорічні конкурси програм-співрозмовників (в основному англомовних). Один з найвідоміших — конкурс Лебнера [7].

Дане віртуальним співрозмовникам визначення не зовсім точно. Справа в тому, що цілі конкретних діалогів між людьми розрізняються. Можна просто «поговорити», а можна обговорити важливу проблему. Реалізація останнього типу діалогу являє додаткову проблему: навчити програму мислити [5].

Тому функціональність більшості сучасних програм обмежується можливістю ведення простих бесід.

Програми, здатні розуміти окремі висловлювання користувача, утворюють клас програм з природно-мовним інтерфейсом. Наприклад, питально-відповідна система.

Створення віртуальних співрозмовників межує з проблемою загального штучного інтелекту, тобто єдиної системи (програми, машини), що моделює інтелектуальну діяльність людини [16].

1.2.1 Принцип дії діалогових агентів

Віртуальні співрозмовники працюють з «живою» мовою. Обробка природної мови, особливого розмовного стилю — гостра проблема штучного інтелекту. І звичайно, сучасні програми-співрозмовники — лише спроби імітувати розумний діалог з машиною [8].

Як будь-яка інтелектуальна система, віртуальний співрозмовник має базу знань. У найпростішому випадку вона являє собою набори можливих питань користувача і відповідних їм відповідей. Найбільш поширені методи вибору відповіді в цьому випадку такі:

- реакція на ключові слова: Даний метод був використаний в Елізе. Наприклад, якщо фраза користувача містила слова «батько», «мати», «син» і інші, Еліза могла відповісти: «Розкажіть більше про вашу сім'ю»;
- збіг фрази: мається на увазі схожість фрази користувача з тими, що містяться в базі знань. Може враховуватися також порядок слів;
- збіг контексту: часто в інструкціях до програм-співрозмовникам просять не використовувати фрази, насичені займенниками, типу: «А що це таке?» Для коректної відповіді деякі програми можуть проаналізувати попередні фрази користувача і вибрати одну з найбільш прийнятних відповідей;
- інший тип — інтелектуальний варіант, що генерую відповідь, враховуючи все повідомлення користувача.

1.3 Особливості та проблеми віртуального спілкування

Питання застосування систем віртуального спілкування на основі штучного інтелекту досліджують протягом багатьох років. На сьогоднішній день проблема віртуального спілкування актуальна завдяки швидкості доступу до інформації, можливості одночасної роботи в системі багатьох користувачів, обміну інформацією, взаємодії з метою вирішення будь-яких питань, підтримки навчання, комунікації з клієнтами і партнерами по бізнесу, проведення аналітичних досліджень, збору необхідної інформації, підвищення кваліфікації та інших переваг [12].

Основними питаннями в створенні систем спілкування є розробка моделі спілкування, моделі учасника спілкування, розвиток засобів, в першу чергу, семантичних і прагматичних, опису навколишнього середовища (моделі мови, моделі користувача, моделі навколишнього середовища, моделі системи спілкування). Тому для вирішення цих питань необхідно визначення принципів роботи, особливостей імітації мовної поведінки людини в процесі спілкування,

розробка моделі спілкування, написання чат-бота. Серед програм-співрозмовників є програми, створені на основі штучного інтелекту.

При розробці таких програм необхідно знати психологію, а також принципи побудови фраз людської мови. Більш того, якщо правильно визначити мовні обмеження і предметну область, то методами, що вже існують можна отримати системи, придатні для спілкування. З точки зору теорії мови і спілкування необхідна розробка семантичного опису структур текстів і пропозицій. З точки зору моделі навколишнього середовища основним обмеженням є необхідні ресурси для подолання динамічно мінливого спілкування. Ця пов'язано з проблемою сприйняття системою тверджень і навчанням системи.

1.3.1 Основні функції і принципи роботи чат-боту

Комп'ютерну програму бот використовують для введення-виведення повідомлень і виконання різних функцій. Боти виконують такі основні функції: службові, інформаційно розважальні, функції утиліт. Розглянемо їх докладніше. Службові функції ботів полягають у веденні логів чату обліку прав учасників, забезпеченні заходів безпеки, забезпеченні можливості конференції між більш ніж двома користувачами, коли в протоколі відсутня така функція. Інформаційно-розважальними функціями, які забезпечують бот, можуть бути довідка, словники, віртуальні співрозмовники, ігри. Також в боті використовують утиліти, наприклад, перекладач, калькулятор, коментатор, пошук. Принцип роботи чат-бота полягає в реалізації етапів: бот приймає вхідні повідомлення, аналізує їх і відсилає результат і / або виконує команду.

1.3.2 Основні проблеми чат-ботів

Величезна кількість компаній прагнуть для своїх потреб впровадити інтелектуальних помічників, завдяки яким можливо значно скоротити трудомісткість, а також підвищити ефективність систем підприємства, пов’язаних з таким типом робіт. Але є декілька основних проблем, котрі притаманні інтелектуальним системам, з якими стикаються як розробники, так і користувачі. Основні з них:

1) принцип «увімкнув-запрацювало» важко застосувати до інтелектуальних ботів.

Впроваджувати чат-ботів на основі штучного інтелекту дуже складно, особливо коли мова заходить про по-справжньому просунутих AI-системах, наприклад, система IBM Watson. Щоб навчити бота, потрібно маса вихідних даних, часу і знань. Є навіть окрема дисципліна — «напрацювання комунікаційних навичок (НКН)» для AI. Фахівці з НКН — це психолінгвісти, які

продумують діалоги за певними тематиками і бізнес-процесами, а потім вчать роботів спілкуватися на ці теми.

Ось що говорить фахівець з НКН Томас Хебнер з компанії Nuance Communications: «У сегменті AI-ботів мене найбільше дивує необґрунтований оптимізм їх постачальників. Вони явно перебільшують можливості роботів. Останній рік я провів в контакт-центрах і переконував їх власників, що принцип «увімкнув-заробило» — це не про чат-ботів. Для їх налаштування потрібні час, знання і дані» [30].

Слова Хебнера підтверджує засновник компанії Humanotics Девід Нейлор: «Один з австралійських банків з'ясував, що клієнти запитують про баланс на рахунку 200 різними способами. Банк витратив багато часу, щоб запрограмувати всі ці варіанти. І тільки тоді все запрацювало, як потрібно» [14];

2) чат-боти найкраще показують себе у певних сценаріях.

У виробничих галузях робототехніка розвивається дуже швидко. Але одна справа — поставити робота на складальний конвеєр замість людини, що виконує однотипні дії. І зовсім інше — замінити чат-ботом, наприклад, оператора контакт-центру, навченого реагувати на нестандартні ситуації.

Ніколь Міллард, керівник компанії BT за новими технологіями пояснює: «Машини хороші, коли ситуацію можна описати певним набором правил. Але як тільки справа доходить до нешаблонних, складних і емоційних питань, боти пасують. Мій улюблений приклад — чат-бот Margot з німецької мережі супермаркетів Lidl. Margot відмінно відповідає на питання, яке вино краще подавати до якої страви. Але просто спробуйте запитати його про що-небудь ще» [15].

3) різна мовна стилістика діалогів може вимагати впровадження різних чат-ботів.

Важливо пам'ятати, що користувачі можуть писати в різних емоційних стилях. Емоції багато в чому залежать від причини звернення. Томас Хебнер ділиться таким прикладом. «Ми працювали з авіакомпанією, яка вважала, що її бренд повинен бути веселим, а комунікації — дотепними. Компанія вирішила впровадити систему інтерактивної мовної відповіді (IVR). Вона включалася, тільки якщо оператори контакт-центру не встигали відповісти на виклики. Причому обробляла тільки ті запити, які стосувалися розкладу рейсів. Ми вирішили, що дотепні відповіді тут недоречні. Люди уточнюють час вильоту і запитують про затримки рейсів, часто перебуваючи в стані стресу. Їм потрібна швидка і небагатослівна відповідь. Тому ми вибрали дуже консервативний сценарій діалогу і спокійний голос» [29].

Те ж правило діє відносно чат-ботів. У деяких випадках потрібно повністю відмовитися від стилю спілкування. Стилістика відповідей повинна відповідати типу запиту, але чат-боти поки не вміють змінювати її «на льоту». А значить, може знадобитися кілька типів ботів.

Для прийому скарг і рекламацій краще зовсім не використовувати ботів, вважає Хебнер. У подібних випадках абонент чекає співпереживання і емоційного залучення. Боти на це не здатні;

4) жоден чат-бот не може розпізнати сарказмом.

Останнім часом в ЗМІ з'являються повідомлення, що чат-боти навчилися розпізнавати саркастичні репліки, тобто у них з'явилися зачатки емоційного інтелекту. Більшість подібних статей посилаються на замітку з журналу The Telegraph. У ній говориться, що вчені з Массачусетського технологічного інституту розробили програму, здатну аналізувати публікації в соцмережах і знаходити саркастичні пости. Але при цьому журналіст The Telegraph чітко вказував, що програма не аналізує слова і тон висловлювань, а просто знаходить саркастичні смайлики.

Томас Хебнер категорично висловився: «Будь-який, хто розповідає, що його чат-бот здатний розпізнати сарказм, вигадує. У моєму університеті я виявився єдиним аспірантом, для якого англійська мова була рідною. Так ось, для мене ввели правило: я повинен був підняти руку щоразу, коли відпускав саркастичну репліку. Інакше мене неправильно розуміли. Навіть люди не завжди розпізнають сарказм» [29].

Деякі чат-боти дійсно можуть «зчитувати» певні емоційні стани, якщо абонент використовує емоційно забарвлені слова. У більшості таких випадків краще, щоб бот передав розвиток діалогу людині;

5) багато «інтелектуальних» чат-ботів взагалі не мають штучного інтелекту.

Сьогодні сотні стартапів заявляють, що побудують вам «інтелектуального бота за півгодини, максимум — за годину». Але на перевірку виявляється, що такі «інтелектуальні» чат-боти — це прості системи, створені за принципом «питання-відповідь». Іншими словами, це дорогі аналоги сторінки «Інфо».

Існують «розумні» боти, які перенаправляють текстові запити споживачів в систему обробки природної мови (natural language processing, NLP). У відповідь така система видає код, що відповідає наміру клієнта. Бот інтерпретує цей код і вибирає потрібну відповідь зі списку сценаріїв.

Зазвичай NLP-боти здатні відповідати тільки за останній запит користувача і не мають «пам'яті», про минулі фрази в розмові. Крім того, потрібно розуміти, що сценарії потрібно дуже добре продумати і безперервно вдосконалювати. А

значить, їх постійно повинні коригувати люди — програмісти та бізнес-менеджери.

У деяких строго визначених ситуаціях такий підхід відмінно працює. Але, наприклад, Томас Хебнер не радить витрачати на NLP-ботів багато часу. «У кінцевому рахунку, все буде виглядати так: програміст постійно просить менеджера роз'яснити йому чергову бізнес-проблему, щоб потім роз'яснити її боту. Це не має сенсу. Справжні боти зі штучним інтелектом самі вчаться на історії діалогів в контакт-центрі. Ручного навчання стає все менше» [30].

AI-боти повинні самостійно аналізують довгі ланцюжки діалогів. А навіть вміти перетворювати аудіозаписи діалогів, наприклад, абонента з оператором контакт-центру в текст і вивчати його. Знову ж таки, треба пам'ятати: для машинного навчання AI-ботів потрібно дуже багато подібних вихідних даних.

2 АНАЛІЗ ОСНОВНИХ ТЕХНОЛОГІЙ ДЛЯ СТВОРЕННЯ НЕЙРОННИХ МЕРЕЖ ТА РОЗРОБКИ ДІАЛОГОВИХ МОДЕЛЕЙ

2.1 Нейронна розмовна модель

2.1.1 Моделювання розмови

Досягнення в наскрізному (end-to-end) навчанні нейронних мереж привели до помітного прогресу в багатьох областях, таких як розпізнавання мови, комп'ютерному зорі і обробки мови [6]. Недавні досягнення припускають, що нейронні мережі можна використовувати не тільки для задач класифікації, їх також можна використовувати для зіставлення одних складних структур з іншими складними структурами. Прикладом цього є завдання зіставлення послідовності з іншою послідовністю, яка має пряме застосування в розумінні природної мови. Основною перевагою цієї структури є те, що вона вимагає невеликої функціональної інженерії та специфіки галузі при зіставленні. Ця перевага, дозволяє дослідникам працювати над завданнями, для яких знання області можуть бути недоступні [22].

2.1.2 Модель послідовність-в-послідовність

Підхід до вирішення задачі моделювання діалогу використовує структуру послідовностей (seq2seq) [18]. Модель заснована на основі поворотної нейронної мережі, яка зчитує вхідну послідовність по одному токену за раз і передбачає вихідну послідовність, також одним токеном за раз. Під час навчання послідовність виведення задається моделлю, тому навчання може бути виконано шляхом зворотного поширення помилки(“backpropagation”). Модель навчається максимізувати крос-ентропію правильної послідовності з урахуванням її контексту. Під час виведення, враховуючи, що справжня послідовність виведення не спостерігається, ми просто подаємо передбачений вихідний токен в якості вхідного сигналу для прогнозування наступного висновку. Це «жадібний» підхід виведення. Менш жадібний підхід полягає у використанні пошуку променя і подачі кількох кандидатів на попередньому кроці на наступний крок. Прогнозована послідовність може бути обрана на основі ймовірності послідовності.

Конкретно, припустимо розмову: перша людина вимовляє «ABC», а друга людина відповідає «WXYZ». Ми можемо використовувати рекурентну нейронну мережу і тренувати її для відображення «ABC» в «WXYZ», як показано на рис 2.1.

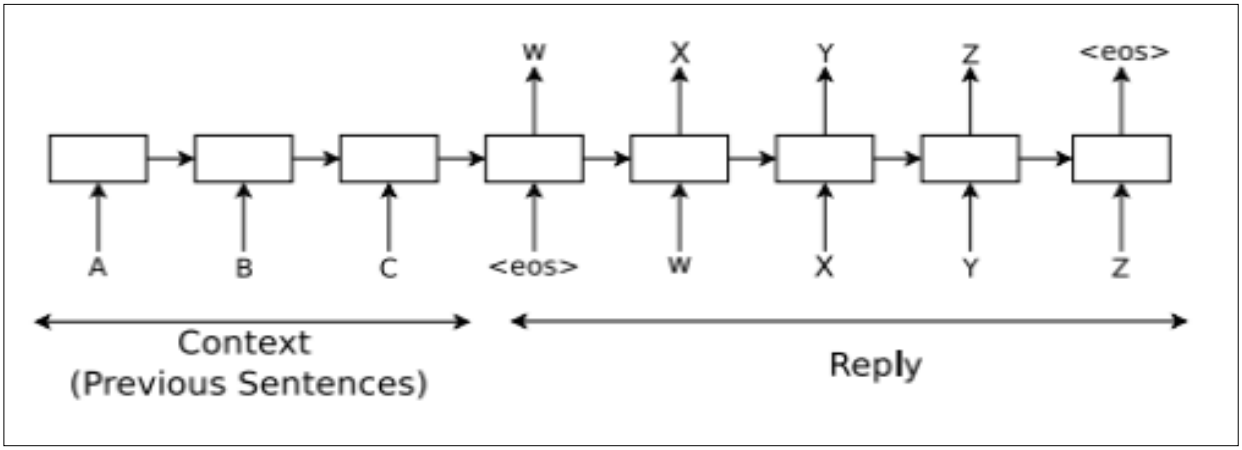

Рисунок 2.1 — Використання seq2seq для моделювання діалогу

Прихований стан моделі, коли модель отримує кінець символу послідовності «<eos>», можна розглядати як вектор мислення, оскільки воно зберігає інформацію про речення або думки «ABC». Перевага цієї моделі полягає в її простоті і спільності. Ми можемо використовувати цю модель для машинного перекладу, питань / відповідей і розмов без істотних змін в архітектурі. Застосування цієї методики до моделювання бесіди також прямолінійне: вхідна послідовність може бути конкатенацією того, що було до цих пір обговорено (контекст), а послідовністю виведення є відповідь. На відміну від більш простих завдань, таких як переклад, модель, подібна послідовності, не зможе успішно «вирішити» проблему моделювання діалогу через декілька очевидних спрощень: оптимізована цільова функція не відображає реальну мету, обмін за допомогою людської комунікації, яка, як правило, більш тривала і заснована на обміні інформацією, а не на прогнозі наступного кроку. Відсутність моделі для забезпечення узгодженості та загального знання є ще одним очевидним обмеженням неконтрольованої моделі.

2.2 Векторизація слів

Векторизація слів чи, як ще називають, “word embedding” — це просто процес зіставлення слів у вектори. Цей векторний вигляд може бути подано безпосередньо в алгоритм машинного навчання як функція. Існує кілька способів виконання цієї операції, починаючи від простого вектора підрахунку до глибокого підходу до навчання, такого як Word2Vec та GloVe [20]. Останні становлять особливий інтерес для цієї тези, оскільки вони довели свою ефективність у галузі розмовних агентів. Зокрема, далі буде описано про модель Skip-gram, оскільки цей

метод використовується в бібліотеці Tensorflow, що використання для спеціальних вбудованих шарів, наприклад, Embedding шару [31].

2.2.1 Word2Vec та Skip-gram модель

Word2Vec — технологія від компанії Google, яка призначена для статистичної обробки великих масивів текстової інформації. Word2Vec збирає статистику про появу слів в даних, видаляє слова, що найбільш рідко і часто зустрічаються, після чого за допомогою нейронних мереж вирішує завдання зниження розмірності і видає на виході компактні векторні уявлення слів заздалегідь певної довжини.

У Word2Vec можливе використання двох різних архітектур нейронної мережі, призначених для перекладу слова в вектор: Continuous Bag of Words (CBOW) і Skip-gram, що описана нижче.

Вперше введений в «Ефективна оцінка представлень слів у векторному просторі», основоположний принцип моделі Skip-gram дуже простий: навчити неглибоку нейронну мережу, що містить один прихований шар фіксованого розміру, щоб передбачити контекст, що задається словом. На рис. 2.2 схематично зображений вигляд архітектури такої моделі.

По суті, створюються приклади штучного тренінгу форми (w t, [w t − 2, w t − 1, w t + 1, w t + 2]), якщо розмір вікна становить 5. Визначення контексту різниться між авторами, але основний принцип залишається тим самим: розсувне вікно фіксованого розміру проходить через слова рівняння; середина цього слова відповідає цільовому слову та словам, що йдуть передує ним, і тим, які після нього утворюють контекст цього слова. Кінцева мета — витягнути внутрішній шар і використовувати його як векторне уявлення для навченої лексики. Дійсно, якщо слова вхідного кодування є гарячими, вони служать лише таблицею пошуку векторних уявлень. На практиці це означає, що корпус спочатку перетворюється в послідовності індексів, потім пропускається модель контексту на цих послідовностях і, нарешті, послідовності індексів перетворюються на послідовності векторів для тренування кінцевого алгоритму.

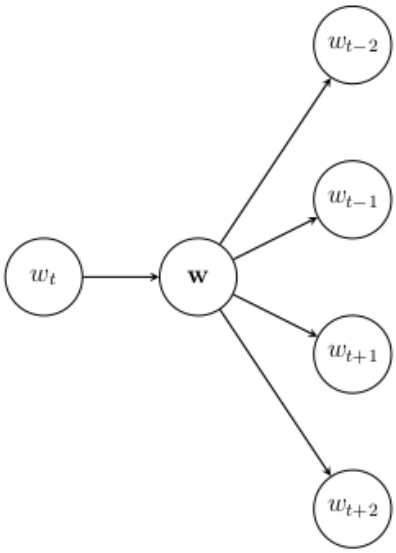

Рисунок 2.2 — Приклад моделі у випадку на 5 слів

2.3 Рекурентні нейромережі і чим вони відрізняються від звичайних

Рекурентні нейронні мережі (recurrent neural networks – RNN) – це клас нейронних мереж, у якому наявні зворотні з’єднання між вузлами, що таким чином утворюють орієнтований цикл, на відміну від ациклічних мереж прямого поширення. Однією із перших рекурентних нейронних мереж стала мережа Хопфілда. Завдяки введенню до моделі нейронної мережі зворотніх зв’язків, у моделі з’явився внутрішній стан, який дозволив обробляти послідовності пов’язаних входів, наприклад таких, як текст або звуки, дані подані у вигляді часових рядів тощо. У традиційних нейронних мереж всі пари вхід-вихід елементів вибірки вважаються незалежними, що є припустимим не для всіх задач.

Рекурентні мережі можна розглядати як динамічні системи, які мають зовнішню складову, на яку впливають спостережувані вхід та вихід, і приховану складову, що характеризує внутрішній стан нейронної мережі. RNN комбінують вхідний вектор зі своїм вектором стану за допомогою фіксованої функції, що навчається в процесі тренування мережі, для отримання нового вектору стану. З точки зору програмування це можу бути інтерпретовано як виконання фіксованої програми з певними вхідними даними та внутрішніми змінними. Таким чином, рекурентні нейронні мережі фактично описують деякі програми. Вони є повними за Тьюрингом в тому сенсі, що з певним набором вагових коефіцієнтів вони можуть імітувати задані програми [1].

Крім того, навіть коли вхідні дані не є послідовностями за своєю суттю, їх послідовна покрокова обробка все одно є можливою. Таким чином рекурентні мережі застосовуються, скажімо, в деяких задачах обробки зображень.

Існує декілька видів архітектури рекурентних нейронних мереж: повнорекурентні мережі, рекурсивні нейронні мережі, двонаправлені рекурентні мережі, мережі Хопфілда, мережі Елмана та Джордана, довга короткочасна пам’ять, тощо. Вони є пристосованими до різних типів задач.

У даному дослідженні нас передусім цікавить архітектура “довга короткочасна пам’ять” та “керований рекурентний блок”, котрі застосовуються при моделюванні природної мови для визначення розподілу ймовірностей на певному обмеженому словнику наступного слова в реченні за умови слів, що передували йому. Ці архітектури рекурентних мереж детальніше будуть описані далі.

Загалом рекурентні нейронні мережі є потужним класом обчислювальних моделей, що здатні виражати майже довільну динаміку. Проте ця здатність обмежується ефективністю процедури тренування моделей, зокрема відомою проблемою зникання градієнту (рис. 2.3): чим більше циклів пройшло з моменту отримання тієї чи іншої інформації, тим більша ймовірність, що значимість цих даних не буде грати великої ролі на новому циклі роботи [23].

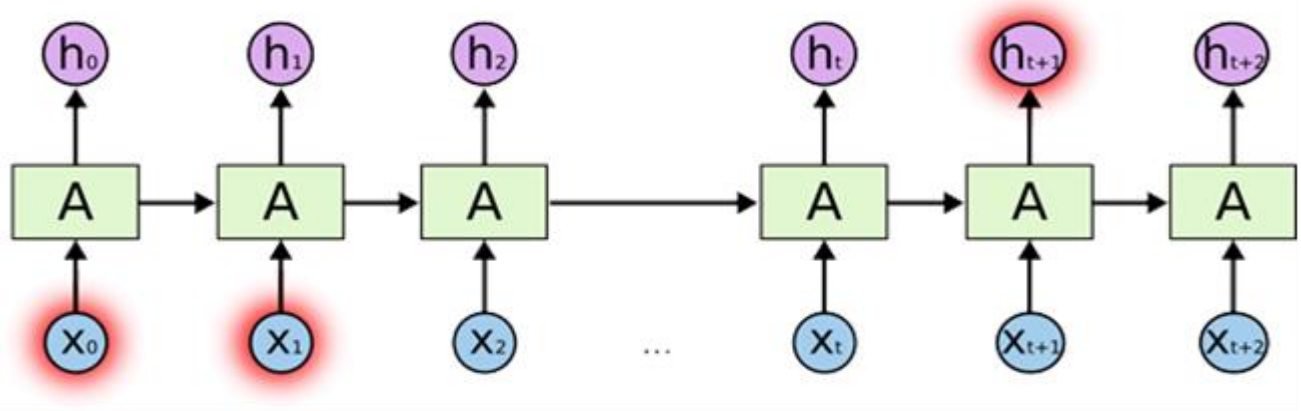

Рисунок 2.3 — Проблема довгостроковій пам'яті в простих RNN

2.3.1 Архітектура “Довга короткочасна пам’ять”

Архітектура довга короткострокова пам’ять (long short-term memory - LSTM) була запропонована у 1997 р. Зеппом Хохрайтером та Юргеном Шмідгубером як вдосконалення базової концепції рекурентних нейронних мереж [13].

Вибір LSTM зумовлюється здатністю впоратися зі зникаючими або вибухаючими значеннями градієнтів при застосуванні варіацій методу градієнтного спуску – найбільшою проблемою у тренуванні рекурентних

нейронних мереж. LSTM мережі покликані саме для боротьби з такою проблемою, не змінюючи при цьому алгоритм навчання.

Ще однією проблемою звичайних рекурентних нейронних мереж, яка була розв’язана із введенням LSTM, була їх нездатність навчатися довготерміновим зв’язкам.

Основою LSTM моделі є блок пам’яті *c*, який у кожний момент часу зберігає агреговану інформацію про входи, які спостерігалися до цього моменту. Структуру цього блоку та принцип роботи наведено на рис. 2.4: нейрони внутрішніх шарів можуть зчитувати і змінювати стан “фільтрів” (cell state), які поєднують у собі функції короткостроковій і довгостроковій пам'яті [17].

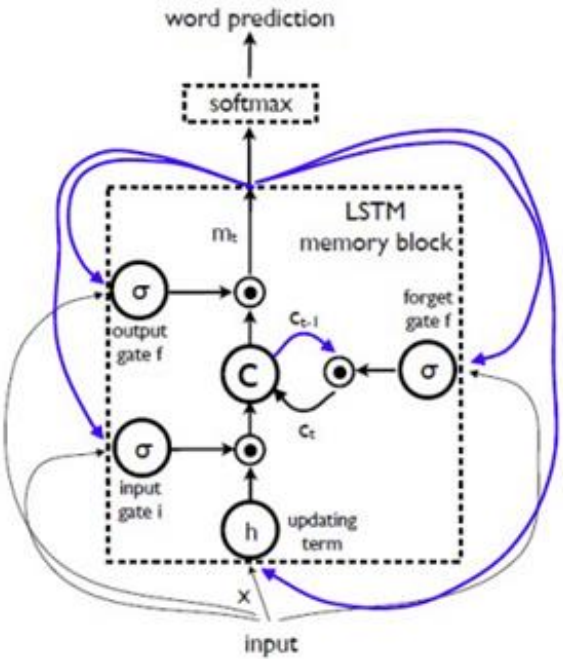

Рисунок 2.4 — Принцип роботи RNN типу LSTM

Поведінка блоку контролюється “фільтрами”, які використовуються для керування плином інформації до блоку або з нього. Ці фільтри реалізують за допомогою логістичної функції, яка приймає значення між 0 та 1. Для часткового дозволу або заборони плину інформації до або з пам’яті LSTM блоку застосовується множення на визначене “фільтром” значення. “Вхідний фільтр” визначає міру, до якої нове значення входить до пам’яті. “Фільтр забуття” визначає міру, до якої значення залишається в пам’яті. “Вихідний фільтр” керує мірою, до якої значення в пам’яті використовується для обчислення виходу блоку [24].

Значення “фільтрів”, оновлення блоку та його вихід визначаються наступними рівняннями:

$$it = \sigma g(Wixxt + Wimmt - 1 + bi) \quad (2.1)$$

$$ft = \sigma g(Wfxxt + Wfmmt - 1 + bf) \quad (2.2)$$

$$ot = \sigma g(Woxxt + Wommt - 1 + bo) \quad (2.3)$$

$$ct = ft \circ ct - 1 + it \circ h(Wcxxt + Wcmmt - 1) \quad (2.4)$$

$$mt = ot \circ ct \quad (2.5)$$

$$pt + 1 = Softmax(mt) \quad (2.6)$$

де операція $\circ$ позначає добуток зі значенням “фільтрів”, а різні матриці W - параметри, що навчаються. Нелінійностями є сигмоїд $(\cdot)$ та гіперболічний тангенс $h(\cdot)$.

Таке мультиплікативне застосування “фільтрів” уможливлює стійке тренування LSTM мереж, оскільки ці “фільтри” вдало нівелюють вплив зникаючих та вибухаючих градієнтів.

На рис. 2.4 блакитним кольором показано рекурентні зв’язки – вихід m у момент часу $t - 1$ подається назад у пам’ять в момент часу t через 35 три “фільтри”; значення блоку подається назад через “забувальний фільтр”; згенероване в момент $t - 1$ слово подається назад разом з виходом пам’яті m у момент t до функції Softmax для отримання розподілу ймовірностей pt над всіма словами та подальшого прогнозування наступного слова.

2.3.2 Архітектура “Керований рекурентний блок”

Введений в 2014 році, керовані рекурентні блоки (Gated Recurrent Units) (або GRU) є варіантом LSTM. Метод GRU дозволяє обчислювати $h(t)$ дещо інакше. Обчислення розбиваються на три блоки: фільтр поновлення (update gate), фільтр скидання стану (reset gate) і новий контейнер пам'яті (memory container). Ці фільтри — функції від вхідного векторного уявлення слова і прихованого стану на попередньому кроці[13].

Для повноти рівняння GRU наведені нижче:

$$zt = \sigma(Wz \cdot [ht - 1, xt] + bz) \quad (2.7)$$

$$rt = \sigma(Wr \cdot [ht - 1, xt] + br) \quad (2.8)$$

$$ht = \tanh(W \cdot [rtht - 1, xt]) \quad (2.9)$$

$$ht = (1 - zt)ht - 1 + ztht \quad (2.10)$$

де *zt* і *rt* відповідають внутрішнім фільтрам GRU, як показано на схемі на рисунку 2.5.

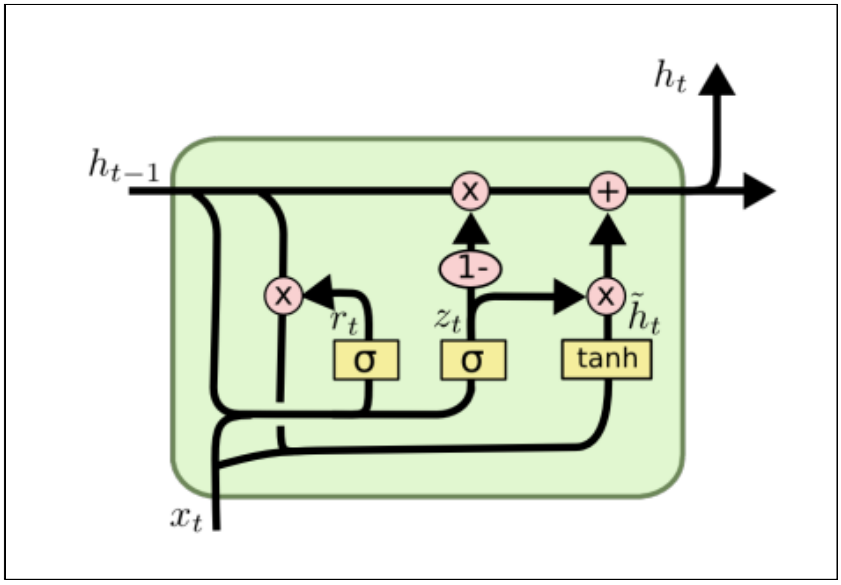

Рисунок 2.5 — Схематичний вигляд GRU

2.3.3 Порівняння LSTM і GRU

Як і LSTM метод GRU розроблений для того, щоб зберігати віддалені залежності в послідовності слів. Під віддаленими залежностями маються на увазі такі ситуації, коли два слова або фрази можуть зустрітися на різних часових кроках, але відносини між ними важливі для досягнення кінцевої мети. LSTM і GRU відстежують ці відносини за допомогою фільтрів, які можуть зберігати або скидати інформацію з оброблюваної послідовності.

Різниця між цими методами полягає в кількості фільтрів (GRU - 2, LSTM - 3). Це впливає на кількість нелінійностей, котре приходить від вхідних даних і в кінцевому підсумку впливає на процес обчислень. Крім того, в GRU відсутня комірка пам'яті, як в LSTM [13].

2.3.4 Області застосування RNN

Одна з головних областей застосування RNN на сьогоднішній день - робота з мовними моделями, зокрема - аналіз контексту і загальної зв'язку слів в тексті. Для RNN структура мови - це довгострокова інформація, яку треба запам'ятати. До неї відносяться граматика, а також стилістичні особливості того корпусу текстів, на яких проводиться навчання. Фактично RNN запам'ятовує, в якому порядку зазвичай йдуть слова, і може дописати пропозицію, отримавши деяку

послідовність. Якщо ця послідовність випадкова, може вийти зовсім безглуздий текст, що стилістично нагадує шаблон, на якому вчилася RNN. Якщо ж вихідний текст був осмисленим, RNN допоможе його стилізувати, проте в останньому випадку однієї RNN буде мало, так як результат повинен являти собою «суміш» випадкового, але стилізованого тексту від RNN і осмисленої, але «неокращенної» вихідної частини. Це завдання вже нагадує популярні нині програми для обробки фотографій в стилі Моне і Ван Гога, що може слугувати деякою аналогією [26].

Дійсно, завдання перенесення стилю з одного зображення на інше вирішується за допомогою нейромереж і операцій згортки, які розбивають зображення на кілька сфер та дозволяють нейромережам аналізувати їх незалежно один від одного, а згодом і перемішувати між собою. Аналогічні операції проводилися і з музикою (також за допомогою згортальних нейромереж): в цьому випадку мелодія є змістом, а аранжування - стилем. І ось з написанням музики RNN якраз успішно справляється. Оскільки обидва завдання - і написання, і змішування мелодії з довільним стилем - вже успішно вирішені за допомогою нейромереж, поєднати ці рішення залишається справою техніки.

Але, чому музику RNN можуть бути проблеми, а з повноцінними текстами Толстого і Достоєвського проблеми не виникають? Справа в тому, що в інструментальній музиці, як би по-варварськи це не звучало, немає сенсу в тому ж значенні, в якому він є в більшості текстів. Тобто музика може подобатися або не подобатися, але якщо в ній немає слів - вона не несе інформаційного навантаження. Саме з доданням своїм творам сенсу і спостерігаються проблеми у RNN: вони можуть чудово вивчити граматику мови і запам'ятати, як повинен виглядати текст в певному стилі, але створити і донести якусь ідею або інформацію RNN (поки) не можуть.

Також особливим випадком в цьому питанні — це автоматичне написання програмного коду. Дійсно, оскільки мова програмування по визначенню є мова, RNN може його вивчити. На практиці виявляється, що програми, написані RNN, цілком успішно компілюються і запускаються, проте вони не роблять нічого корисного, якщо їм заздалегідь не позначити завдання. А причина цього та ж, що і в разі літературних текстів: для RNN мову програмування — не більше ніж стилізація, в яку вони, на жаль, не можуть вкласти ніякого сенсу.

На поточний момент в Google для машинного перекладу використовуються RNN типу LSTM, що дозволило домогтися найбільшої точності в порівнянні з існуючими аналогами, проте, за словами самих авторів, машинного перекладу ще дуже далеко до рівня людини. Складнощі, з якими стикаються нейромережі в задачах перекладу, обумовлені відразу декількома факторами: по-перше, в будь-

якому завданні існує неминучий розмін між якістю і швидкістю. На даний момент чоловік дуже сильно випереджає штучний інтелект за цим показником. Оскільки машинний переклад найчастіше використовується в онлайн-сервісах, розробники змушені жертвувати точністю на вигоду швидкодії. У недавній публікації Google на цю тему розробники детально описують багато рішень, які дозволили оптимізувати поточну версію Google Translate, однак проблема досі залишається [10]. Наприклад, рідкісні слова, або сленг, або навмисне спотворення слова (наприклад, для більш яскравого заголовка) може збити з пантелику навіть перекладача-людини, якому доведеться витратити час, щоб підібрати найбільш адекватний аналог в іншій мові. Машину ж така ситуація поставить у глухий кут, і перекладач буде змушений «викинути» складне слово і залишити його без визначення. У підсумку проблема машинного перекладу не настільки обумовлена архітектурою (RNN успішно справляються з рутинними завданнями в цій галузі), наскільки складністю і різноманіттям мови. Радує те, що ця проблема має більш технічний характер, ніж написання осмислених текстів, де, ймовірно, потрібно кардинально новий підхід. Принцип роботи машинного перекладача Google Translate приведено на рис. 2.6.

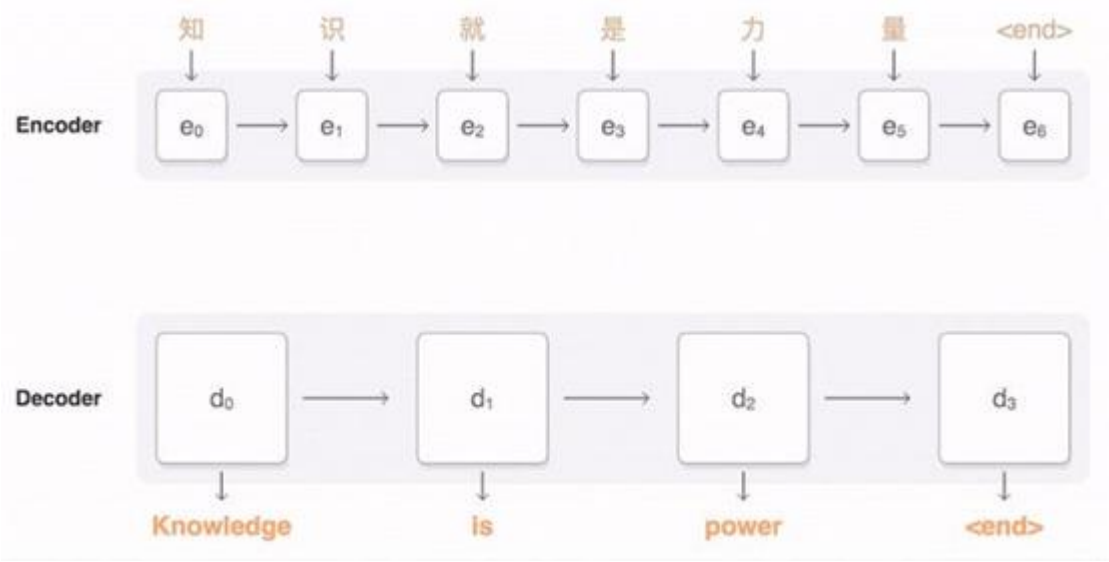

Рисунок 2.6 — Принцип роботи машинного перекладача Google Translate, заснованого на комбінації кілька рекурентних нейромереж

2.4 Механізми “уваги” в нейронних мережах

Найсерйозніший недолік простої архітектури кодер-декодер полягає в тому, що фраза довільної довжини має бути стиснута в вектор фіксованого розміру. Цей

підхід здається досить дивним, зважаючи на те, як комп'ютери зазвичай вирішують задачу стиснення. Коли стискається файл за допомогою архіватора, розмір результуючого файлу приблизно пропорційний розміру вихідного файлу. (Це не зовсім вірно: розмір стисненого файлу пропорційний кількості інформації в початковому файлі, а не його розміру. Проте припустимо, що розмір вихідного файлу досить точно відповідає кількості інформації в ньому.)

Продовжуючи аналогію з комп'ютерами, зберігати всі фрази не у форматі вектора фіксованого розміру, а в формі пам'яті, що містить кількість банків дорівнює кількості вихідних слів. Це реалізується за допомогою двобічної рекурентної нейронної мережі (ДРНМ, bidirectional recurrent neural network, BiRNN), що складається з прямої RNN (forward RNN) і зворотного RNN (backward RNN). Як і випливає з їхніх назв, пряма і зворотна RNN зчитують вихідну фразу в прямому і зворотньому напрямку відповідно.

Тепер давайте позначимо приховані стану прямого і зворотного RNN, як відповідно. Як вже було зазначено, RNN узагальнює послідовність, зчитуючи по одному елементу за раз. Це означає, що прямий RNN узагальнює вихідну фразу до j-го слова, починаючи з першого слова, а зворотної RNN — до j-го слова, починаючи з останнього слова. Тобто разом узагальнюють все вхідне повідомлення (рис. 2.7).

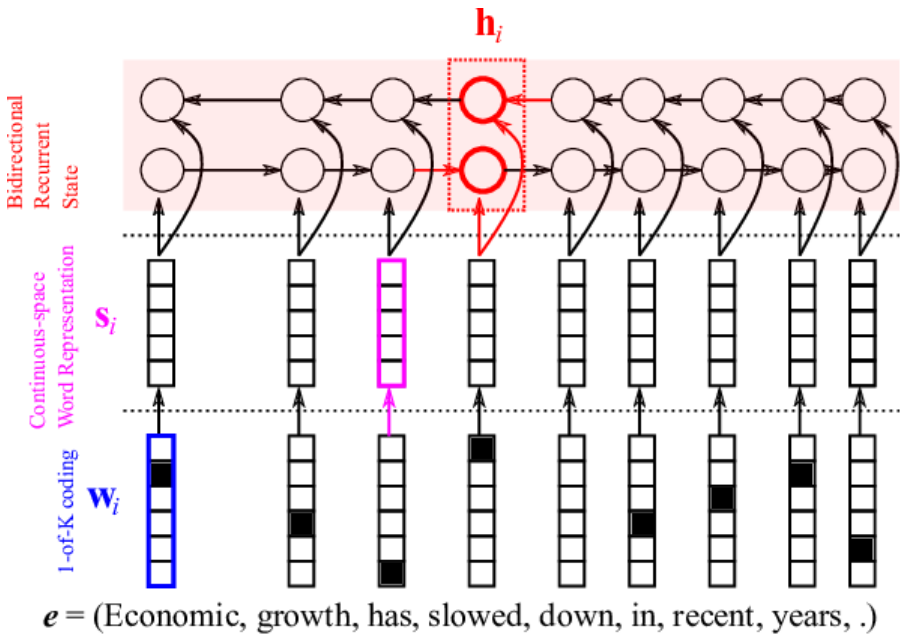

Рисунок 2.7 — Застосування двобічної рекурентної нейронної мережі для кодування вихідного пропозиції

Однак узагальнення на позиції кожного слова не є ідеальним узагальненням всього вхідного повідомлення. Через свою послідовну природу RNN краще пам'ятають останні слова. Тобто, чим далі вхідне слово знаходиться від j, тим менш імовірно, що прихований стан RNN (або) добре його пам'ятає. Аннотаційний вектор (annotation vector), що є об'єднанням (concatenation) і, найкращим чином представляє поточний слово *wj*.

Безумовно, це не загальноприйнята угода, але цієї причини розглядають аннотаціонний вектор, як контекстно-залежне уявлення слова (context-dependent word representation). Більш того, ми можемо трактувати набір контекстно-залежних уявлень, як механізм, за допомогою якого ми зберігаємо вихідне повідомлення у вигляді уявлення змінної довжини, на відміну від узагальнення фіксованої довжини в простій моделі кодер-декодер.

При наявності повідомлення змінної довжини вихідного повідомлення декодер повинен мати можливість вибірково фокусуватися на одному або декількох контекстно-залежних уявленнях слів (аннотаційних векторах) для кожного цільового слова. Проблема лише у тому, на якому аннотаційному векторі повинен фокусуватися декодер в кожен момент часу.

Модель дивиться на кожне вихідне слово *xj* (в разі алгоритму — це контекстно-залежне уявлення *h(j)*), розглядає його спільно з уже переведеними словами (*y1, y2, ..., yi-1*) і вирішує, чи було початкове слово *xj* вже застосовано (це еквівалентно тому, наскільки релевантно або нерелевантні вихідне слово *xj* кожному цільовому слову). Цей процес повторюється для кожного слова у вихідному реченні.

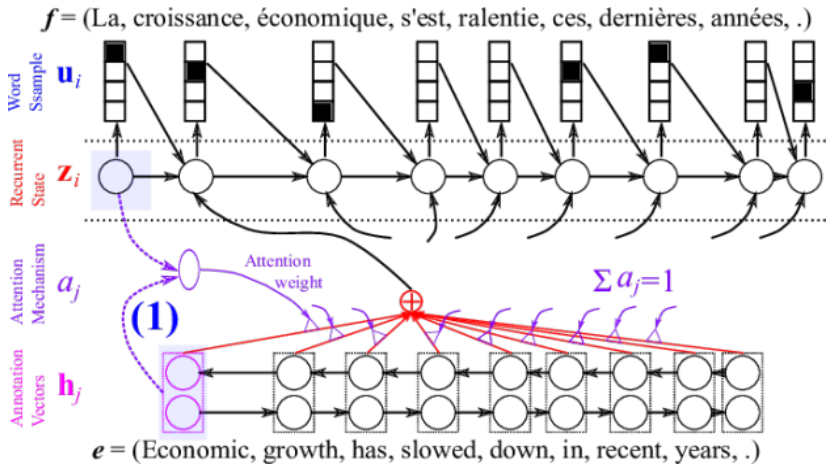

Рисунок 2.8 — Механізм уваги зіставляє уже перекладені слова і одне з вихідних слів

Дмитро Багданов (Dzmitry Bahdanau), Йошуа Бенджі (Yoshua Bengio) влітку 2014 року запропонували включити в декодер невелику нейронну мережу, що реалізує процес аналогічний описаному вище [11]. Ця невелика нейронна мережа, яку називають “механізмом уваги” позначена фіолетовим кольором (рис. 2.8), приймає на вході попередній прихований стан декодера *zi-1* (тобто те, що вже було переведено) і одне з контекстно-залежних уявлень вихідних слів *h(j)*. Механізм уваги реалізований у вигляді нейронної мережі з одним прихованим шаром і одним скалярним виходом, як показано на рис. 2.9. Процес повторюється для кожного вихідного слова.

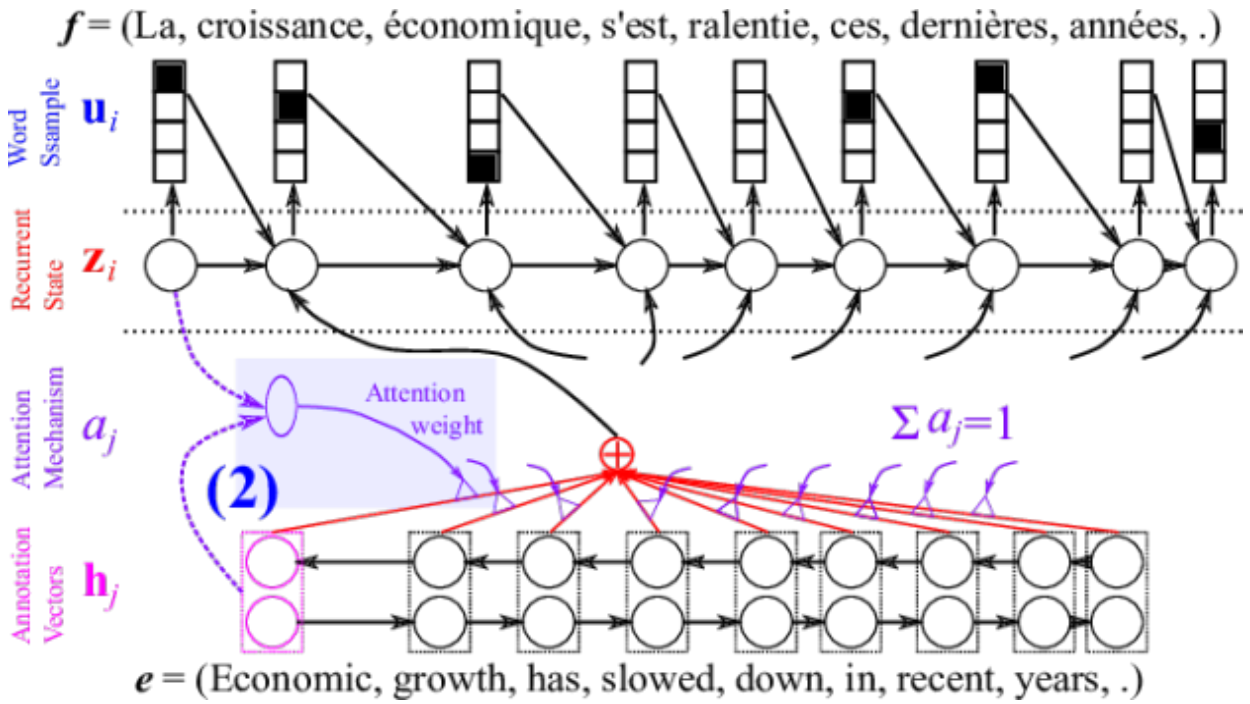

Рисунок 2.9 — Механізм уваги повертає одне скалярне значення, що є оцінкою релевантності *j-го* вихідного слова

Після того, як розрахували оцінки релевантності для кожного вихідного слова, необхідно провести таке перетворення, щоб сума оцінок дорівнювала одиниці. На основі цієї ймовірнісної інтерпретації, обчислюється очікуваний аннотаційний вектор в рамках даного розподілу (рис. 2.10).

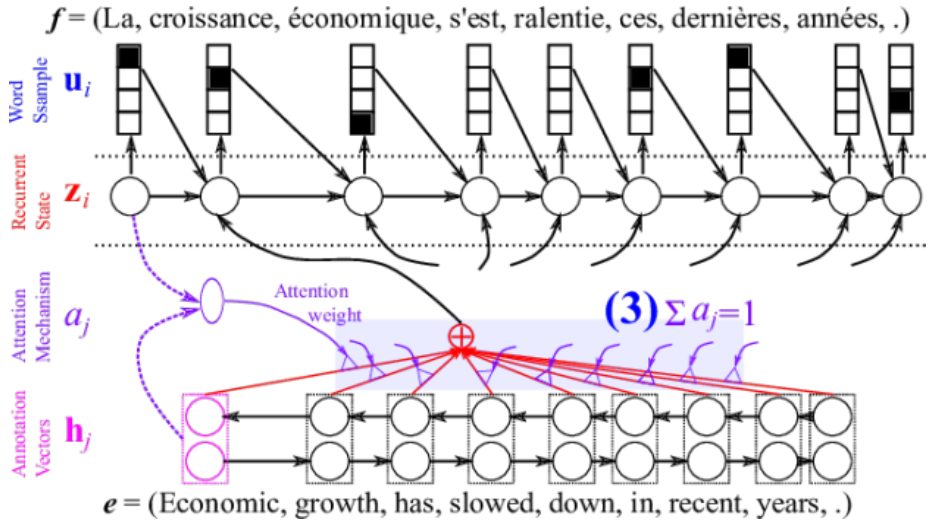

Рисунок 2.10 — Оцінки релевантності

Це легко можна зробити за допомогою софтмакс-нормалізації (softmax normalization) наступним чином:

$$a = Softmax(e) \tag{2.11}$$

Причина, чому потрібен подібний тип нормалізації? Можна назвати масу причин, але найважливіша з них полягає в тому, що це дає можливість інтерпретувати оцінки, присвоєні механізмом уваги, як ймовірності. З точки зору ймовірності, ми можемо розглядати вагові коефіцієнти уваги (attention weight) *αj*, як ймовірність того, що декодер вибере j-е контекстно-залежне уявлення вихідного слова з усіх *t* вихідних слів. Потім можливо обчислити очікуване контекстно-залежне уявлення слова в рамках даного розподілу (заданого ваговими коефіцієнтами уваги *αj*) наступним чином:

$$ci = \sum_{j=1}^{T} \alpha_j h_j = \mathrm{E}_{a_t^1}[h_j] \tag{2.12}$$

Очікуваний вектор *ci* узагальнює інформацію про всю вихідному послідовність, але з різним акцентом на різних позиціях / словах вихідного повідомлення. Будь-який аннотаційний вектор (контекстно-залежний вектор), який механізм уваги «вважає» релевантним (тобто вектор з великою вагою уваги), буде мати краще уявлення, ніж вектори з малими ваговими коефіцієнтами уваги.

3 БІБЛІОТЕКА TENSORFLOW ТА РОЗРОБКА НЕЙРОННОЇ ДІАЛОГОВОЇ МОДЕЛІ

3.1 Обґрунтування вибору технологій та засобів розробки

Розробка діалогового агента відбувалася на дистрибутиві Linux Ubuntu 18.04 LTS. Для того щоб запустити навчання необхідно встановити наступні пакети та бібліотеки:

- Python 3.7.0 або можливо новіше;
- Tensorflow 2.0.0 або можливо новіше;
- Tensorflow datasets 2.0.0 або можливо новіше [28];

3.1.1 Ubuntu

Ubuntu — це повноцінна операційна система Linux, вільно доступна як для широкої спільноти, так і для професійної діяльності. Спільнота Ubuntu побудована на ідеях, закладених у Маніфесті Ubuntu: що програмне забезпечення має бути доступне безкоштовно, що програмні засоби повинні бути використані людьми на необхідній мові та, незважаючи на будь-які обмеження, що люди повинні мати свободу налаштування та змогу змінювати своє програмне забезпечення будь-яким способом, який вони вважають за потрібне [32].

Головні ідеї ОС:

- Ubuntu завжди буде безкоштовною, і за “корпоративне видання” додаткової плати немає, найкраща версія доступна для всіх на тих самих безкоштовних умовах;
- Ubuntu включає найкращу інфраструктуру перекладів та доступності, яку може запропонувати співтовариство вільного ПЗ, щоб зробити Ubuntu якомога більше доступним для використання;
- Ubuntu поставляється у стабільних та регулярних циклах випуску; новий реліз виходить кожні півроку. Кожні два рівні роки виходить версія з довгостроковою підтримкою Ubuntu LTS(Long Term Support), яка підтримується протягом 5 років. Випуски Ubuntu між ними (відомі як тестові версії або не-LTS) підтримуються протягом 9 місяців;
- Ubuntu повністю відданий принципам розробки програмного забезпечення з відкритим кодом; заохочує людей використовувати програмне забезпечення з відкритим кодом, вдосконалювати його та передавати його далі.

Інтерфейс ОС Ubuntu 18.04 “Bionic beaver” приведено на рис. 3.1.

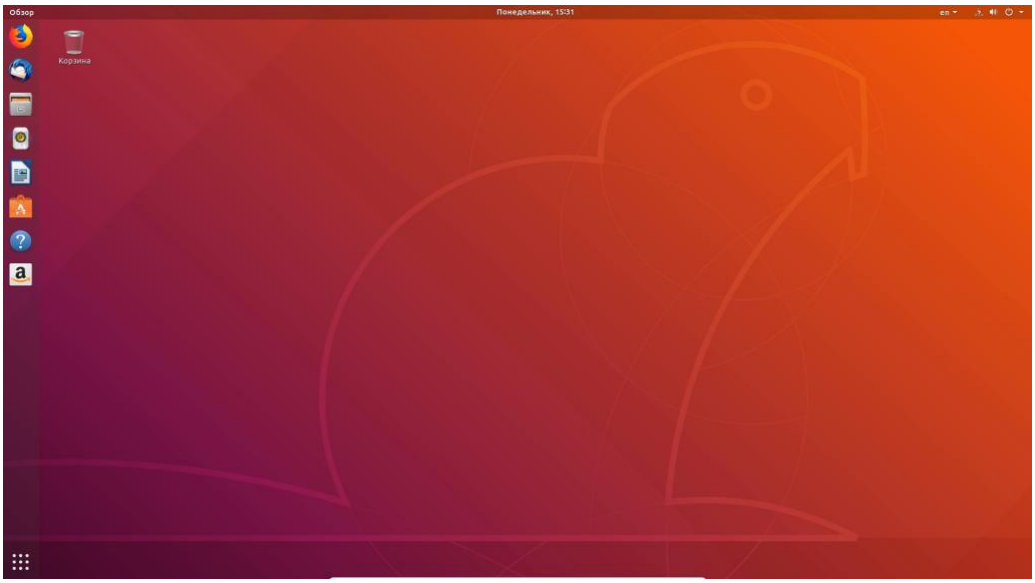

Рисунок 3.1 — Інтерфейс ОС Ubuntu 18.04 “Bionic beaver”

3.1.2 Python

Python — інтерпретована об'єктно-орієнтована мова програмування високого рівня зі строгою динамічною типізацією. Структури даних високого рівня разом із динамічною семантикою та динамічним зв'язуванням роблять її привабливою для швидкої розробки програм, а також як засіб поєднування наявних компонентів. Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду. Інтерпретатор Python та стандартні бібліотеки доступні як у скомпільованій, так і у вихідній формі на всіх основних платформах. В мові програмування Python підтримується кілька парадигм програмування, зокрема: об'єктно-орієнтована, процедурна, функціональна та аспектно-орієнтована [25].

Серед основних переваг можна назвати такі:

- чистий синтаксис (для виділення блоків слід використовувати відступи);
- переносність програм (що властиве більшості інтерпретованих мов);
- стандартний дистрибутив має велику кількість корисних модулів (включно з модулем для розробки графічного інтерфейсу);
- можливість використання Python в діалоговому режимі (дуже корисне для експериментування та розв'язання простих задач);
- стандартний дистрибутив має просте, але разом із тим досить потужне середовище розробки, яке зветься IDLE і яке написане на мові Python;

- зручний для розв'язання математичних проблем (має засоби роботи з комплексними числами, може оперувати з цілими числами довільної величини, у діалоговому режимі може використовуватися як потужний калькулятор);

- відкритий код (можливість редагувати його іншими користувачами);

- Python має ефективні структури даних високого рівня та простий, але ефективний підхід до об'єктно-орієнтованого програмування. Елегантний синтаксис Python, динамічна обробка типів, а також те, що це інтерпретована мова, роблять її ідеальною для написання скриптів та швидкої розробки прикладних програм у багатьох галузях на більшості платформ;

- інтерпретатор мови Python і багата Стандартна бібліотека (як вихідні тексти, так і бінарні дистрибутиви для всіх основних операційних систем) можуть бути отримані з офіційного сайту, і можуть вільно розповсюджуватися. Цей самий сайт має дистрибутиви та посилання на численні модулі, програми, утиліти та додаткову документацію;

- інтерпретатор мови Python може бути розширений функціями та типами даних, розробленими на C чи C++ (або на іншій мові, яку можна викликати із C). Python також зручна як мова розширення для прикладних програм, що потребують подальшого налагодження.

Інтерпретатор мови заздалегідь встановлено у ядро системи, тому нема необхідності для встановлення.

3.1.3 Tensorflow

Значто спростити розробку може Google Tensorflow — одна з популярніших бібліотека для машинного навчання з усіх, оскільки вона створена для того, щоб бути доступною для всіх та спростити розробку моделей. Бібліотека Tensorflow містить різні API, побудовані на масштабній архітектурі глибокого навчання, такі як CNN або RNN. Tensorflow заснований на обчисленні графіка; це дозволяє розробнику візуалізувати побудову нейронної мережі за допомогою Tensorboad. Цей інструмент корисний для налагодження програми.

Архітектура Tensorflow працює в трьох частинах: попередня обробка даних, побудова моделі, тренування та оцінювання моделі(рис. 3.2).

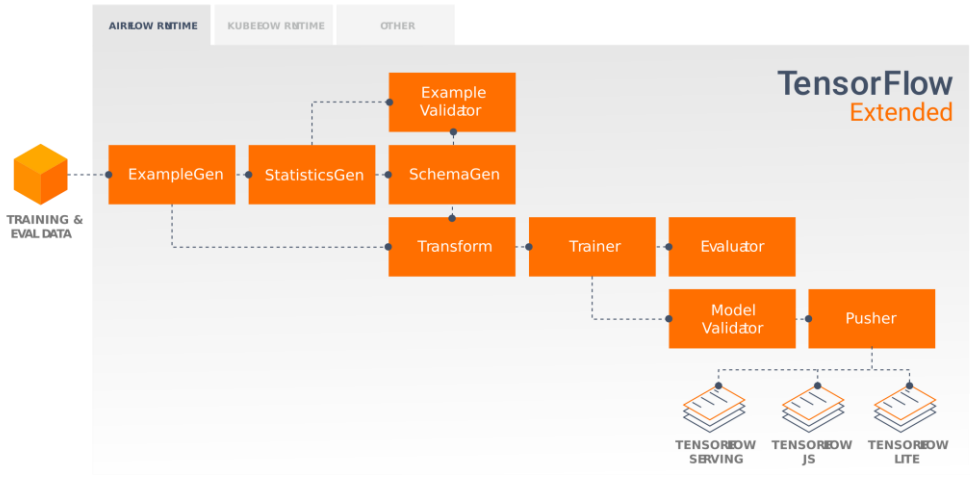

Рисунок 3.2 — Архітектура Tensorflow

Користувач може побудувати своєрідну блок-схему операцій (звану графік). Вхід надходить з одного кінця, а потім він протікає через цю систему операцій і виходить з іншого кінця як вихід.

У Tensorflow 2.0 відбувся ряд численних змін. Видалено API-інтерфейси, після чого API стали більш узгодженим, і краще інтегрувався з Python runtime, з Eager execution [27]. А також багато інших змін, що роблять користувачів Tensorflow ще більш продуктивними. Систему Tensorflow приведено на рис. 3.3.

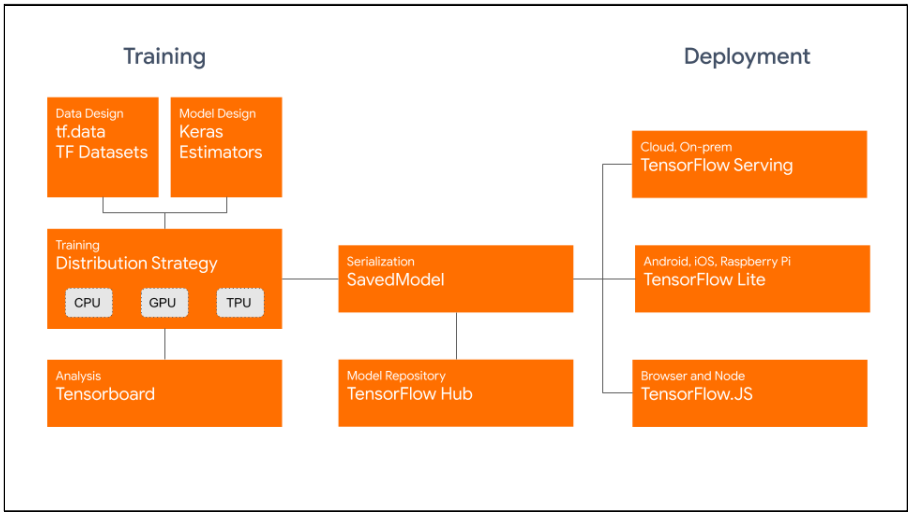

Рисунок 3.3 — Система Tensorflow 2.0

З усіма перевагами, змінами та вдосконаленнями, внесеними в Tensorflow 2.0, можливо легко створювати навіть такі дуже складні моделі, що можуть на достатньому рівні розуміти природну мову та вести розмову з людиною.

Інсталяція Tensorflow:

1. Необхідно виконати наступну команду для встановлення та оновлення до необхідної версії Tensorflow рис. 3.4.

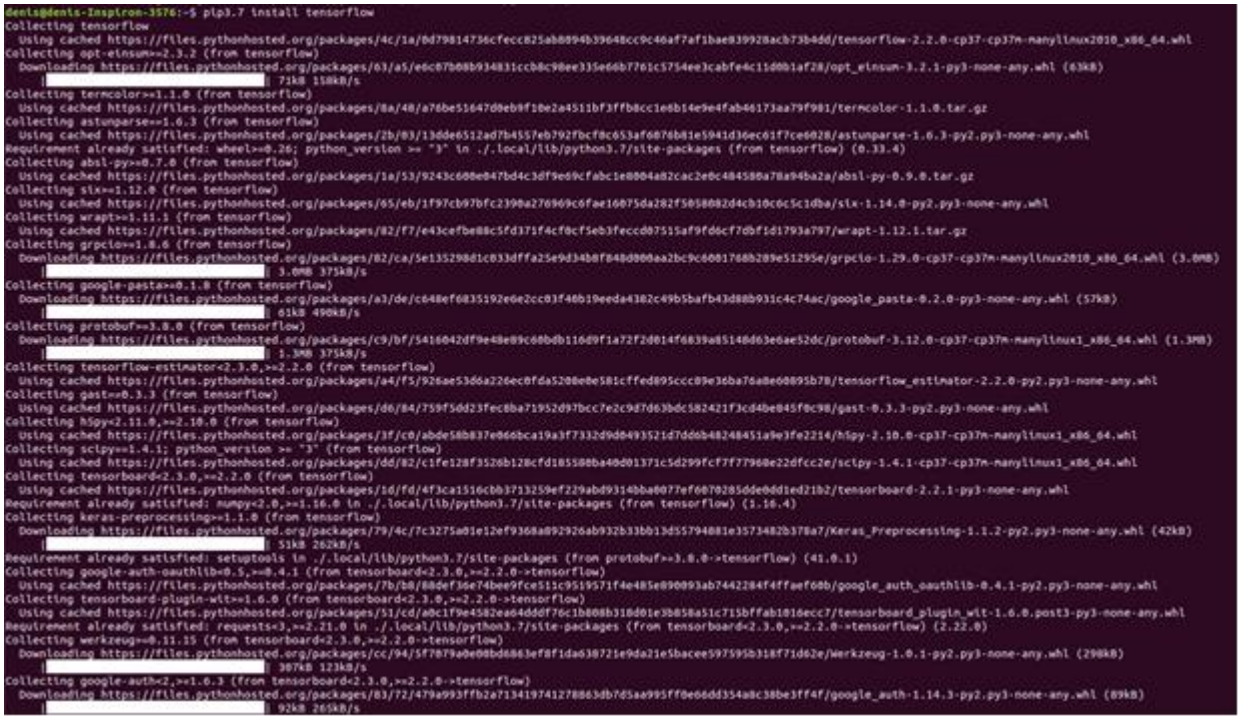

Рисунок 3.4 — Результат встановлення Tensorflow

2. Потім необхідно впевнитися, що все встановилося вірно рис. 3.5.

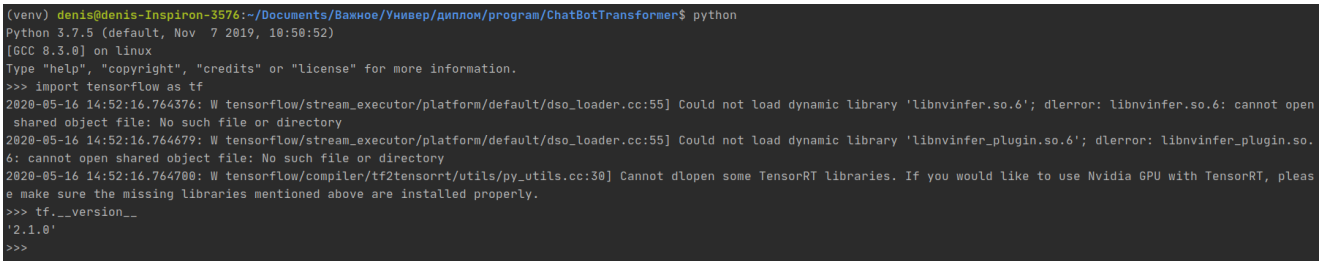

Рисунок 3.5 — Перевірка роботи роботи Tensorflow

3.2 Розробка нейронної діалогової моделі

Створення нейронної мережі дуже довгий процес, який складається з багатьох частин. Перш за все необхідно визначити яку задачу треба вирішити та з якими даними потрібно буде працювати. Після цього проводиться пошук корпусів діалогів серед відкритих наборів даних. Якщо потрібного корпусу не існує то потрібно створити його власноруч.

3.2.1 Структура програмних модулів

Кінцева структура проекту приведено на рис 3.6.

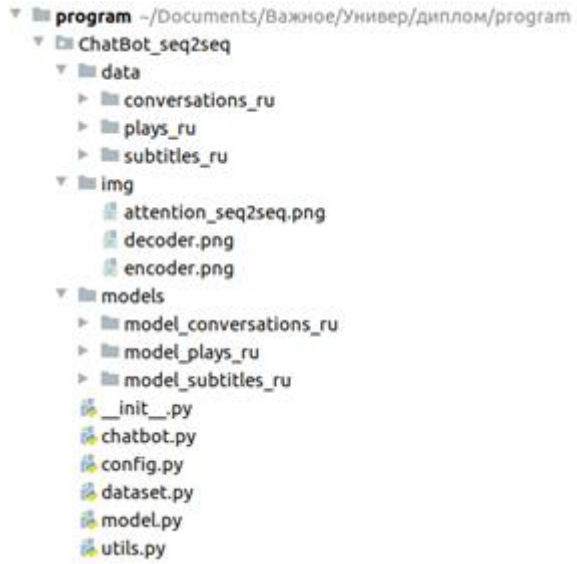

Рисунок 3.6 — Кінцева структура проекту

Таблиці 3.1 описує основні функції модулів системи та основне призначення допоміжних папок.

Таблиця 3.1 — Функції модулів

Назва модулю, папки	Призначення
1	2
chatbot.py	Слугує для навчання та використання моделей.
config.py	Зберігає основні параметри необхідні для використання
dataset.py	Реалізує функціонал обробки та збереження даних для подальшого їх використання при навчанні
model.py	Містить реалізацію моделі нейронної мережі
utils.py	Файл, що зберігає інші допоміжні функції
data folder	Папка з основними даними для навчання
img folder	Пака зі схемами архітектур моделі(кодер, декодер та ін.)
models folder	Папка, де зберігаються результати навчання моделі

3.2.2 Архітектура діалогової моделі

У машинному навчанні модель є функцією з параметрами, які можна вивчати, що відображає вхід на вихід. Оптимальні параметри отримують шляхом навчання моделі за даними. Добре підготовлена модель забезпечує точне відображення від входу до потрібного виходу.

Основа — рекурентна нейронна мережа, модель AttentionSeq2Seq, що складається з двох підмереж кодувальника та декодувальника.

Архітектура кодувальника:

1. Input шар: бере англійське речення і передає його шару векторизації.

2. Embedding шар: бере англійське речення та перетворює кожне слово у вектор фіксованого розміру.

3. Dropout шар: отримує вектор та обнуляє певний відсоток даних, після чого передає результат наступному шару.

4. Перший GRU шар: на кожному кроці він отримує вектор, який представляє слово, і передає результат наступному шару.

5. Другий GRU шар: він робить те саме, що і попередній шар, але замість передачі його виводу він передає свої стани першому шару LSTM декодера.

Декодувальник має майже таку саму архітектуру, як кодувальник, але використовується, за вибором, один із шарів уваги, який можливо налаштувати у config файлі. AdditiveAttention — шар, що реалізує механізм уваги Богданова. Attention шар — реалізує увагу за алгоритмом Луонга.

Архітектура декодувальника:

1. Input шар: бере англійське речення і передає його шару векторизації.

2. Embedding шар: бере англійське речення та перетворює кожне слово у вектор фіксованого розміру.

3. Перший Dropout шар: отримує вектор та обнуляє певний відсоток даних, після чого передає результат наступному шару.

4. Перший GRU шар: на кожному кроці він отримує вектор, який представляє слово, і передає результат наступному шару.

5. Один з шарів уваги: знаходить контекст та та передає до наступного шару.

3. Другий Dropout шар: отримує контекстний вектор, обнуляє певний процент даних, після чого передає результат далі на рекурентний шар.

6. Другий GRU шар: обробляє результат першого GRU шару, скоректований контекстним вектором від попередніх шарів, після чого передає результат далі.

7. Dense шар(вихідний шар): приймає вихід з попереднього шару і виводить один гарячий вектор, що представляє цільове слово.

Використання моделі уваги дозволяє встановити "м'яке" відповідність між вхідними та вихідними послідовностями, що підвищує якість і продуктивність.

Повну архітектуру моделі приведено на рисунку 3.7.

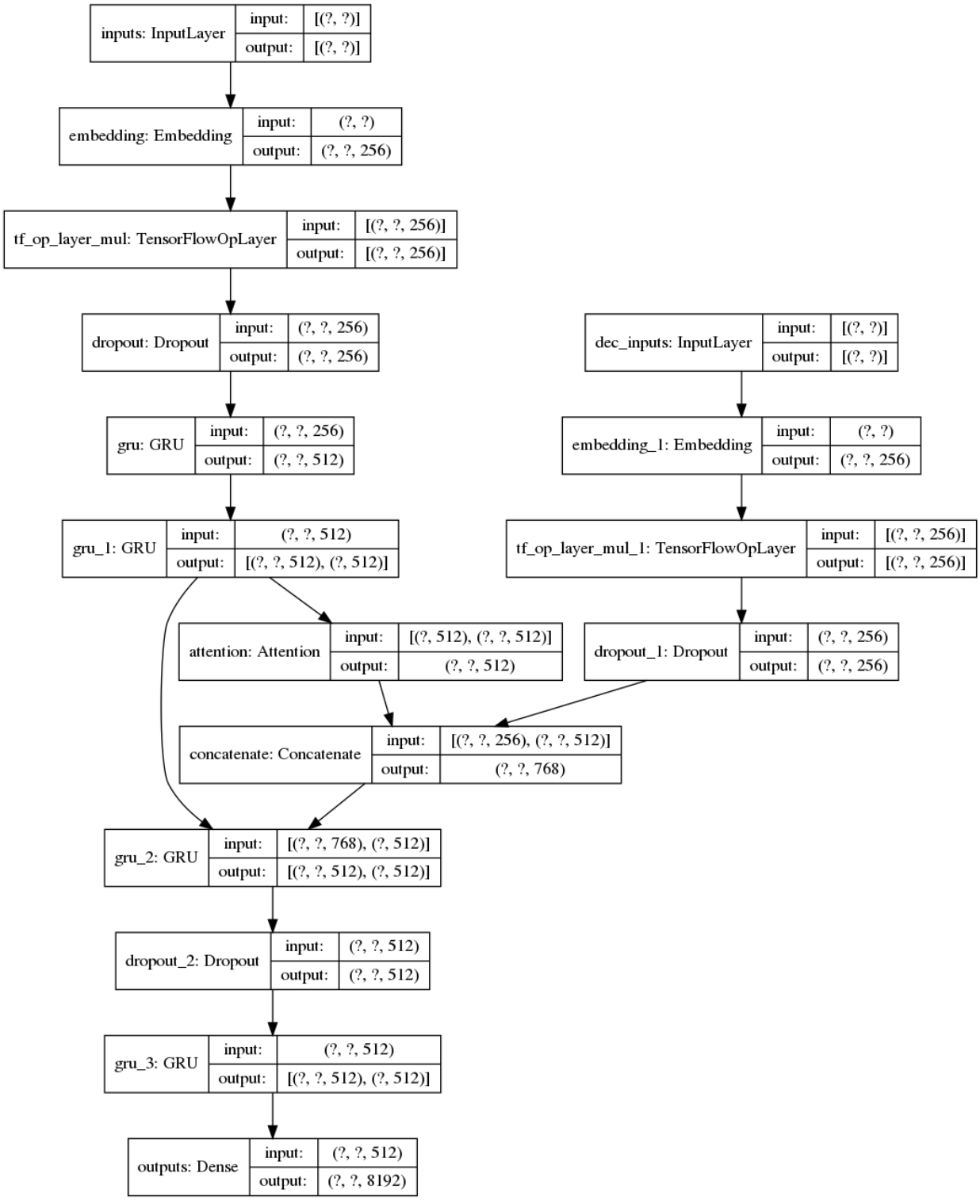

Рисунок 3.7 — Архітектура діалогової моделі

3.2.3 Навчання моделі

Основні параметри для використання приведено на рисунку 3.8

```
4   CURRENT_FOLDER = os.getcwd()
5
6   # --- TRAIN CONFIG ---
7   # - Data -
8   PATH_TO_DATA_FOLDER = 'data'
9   '''...'''
24  DATASET = 'subtitles_ru'
25
26  MAX_SAMPLES = 50000
27  MAX_LENGTH = 40
28  VOCAB_SIZE = 2 ** 13  # 8192
29
30  # - Model -
31  TRAIN_MODEL = True
32  MODELS_DIR = 'models'
33  MODEL_CKPT_DIR = 'model_subtitles_ru'
34  # MODEL_CKPT_DIR = 'model_plays_ru'
35  # MODEL_CKPT_DIR = 'model_conversations_ru'
36  # MODEL_CKPT_DIR = 'model_{}'.format(datetime.datetime.now().strftime("%Y-%m-%d"))
37
38  '''...'''
44  ATTENTION_TYPE = 1
45  BATCH_SIZE = 64
46  NUM_UNITS = 512
47  D_MODEL = 256
48  DROPOUT = 0.1
49  EPOCHS = 2
```

Рисунок 3.8 — Основні параметри системи

У таблиці 3.2 описані основні параметри з їх призначенням, що використовуються для навчання та використання моделі.

Таблиця 3.2 — Функції модулів

Параметр	Призначення
1	2
NUM_UNITS	Кількість одиниць шару
MAX_SAMPLES	Відповідає за кількість тренувальних пар (послідовностей)
MAX_LENGTH	Максимальна довжина послідовностей

Закінчення таблиці 3.2

1	2
VOCAB_SIZE	Кількість кінцевого запасу слів(унікальних слів, що зустрілися в послідовностях
TRAIN_MODEL	Логічний параметр. Якщо True — відбувається навчання / донавчання, якщо False — використання моделі
ATTENTION_TYPE	Відповідає, який тип шару “уваги” використовувати. 1 — “увага” Богданова. Tensorflow шар — BahdanauAttention() 0 — “увага” за Лаунгом. Tensorflow шар — LuongAttention().
BATCH_SIZE	Кількість порцій, на які будуть поділені дані
D_MODEL	Розмірність вихідного вектору шару Embedding
DROPOUT	Процент даних, який обнулить відповідний шар моделі
EPOCHS	Параметр, що відповідає за кількість ітерацій навчання / донавчання
DATASET	Відповідає за дані, які будуть використовуватися. На даний момент заздалегідь створено 3 набори даних для навчання бота: - набір “plays_ru”. Містить 1601 пар від різних спектаклів та п'єс (data/plays_ru); - набір даних “convestations_ru”. Містить 136 000 пар з різноманітних розмов (data/convestations_ru); - набір “subtitles_ru”, що містить 2 500 000 пар з субтитрів до 347 серіалів (data/subtitles_ru).

Весь процес навчання моделі складається з двох основних етапів:

1. Підготовка тренувальних даних.

2. Саме навчання моделі.

Перед навчанням моделі потрібно обробити та підготувати сирі дані для подальшого їх використання у моделі:

1. Дані у файлу зберігається у форматі: “Питання %% Відповідь”, приклад файлу даних приведено на рис. 3.9. Необхідна кількість пар зчитується з файлу для подальшої роботи з ними.

```
Что случилось?  %%  Пока ничего.
Срочно нужна твоя помощь.  %%  В чем?
Ты уезжаешь?  %% Нет, не уезжаю
Что значит уходишь, куда?  %% Неважно, куда. Важно, откуда.
Уходишь от жены?  %% А от кого еще мне уходить?
Ты серьезно?  %% Этим не шутят.
Вы поссорились что ли?  %% Нет.
Так почему же вдруг ты решил уходить?  %% Это я тебе потом объясню.
Но ты твердо решил?  %% Железно.
У тебя кто-то появился?  %% Ну в общем, да.
Куда мне сложить галстуки и рубашки?  %%  Засунь всё в наволочки.
Что ты мечешься?  %% Успокойся.
Куда-то опаздываешь?  %% Нет.
Раз тебе так некогда, зачем ты меня звал?  %%  Вообще посоветоваться.
О чем советоваться, если ты уже железно решил?  %%  Вопрос в том, как ей это сказать.
А ты еще ей не сказал?  %%  В том-то и дело, что нет.
Для меня это тоже будет травма.  %%  Да, это непросто.
Вот поэтому я и хочу, чтобы ты поговорил с ней вместо меня.  %%  Я? Нет уж, уволь.
Ты мне друг или нет?  %%  Я тебе друг. Но с женой объясняйся сам.
Да какое тебе дело до ее слез?  %%  Я тоже человек.  Жалко ее.
```

Рисунок 3.9 — Приклад даних з набору даних “plays_ru”

2. Попередньо обробляється кожна пара, видаляються спеціальні символи в кожному реченні та формуються дві колекції з запитаннями та відповідями.

3. З корпусу отриманих колекцій створюється токенізатор (векторізований текстовий корпус, де текст перетворений у послідовність цілих чисел - ідентифікаторів) за допомогою Tensoralow Datasets пакету та його класу SubwordTextEncode . Для зменшення часу на створення токенізатора, якщо його не існує у паці з даними, то він створюється та зберігається у тій самій папці. Якщо токененізатор існує — залишається лише завантажити його, що займає набагато менше часу.

4. Дані “токенізуються”. До початку послідовності додається <START_TOKEN>(VOCAB_SIZE + 1) символ, а для кінця ставиться <END_TOKEN>(VOCAB_SIZE + 2).

5. Фільтруються послідовності, що містять більше MAX_LENGTH символів.

6. Послідовності, де не вистачає довжини доповнюються (tf.keras.preprocessing.sequence.pad_sequences()) до MAX_LENGTH символів

7. З оброблених даних будується tf.data.Dataset. Створення зображено на рис. 3.10.

```
dataset = tf.data.Dataset.from_tensor_slices((
    {
        'inputs': questions,
        'dec_inputs': answers[:, :-1]
    },
    {
        'outputs': answers[:, 1:]
    }
))
dataset = dataset.cache()
dataset = dataset.shuffle(len(questions))
dataset = dataset.batch(params.batch_size)
dataset = dataset.prefetch(tf.data.experimental.AUTOTUNE)
```

Рисунок 3.10 — Створення набору даних за допомогою tf.data.Dataset

Процес навчання моделі складається з наступних етапів:

1. Передається вхід через кодувальник, який повертає: основний вихід і прихований стан останнього рекурентного шару.

2. Вихід кодувальника, його прихований стан та вхід декодувальника (який є початковим токеном) передаються декодеру.

3. Декодувальник повертає прогнози та свій прихований стан.

4. Прихований стан декодувальника передається назад у модель, і прогнози використовуються для обчислення функції втрат та оцінки точності моделі. Необхідно маскувати функцію втрат таким чином, щоб маркери початку та кінця послідовності ігнорувалися (рис. 3.11).

```
def accuracy(y_true, y_pred):
    y_true = tf.reshape(y_true, shape=(-1, params.max_length - 1))
    return tf.keras.metrics.sparse_categorical_accuracy(y_true, y_pred)
```

Рисунок 3.11 — Функція для розрахунку точності, що ігнорує маркери початку та кінця послідовності

5. Далі використовується “навчання з вчителем”(техніка, при якій цільове слово передається як наступний вхід до декодувальника), щоб визначити наступний вхід до декодера.

6. Останнім кроком є обчислення градієнтів та їх застосування до функції оптимізатора та “backpropagation”(зворотного розповсюдження помилки).

Приклад процесу навчання приведено на рис. 3.12

```
/home/denis/Documents/Важное/Универ/диплом/program/venv/bin/python /home/denis/Documents/Важное/Универ/диплом/program/ChatBot_seq2seq/chatbot.py
2020-05-17 13:11:01.560583: E tensorflow/stream_executor/cuda/cuda_driver.cc:351] failed call to cuInit: UNKNOWN ERROR (303)

Model weights are loaded!
Train for 777 steps
Epoch 15/16
776/777 [============================>.] - ETA: 2s - loss: 0.5886 - accuracy: 0.1584
Epoch 00015: saving model to /home/denis/Documents/Важное/Универ/диплом/program/ChatBot_seq2seq/models/model_subtitles_ru/0015.ckpt
777/777 [==============================] - 1689s 2s/step - loss: 0.5889 - accuracy: 0.1585
Epoch 16/16
776/777 [============================>.] - ETA: 2s - loss: 0.5596 - accuracy: 0.1644
Epoch 00016: saving model to /home/denis/Documents/Важное/Универ/диплом/program/ChatBot_seq2seq/models/model_subtitles_ru/0016.ckpt
777/777 [==============================] - 1908s 2s/step - loss: 0.5596 - accuracy: 0.1644

Model weights are saved!

Bot works for a number of epochs: 16
Input:
```

Рисунок 3.12 — Приклад процесу навчання моделі

3.2.3.1 Результати навчання моделі

Результати роботи моделі на вагових коефіцієнтах після навчання з використанням набору “plays_ru”, 1601 тренувальних пар даних та після 210 інтерацій навчання приведено на рисунку 3.13.

```
Model weights are loaded!

Bot works for a number of epochs: 210
Input:  Привет
Output:  Пусть между мужем для тебя . нравится !
Input:  Как дела?
Output:  Ценные скорее .
Input:  Кто ты?
Output:  Вс зависит от тебя !
Input:  Ты бот?
Output:  Это было бы просто раньше предлагали ?
Input:  Ты робот?
Output:  Это было бы просто все таки выходят замуж ?
Input:  Ты любишь людей?
Output:  У людей много недостатков .
Input:  Да
Output:  Не знаю , какой ты раз нибудь муж .
Input:  Спасибо за разговор
Output:  Мне знаю , что говорю . может , тебе в самом деле стоит поехать .
Input:  Выход
```

Рисунок 3.13 — Результат навчання моделі на наборі “plays_ru”

Результати роботи моделі на вагових коефіцієнтах після навчання з використанням набору “subtitles_ru”, 50.000 тренувальних пар даних та після 20 ітерацій навчання приведено на рисунку 3.14.

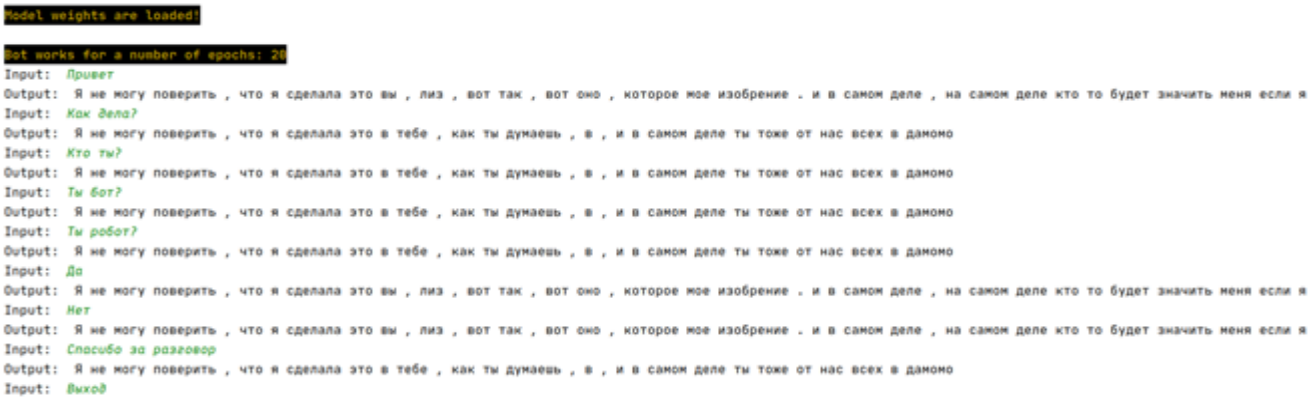

Рисунок 3.14 — Результат навчання моделі на наборі “subtitles_ru”

Результати роботи моделі на вагових коефіцієнтах після навчання з використанням набору “convestations_ru”, 50.000 тренувальних пар даних та після 15 ітерацій навчання приведено на рисунку 3.15.

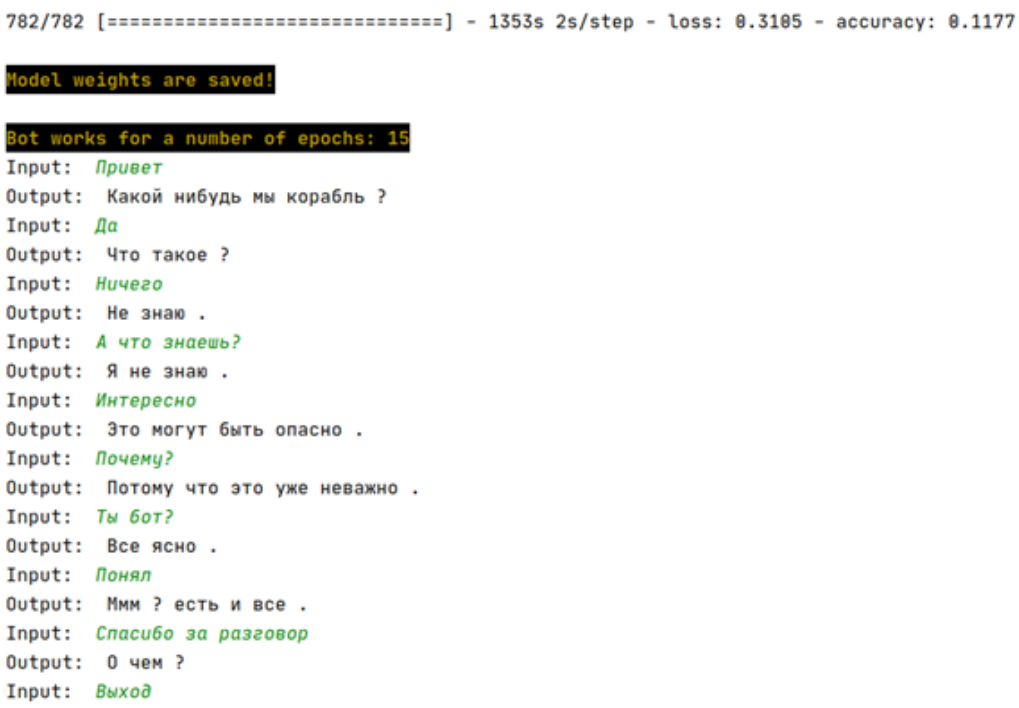

Рисунок 3.15 — Результат навчання моделі на наборі “convestations_ru”

ВИСНОВКИ

Таким чином, в дипломному проєкті був проведений аналіз особливостей імітації мовної поведінки людини в процесі спілкування, було визначено основні функції та принципи роботи чат-бота з метою моделювання архітектури та створення модуля за допомогою бібліотеки Tensorflow, що може навчати та використовувати вже навчені моделі для подальшого застосування і тестів.

Унікальні можливості Інтернет такі, як швидкість, оперативність, доступність комунікації між користувачами - дозволяють використовувати мережу як засіб спілкування і створювати інтерактивні форми спілкування: чати, форуми, телеконференції, електронну пошту та інші. На зміну реальним співрозмовникам приходять програми штучного інтелекту, такі як чати, консультанти, помічники, розважальні програми та інші. Але, на відміну від розмови людей, програма не володіє гнучким розумовою інтелектом. На жаль, сучасні віртуальні співрозмовники лише частково вирішують питання імітації розмови людини. Словниковий запас більшості віртуальних співрозмовників обмежений, крім цього, у них відсутня емоційне забарвлення, тембр голосу тощо., тому більшість віртуальних співрозмовників запрограмовані на ведення нескладної бесіди. Обробка природної мови людини, особливо розмовного стилю, є проблемою, що стосується штучного інтелекту. В основу функціонування віртуальних співрозмовників покладена база знань. У найпростішому випадку вона містить набори можливих питань користувача і відповідних відповідей на них. Деякі програми можуть навчатися, а саме: поповнювати словниковий запас, враховувати певні особливості мови, стилю спілкування. Але проблема створення програм співрозмовників на базі штучного інтелекту, які можуть моделювати інтелектуальну діяльність людини, на сьогоднішній день залишається відкритою. Незважаючи на переваги, віртуальні співрозмовники в даний час не можуть пройти тест Тьюринга на відповідність інтелекту комп'ютера людському інтелекту.

ПЕРЕЛІК ПОСИЛАНЬ

1. Азбука ИИ: «Рекуррентные нейросети». [Електрон. ресурс]. Спосіб доступу url: https://nplus1.ru/material/2016/11/04/recurrent-networks

2. Гусев С.С. Взаимосвязь человеческого фактора и искусственного интеллекта // Искусственный интеллект: философия, методология, инновации: Материалы III Всероссийской конференции студентов, аспирантов и молодых ученых, г. Москва, МИРЭА, 11–13 ноября 2009 г. / Под ред. Д.И. Дубровского и Е.А. Никитиной. – М.: Связь-Принт, 2009. – С. 279–281.

3. Гусев С.С. Нейронные сети, как основное направление развития искусственного интеллекта // Современная российская наука глазами молодых исследователей: Материалы III Международной научно-практической конференции молодых ученых и специалистов (Красноярск, 28 февраля 2013) / Научн. ред. Я.А. Максимов. – Красноярск: Изд. Научно-инновационный центр, 2013. – С. 147–153.

4. Гусев С.С. Современное мировоззрение на проблему искусственного интеллекта // Научное творчество XXI века: Сб. статей / Научн. ред. Я.А. Максимов. – Т. 2. – Красноярск: Изд. Научно-инновационный центр, 2012. – С. 73–77.

5. Гусев С.С. К вопросу о «рождении» и истории развития искусственного интеллекта // Искусственный интеллект: философия, методология, инновации: Материалы IV Всероссийской конференции студентов, аспирантов и молодых ученых, г. Москва, МИРЭА, 10–12 ноября 2010 г. 77/ Под ред. Д.И. Дубровского и Е.А. Никитиной. – Ч. 2. – М.: Связь-Принт, 2010. – С. 161–164.

6. Киунхьюн Чо. «Нейронный машинный перевод с применением GPU. Вводный курс. Часть 3.». [Електрон. ресурс]; Спосіб доступу url: http://datareview.info/article/neyronnyiy-mashinnyiy-perevod-s-primeneniem-gpu-vvodnyiy-kurs-chast-3/

7. Обзор исследований в области глубокого обучения: обработка естественных языков. [Електрон. ресурс]; Спосіб доступу url: https://habr.com/ru/company/wunderfund/blog/330194/

8. Amanda Stent and Srinivas Bangalore. 2014. Natural Language Generation in Interactive Systems. Cambridge University Press.

9. A. M. Turing. «Computing Machinery and Intelligence». In: Mind 49 (1950), pp. 433–460.

10. A Neural Network for Machine Translation, at Production Scale. [Internet]; url: https://ai.googleblog.com/2016/09/a-neural-network-for-machine.html

11. Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv: 1409.0473, 2014

12. Chatbots.org. Consumers Say No to Chatbot Silos in US and UK Survey. 2017. [Internet]; url: https://www.chatbots.org/images/news/chatbot_survey_2018.pdf

13. C. Olah. Understanding LSTM Networks. Aug. 2015. [Internet]; url: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

14. David Naylor «Podcast – Contact Centre AI: What are your options?». [Internet]; url: https://www.humanotics.ai/contact-centre-ai-what-are-your-options/

15. Dr. Nicola Millard «What do digital customers want». [Internet]; url: https://www.globalservices.bt.com/en/insights/articles/what-do-digital-customers-want

16. Goodman Joshua T. (2001). A bit of progress in language modeling, extended version. Technical report MSR-TR-2001-72.

17. Hochreiter S. Long short-term memory / S. Hochreiter, J. Schmidhuber // Neural Computation. – 1997. – Vol. 9, No. 8 – Pp. 1735-1780.

18. I. Sutskever, O. Vinyals, and Q. V. Le. «Sequence to Sequence Learning with Neural Networks». In: CoRR abs/1409.3215 (2014). arXiv: 1409.3215. [Internet]; url: http://www.arxiv.org/abs/1409.3215

19. Joseph Weizenbaum. ELIZA - a computer program for the study of natural language communication between man and machine. Communications of the Association for Computing Machinery, 9(1): 36–45, 1966.

20. J. Pennington, R. Socher, and C. D. Manning. «GloVe: Global Vectors for Word Representation». In: Empirical Methods in Natural Language Processing (EMNLP). 2014, pp. 1532–1543. [Internet]; url: http://www.aclweb.org/anthology/D14-1162

21. J. Weizenbaum. «ELIZA—a computer program for the study of natural language communication between man and machine». In: Communications of the ACM 9 (Jan. 1966), pp. 55–56.

22. K. Cho et al. «Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation». In: CoRR abs/1406.1078 (2014). arXiv: 1406.1078. [Internet]; url: http://arxiv.org/abs/1406.1078

23. R. Pascanu, T. Mikolov, and Y. Bengio. «Understanding the exploding gradient problem». In: CoRR abs/1211.5063 (2012). arXiv: 1211. 5063. [Internet]; url: http://www.arxiv.org/abs/1211.5063

24. S. Hochreiter and J. Schmidhuber. «Long Short-Term Memory». In: Neural Comput. 9.8 (Nov. 1997), pp. 1735–1780. [Internet]; url: http://dx.doi.org/10.1162/neco.1997.9.8.1735.

25. Some of the reasons people who use Python would rather not use anything else. [Internet]; url: https://www.python.org/about/

26. Steve Young. 2002. Talking to machines (statistically speaking). In Proc. Of interspeech.

27. Tensorflow Core r2.0. API Documentation [Internet]; 2019 [updated 2019 September 30]. url: https://www.tensorflow.org/api_docs

28. Tensorflow Datasets: a collection of ready-to-use datasets. [Internet]; url: https://www.tensorflow.org/datasets

29. Tom Hebner by James A. Larson «Ten Guidelines for Designing a Successful Voice User Interface». [Internet]; url: https://www.speechtechmag.com/Articles/ReadArticle.aspx/Editorial/Feature/Ten-Guidelines-for-Designing-a-Successful-Voice-User-Interface--29608.aspx

30. Tom Hebner «Thought Leadership: Tom Hebner of Nuance Communications». Nuance Leveraging AI, Offers New Project Pathfinder Tool for Crafting Intelligent Conversation Apps. [Internet]; url: https://www.aitrends.com/thought-leadership/thought-leadership-tom-hebner-of-nuance-communications/

31. T. Mikolov et al. «Efficient Estimation of Word Representations in Vector Space». [Internet]; url: http://www.arxiv.org/abs/1301.3781

32. Ubuntu for desktops. The open source Ubuntu desktop operating system powers millions of PCs and laptops around the world. [Internet]; url: https://ubuntu.com/desktop

ДОДАТОК А

Лістинг програми

Model.py

```
import argparse
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Embedding, Input, Dropout, GRU
from tensorflow.keras.layers import Attention, AdditiveAttention
from config import *

class BahdanauAttention(tf.keras.layers.Layer):
    def __init__(self, units):
        super(BahdanauAttention, self).__init__()
        self.W1 = tf.keras.layers.Dense(units)
        self.W2 = tf.keras.layers.Dense(units)
        self.V = tf.keras.layers.Dense(1)

    def call(self, query, values):
        query_with_time_axis = tf.expand_dims(query, 1)

        score = self.V(tf.nn.tanh(self.W1(query_with_time_axis) +
self.W2(values)))
        attention_weights = tf.nn.softmax(score, axis=1)

        context_vector = attention_weights * values
        context_vector = tf.reduce_sum(context_vector, axis=1)
        return context_vector, attention_weights

class LuongAttention(tf.keras.layers.Layer):
    def __init__(self, units):
        super(LuongAttention, self).__init__()
        self.wa = tf.keras.layers.Dense(units)

    def call(self, query, values):
        query_with_time_axis = tf.expand_dims(query, 1)
```

```
            score = tf.matmul(query_with_time_axis, self.wa(values),
transpose_b=True)
            attention_weights = tf.nn.softmax(score, axis=2)

            context_vector = tf.matmul(attention_weights, values)
            context_vector = tf.reduce_sum(context_vector, axis=1)
            return context_vector, attention_weights

        def AttentionSeq2Seq(params, name='Attention_Seq2Seq'):
            # ENCODER PART
            inputs = Input(shape=(None,), name="inputs")
            dec_inputs = Input(shape=(None,), name="dec_inputs")

            encoder_embedding = Embedding(input_dim=params.vocab_size,
output_dim=params.d_model)(inputs)
            encoder_embedding *= tf.math.sqrt(tf.cast(params.d_model, tf.float32))
            encoder_dropout = Dropout(rate=params.dropout)(encoder_embedding)

            encoder_GRU = GRU(params.num_units,
                        return_sequences=True,
                        return_state=True,
                        recurrent_initializer='glorot_uniform'
                        )(encoder_dropout)

            encoder_GRU2_layer = GRU(params.num_units,
                            return_sequences=True,
                            return_state=True,
                            recurrent_initializer='glorot_uniform')
            encoder_outputs, encoder_states = encoder_GRU2_layer(encoder_GRU)

            # DECODER_PART
            decoder_embedding = Embedding(input_dim=params.vocab_size,
output_dim=params.d_model)(dec_inputs)
            decoder_embedding *= tf.math.sqrt(tf.cast(params.d_model, tf.float32))
            decoder_dropout = Dropout(rate=params.dropout)(decoder_embedding)

            # Define Attention layer
            if params.attention_type:
```

```
            # context_vector = Attention(params.num_units)([encoder_outputs,
encoder_states])
            context_vector, _ = BahdanauAttention(params.num_units)(encoder_states,
encoder_outputs)
        else:
            # context_vector = AdditiveAttention(params.num_units)([encoder_outputs,
encoder_states])
            context_vector, _ = LuongAttention(params.num_units)(encoder_states,
encoder_outputs)

        context_vector = tf.expand_dims(context_vector, axis=1)
        context_vector = tf.tile(context_vector, multiples=[1,
tf.shape(decoder_dropout)[1], 1])
        concatenate_layer = tf.keras.layers.Concatenate()([decoder_dropout,
context_vector])

        decoder_GRU, state = GRU(params.num_units,
                                 return_sequences=True,
                                 return_state=True,
                                 recurrent_initializer='glorot_uniform'
                                 )(concatenate_layer, initial_state=encoder_states)

        decoder_dropout2 = Dropout(rate=params.dropout)(decoder_GRU)

        decoder_GRU2_layer = GRU(params.num_units,
                                 return_sequences=True,
                                 return_state=True,
                                 recurrent_initializer='glorot_uniform')
        decoder_outputs, state = decoder_GRU2_layer(decoder_dropout2)

        outputs = Dense(units=params.vocab_size, name='outputs')(decoder_outputs)

        return tf.keras.Model(inputs=[inputs, dec_inputs], outputs=outputs,
name=name)

    if __name__ == '__main__':
        parser = argparse.ArgumentParser()
        parser.add_argument('--max_samples', default=MAX_SAMPLES, type=int)
```

```
    parser.add_argument('--max_length', default=MAX_LENGTH, type=int)
    parser.add_argument('--batch_size', default=BATCH_SIZE, type=int)
    parser.add_argument('--num_units', default=NUM_UNITS, type=int)
    parser.add_argument('--d_model', default=D_MODEL, type=int)
    parser.add_argument('--dropout', default=DROPOUT, type=float)
    parser.add_argument('--attention_type', default=ATTENTION_TYPE,
type=int)
    params = parser.parse_args()
    params.vocab_size = 8192
    sample_model = AttentionSeq2Seq(params)
    sample_model.summary()
    tf.keras.utils.plot_model(sample_model, to_file='img/attention_seq2seq.png',
show_shapes=True)
```

Dataset.py

```
import os
import re
import time
import glob
import argparse
import tensorflow as tf
import tensorflow_datasets as tfds
from config import *

def preprocess_sentence(sentence):
    sentence = sentence.lower().strip()
    sentence = re.sub(r"([.?,!])", r" \1 ", sentence)
    sentence = re.sub(r[" "]+2', " ", sentence)
    sentence = re.sub(r"[^a-яA-Я?.!,]+", " ", sentence)
    return sentence.strip()

def tokenize_and_filter(params, tokenizer, questions, answers):
    tokenized_questions, tokenized_answers = [], []

    for sentence1, sentence2 in zip(questions, answers):
        sentence1 = params.start_token + tokenizer.encode(sentence1) +
params.end_token
```

```
            sentence2 = params.start_token + tokenizer.encode(sentence2) +
params.end_token

            if len(sentence1) <= params.max_length and len(sentence2) <=
params.max_length:
                tokenized_questions.append(sentence1)
                tokenized_answers.append(sentence2)

        # add length if sentence is not equal to 'max_length'
        tokenized_questions = tf.keras.preprocessing.sequence.pad_sequences(
            tokenized_questions, maxlen=params.max_length, padding='post')
        tokenized_answers = tf.keras.preprocessing.sequence.pad_sequences(
            tokenized_answers, maxlen=params.max_length, padding='post')

        return tokenized_questions, tokenized_answers

    def load_conversations(params, path_to_conversations):
        questions, answers = [], []

        with open(path_to_conversations, 'r') as file:
            lines = file.readlines()

        for line in lines:
            conversation = line.replace('\n', '').split('%%')

            questions.append(preprocess_sentence(conversation[0]))
            answers.append(preprocess_sentence(conversation[-1]))
            if len(questions) >= params.max_samples:
                return questions, answers
        return questions, answers

    def file_exists(path_to_folder, extension=False):
        if extension:
            return glob.glob(path_to_folder)
        return glob.glob(path_to_folder + '.*')

    def get_dataset(params):
```

```
        path_to_dataset = os.path.join(os.path.join(CURRENT_FOLDER,
PATH_TO_DATA_FOLDER), params.dataset)
        path_to_conversations = os.path.join(path_to_dataset, f'{params.dataset}.txt')
        questions, answers = load_conversations(params, path_to_conversations)
        path_to_vocab_file = os.path.join(path_to_dataset,
f'{params.dataset}_tokenizer')
        if file_exists(path_to_vocab_file):
            tokenizer =
tfds.features.text.SubwordTextEncoder.load_from_file(path_to_vocab_file)
        else:
            tokenizer =
tfds.features.text.SubwordTextEncoder.build_from_corpus(questions + answers,
target_vocab_size=VOCAB_SIZE)
            tokenizer.save_to_file(path_to_vocab_file)
        params.start_token = [tokenizer.vocab_size]
        params.end_token = [tokenizer.vocab_size + 1]
        params.vocab_size = tokenizer.vocab_size + 2
        questions, answers = tokenize_and_filter(params, tokenizer, questions,
answers)
        dataset = tf.data.Dataset.from_tensor_slices((
            {
                'inputs': questions,
                'dec_inputs': answers[:, :-1]
            },
            {
                'outputs': answers[:, 1:]
            }
        ))
        dataset = dataset.cache()
        dataset = dataset.shuffle(len(questions))
        dataset = dataset.batch(params.batch_size)
        dataset = dataset.prefetch(tf.data.experimental.AUTOTUNE)
        return dataset, tokenizer

    if __name__ == '__main__':
        parser = argparse.ArgumentParser()
        parser.add_argument('--max_samples', default=MAX_SAMPLES, type=int)
```

```
    parser.add_argument('--max_length', default=MAX_LENGTH, type=int)
    parser.add_argument('--batch_size', default=BATCH_SIZE, type=int)
    parser.add_argument('--num_units', default=NUM_UNITS, type=int)
    parser.add_argument('--d_model', default=D_MODEL, type=int)
    parser.add_argument('--dropout', default=DROPOUT, type=float)
    parser.add_argument('--epochs', default=EPOCHS, type=int)
    parser.add_argument('--attention_type', default=ATTENTION_TYPE,
type=int)
    parser.add_argument('--ckpt_dir', default=MODEL_CKPT_DIR, type=str)
    parser.add_argument('--dataset', default=DATASET, type=str)
    params = parser.parse_args()

    start_time = time.time()
    dataset, tokenizer = get_dataset(params)
    print('Dataset creation time:\t', time.time() - start_time)
```

Chatbot.py

```
import os
# 0 = all messages are logged (default behavior)
# 1 = INFO messages are not printed
# 2 = INFO and WARNING messages are not printed
# 3 = INFO, WARNING, and ERROR messages are not printed
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

import time
import argparse
import tensorflow as tf
from model import *
from utils import *
from dataset import *
from config import *

if tf.test.gpu_device_name():
    config = tf.compat.v1.ConfigProto()
    config.gpu_options.allow_growth = True
    session = tf.compat.v1.InteractiveSession(config=config)
```

```
    def main(params):
        dataset, tokenizer = get_dataset(params)
        model = AttentionSeq2Seq(params)
        model, latest_epoch_number = load_latest_model(params, model)

        def loss_function(y_true, y_pred):
            y_true = tf.reshape(y_true, shape=(-1, params.max_length - 1))

            loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True, reduction='none')(y_true, y_pred)

            mask = tf.cast(tf.not_equal(y_true, 0), tf.float32)
            loss = tf.multiply(loss, mask)

            return tf.reduce_mean(loss)

        def accuracy(y_true, y_pred):
            y_true = tf.reshape(y_true, shape=(-1, params.max_length - 1))
            return tf.keras.metrics.sparse_categorical_accuracy(y_true, y_pred)

        optimizer = tf.keras.optimizers.Adam(CustomSchedule(params), beta_1=0.9, beta_2=0.98, epsilon=1e-9)
        model.compile(optimizer=optimizer, loss=loss_function, metrics=[accuracy])

        if params.train:
            ckpt_callback = get_ckpt_callback(params)
            model.fit(dataset, epochs=params.epochs + latest_epoch_number,
                      initial_epoch=latest_epoch_number,
                      callbacks=[ckpt_callback])
            print('\n\033[1;33;40mModel weights are saved!\033[0m')
            latest_epoch_number += params.epochs
        print('\n\033[1;33;40mBot works for a number of epochs: {}\033[0m'.format(latest_epoch_number))
        print('\n\033[1;33;40mBot works with model: {}\033[0m'.format(params.ckpt_dir))
        print('\033[1;33;40mBot works with dataset: {}\033[0m'.format(params.dataset))
        while True:
```

```
            input_sentence = input('Input:\t')
            if input_sentence.lower() in ['exit', 'e', 'stop', 'выход', 'стоп']: break
            output = predict(params, model, tokenizer, input_sentence)
            print('Output:\t', output.capitalize())

    if __name__ == '__main__':
        parser = argparse.ArgumentParser()
        parser.add_argument('--max_samples', default=MAX_SAMPLES, type=int)
        parser.add_argument('--max_length', default=MAX_LENGTH, type=int)
        parser.add_argument('--batch_size', default=BATCH_SIZE, type=int)
        parser.add_argument('--num_units', default=NUM_UNITS, type=int)
        parser.add_argument('--d_model', default=D_MODEL, type=int)
        parser.add_argument('--dropout', default=DROPOUT, type=float)
        parser.add_argument('--epochs', default=EPOCHS, type=int)
        parser.add_argument('--attention_type', default=ATTENTION_TYPE,
type=int)
        parser.add_argument('--train', default=TRAIN_MODEL, type=str2bool)
        parser.add_argument('--ckpt_dir', default=MODEL_CKPT_DIR, type=str)
        parser.add_argument('--dataset', default=DATASET, type=str)
        params = parser.parse_args()

        try:
            main(params)
        except KeyboardInterrupt:
            print('\n', '\033[1;33;40mProgram was stopped!\033[0m')
```

Utils.py

```
import os
import re
import argparse
import tensorflow as tf
from dataset import preprocess_sentence
from config import *

def str2bool(v):
    if isinstance(v, bool):
        return v
```

```
        if v.lower() in ('yes', 'true', 't', 'y', '1'):
            return True
        elif v.lower() in ('no', 'false', 'f', 'n', '0'):
            return False
        else:
            raise argparse.ArgumentTypeError('Boolean value expected.')


    class CustomSchedule(tf.keras.optimizers.schedules.LearningRateSchedule):
        def __init__(self, params, warmup_steps=4000):
            super(CustomSchedule, self).__init__()

            self.d_model = tf.cast(params.d_model, dtype=tf.float32)
            self.warmup_steps = warmup_steps

        def __call__(self, step):
            arg1 = tf.math.rsqrt(step)
            arg2 = step * (self.warmup_steps ** -1.5)

            return tf.math.rsqrt(self.d_model) * tf.math.minimum(arg1, arg2)

    def load_latest_model(params, model):
        cp_path = os.path.join(CURRENT_FOLDER, MODELS_DIR)
        cp_path = os.path.join(cp_path, params.ckpt_dir)
        latest = tf.train.latest_checkpoint(cp_path)
        latest_epoch_number = 0 if latest is None else int(re.search(r'\d+',
latest.split('/')[-1]).group())

        try:
            model.load_weights(latest).expect_partial()
            print('\n\033[1;33;40mModel weights are loaded!\33[0m')
        except:
            pass

        return model, latest_epoch_number

    def get_ckpt_callback(params):
        cp_path = os.path.join(CURRENT_FOLDER, MODELS_DIR)
```

```
        cp_path = os.path.join(cp_path, params.ckpt_dir)
        cp_path = os.path.join(cp_path, "{epoch:04d}.ckpt")
        return tf.keras.callbacks.ModelCheckpoint(filepath=cp_path,
save_weights_only=True, verbose=1)

    def evaluate(params, model, tokenizer, sentence):
        sentence = preprocess_sentence(sentence)
        sentence = tf.expand_dims(params.start_token + tokenizer.encode(sentence) +
params.end_token, axis=0)

        output = tf.expand_dims(params.start_token, axis=0)

        for i in range(params.max_length):
            predictions = model(inputs=[sentence, output], training=False)
            predictions = predictions[:, -1:, :]
            # return the index of max value element from the dimension and
change(cast) that index to the int32 type
            predicted_id = tf.cast(tf.argmax(predictions, axis=-1), tf.int32)

            if tf.equal(predicted_id, params.end_token[0]):
                break

            output = tf.concat([output, predicted_id], axis=-1)
        return tf.squeeze(output, axis=0)

    def predict(params, model, tokenizer, sentence):
        prediction = evaluate(params, model, tokenizer, sentence)
        # Delete START and END token, after that do decode
        predicted_sentence = tokenizer.decode([i for i in prediction if i <
tokenizer.vocab_size])
```