ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

Пояснювальна записка

до магістерської роботи

на ступінь вищої освіти магістр

на тему: «РОЗРОБКА СИСТЕМИ ДІАЛОГОВОЇ МОДЕЛІ З ВИКОРИСТАННЯМ РЕКУРЕНТНОЇ НЕЙРОННОЇ МЕРЕЖІ ЗА ДОПОМОГОЮ TENSOR FLOW»

Виконав: студент 6 курсу, групи ПДМ-61 спеціальності
121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

Красюк І.В.
(прізвище та ініціали)

Керівник Сторчак К.П.
(прізвище та ініціали)

Рецензент _____
(прізвище та ініціали)

Нормоконтроль_____
(прізвище та ініціали)

КИЇВ – 2020

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут інформаційних технологій
Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти -«Магістр»
Спеціальність підготовки – 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення Негоденко О.В.
“____” _____2020року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

КРАСЮКУ ІЛЛІ ВІКТОРОВИЧУ

(прізвище, ім’я, по батькові)

1. Тема роботи: Розробка системи діалогової моделі з використанням рекурентної нейронної мережі за допомогою Tensor Flow

Керівник роботи: Сторчак Каміла Павлівна, завідувач кафедри ІСТ
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом вищого навчального закладу від «13» жовтня 2020 року №230.
2. Строк подання студентом роботи 24.12.2020р
3. Вихідні дані до роботи
Теоретичні й експериментальні дослідження.
Основи навчання та моделювання при вирішенні задач проектування діалогової моделі.
4. Зміст розрахунково-пояснювальної записки(перелік питань, які потрібно розробити).
4.1 Аналіз стану питання «Особливості ведення бесіди зі штучним інтелеком».
4.2 Використання рекурентних нейронних мереж при моделюванні розмови.
4.3 Опис математичних моделей та проектування системи.

4.4 Опис проектування системи.

5. Перелік демонстраційного матеріалу (назва основних слайдів):

1) Мета, об’єкт та предмет роботи.
2) Наукова новизна та практична значимість.
3) Аналіз аналогів
4) Технічне завдання
5) Спеціалізовані технології.
6) Архітектура TensorFlow.
7) Процес навчання нейронної мережі.
8) Схема рекурентної нейронної мережі та LSTM мереж.
9) Математичне представлення рекурентної нейронної мережі.
10) Демонстраційні скріншоти програми.
11) Апробація
12) Висновки.

6. Дата видачі завдання 02.11.2020р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	02.11-10.11	Виконано
2	Вимоги до системи	11.11-13.11	Виконано
3	Аналіз стану питання	14.11-16.11	Виконано
4	Аналіз нейронних мереж	17.11-20.11	Виконано
5	Конфігурація та результати моделювання	21.11-01.12	Виконано
6	Вступ, висновки, реферат	02.12-11.12	Виконано
7	Розробка презентації	12.12-15.12	Виконано

Студент _____

Керівник роботи _____

РЕФЕРАТ

Пояснювальна записка містить: сторінок – 71, рисунків – 43 , джерел – 20.

Об'єкт дослідження – процес спілкування людини зі штучним інтелектом.

Предмет дослідження – нейронна діалогова модель.

Мета роботи – розробка нейронної діалогової моделі на мові Python, використовуючи рекурентну нейронну мережу і бібліотеку машинного навчання TensorFlow для автоматизації ІС.

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

Новизна дослідження полягає в проведенні аналізу та виявленні недоліків навчанні моделі ввести діалог, а також в реалізації нейронної діалогової моделі на основі використання рекурентной нейронної мережі.

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

Для досягнення поставленої мети були виділені наступні завдання:

- Розробити універсальні програмні модулі, які можуть бути використані для підтримки проектування ділогових систем з різних програмних платформ.
- Розробити інтерфейс для графічного представлення результатів.
- Проаналізувати існуючі системи ведення діалогу зі штучним інтелектом з використанням рекурентної нейронної мережі.
- Провести розрахунки трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки.
- Провести маркетингові дослідження ринку збуту створеного програмного продукту.

Удосконалена методика дозволить проектувати діалогові системи зі значним скороченням як матеріальних витрат, так і тимчасових.

Галузь використання – сучасні діалогові системи на основі нейроної мережі.

НЕРОННА МЕРЕЖА, РЕКУРЕНТНА НЕЙРОННА МЕРЕЖА, ШТУЧНИЙ ІНТЕЛЕКТ, МАШИННЕ НАВЧАННЯ, ДІАЛОГОВА МОДЕЛЬ, PYTHON, TENSORFLOW, NUMPY, CUDA, PIP, UBUNTU, PYCHARM.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ШІ		Штучний інтелект
ОС		Операційна система
ПО		Програмне забезпечення
Seq2seq	sequence2sequence	Сімейство підходів машинного навчання, що використовуються для обробки мови
TF	TensorFlow	Програмного забезпечення для потоку даних та диференційованого програмування для різних завдань
НМ		Нейронна мережа
РНМ		Рекурентна нейронна мережа
ШНМ		Штучна нейронна мережа
ВС		Віртуальний співрозмовник
ДА		Діалоговий агент

ВСТУП

Актуальність роботи – 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 таку пропозицію, враховуючи попереднє речення або пропозиції 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.

Цілі і завдання дослідження – метою даної магістерської роботи 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 рекурентної нейронної мережі і бібліотеки машинного навчання TensorFlow.

Об'єкт дослідження – розробка нейронної діалогової моделі.

Предмет дослідження – діалоговий агент на основі рекурентної нейронної мережі.

Ідея роботи – полягає в реалізації нейронної діалогової моделі на мові python.

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

Наукові положення, очікувані наукові результати:

1) Сформований аналіз підходу до розробки діалогових систем, а також виявлення недоліків.
2) Реалізація нейронної діалогової моделі на основі використання рекурентної нейронної мережі і бібліотеки машинного навчання TensorFlow.

Методика дослідження:

1. Виборі методів досліджень і технологій реалізації.
2. Створення інформаційної системи, що реалізує механізми модельно-орієнтованого підходу.
3. Розробка теоретичної частини роботи, в якій досліджені і систематизовані знання про існуючі підходи розробки інформаційних систем і модельно-орієнтованих.
4. Оцінка отриманих результатів.

Новизна дослідження полягає в реалізації нейронної діалогової моделі з використанням бібліотеки машинного навчання TensorFlow.

Практична значимість дослідження полягає в реалізації нейронної діалогової моделі за допомогою бібліотеки машинного навчання Tensorflow, що дозволяють оцінити переваги проектування діалогових систем.

1 АНАЛІЗ СТАНУ ПИТАННЯ «ОСОБЛИВОСТІ ВЕДЕННЯ БЕСІДИ ЗІ ШТУЧНИМ ІНТЕЛЕКОМ»

1.1 Визначення штучного інтелекту

На новітньому етапі розвитку ШІ підійшов до рівня коли він самовдосконалюється дуже швидко, що позначається на таких процесах управління як алгоритмізація і оптимізація та самоорганізація. За допомогою сучасних суперкомп'ютерів ШІ з легкістю може вирішувати різноманітного роду алгоритмічні, математичні та програмні завдання високого рівня, що дозволяє оптимізувати процеси управління. Раніше теж ставилося питання про вирішення подібного роду завдань, але потужність обчислювальної техніки була непомірно малою і це не дозволяло вирішувати велику кількість поставлених завдань. В XXI столітті алгоритмізація і оптимізація процесів досягла таких масштабів, що можливо навіть управляти складними об’єктами завдяки ШІ. Тому за рахунок ШІ системи мають можливість самонавчатися та самовдосконалюватися. Але на даний час навіть суперкомп'ютери не приблизився до більшості параметрів людського мозку, тільки перевершуючи його за швидкістю обчислень. Швидкість розвитку штучного інтелекту навіть зараз вражає та протікає дуже швидко. По прогнозам аналітиків через 20 років штучний інтелект зрівняється або перевищить можливості людського мозку.

В XXI столітті розвиток ШІ пішов шляхом самовдосконалення через нейронні мережі. Що ж таке нейронні мережі. Нейронні мережі схожі на людську біологічну нейронну мережу таку як нервові клітини людського організму. З цього випливає, що поняття самих біологічних мереж виникло завдяки моделювання процесів, що протікають в мозку кожної людини. При спробі моделювання біологічних нейронних мереж було створено алгоритми, які в подальшому використовувались для вирішення різноманітних завдань по управлінню об’єктами в практичному застосуванні. НМ

містить в собі велику кількість зв’язаних між собою процесів, так званих штучних нейронів, які структурно є дуже простими. Саме завдяки процесам в НМ відбувається обробка, надсилання і аналіз всіх сигналів в системі. Завдяки тому що штучні нейрони з’єднані між собою та тісно взаємодіють, складні завдання розбиваються на прості процеси та виконуються.

НМ вивчає можливості людського інтелекту та інтерпретує їх за допомогою комп’ютерних алгоритмів. Головною ознакою та перевагою над алгоритмами є те що НМ не програмуються, а навчаються крок за кроком. З технічної точки зору навчання в НМ полягає в знаходженні певного коефіцієнту зв’язку між нейронами. Під час навчання нейронна мережа виявляє певну залежність між вхідними та вихідними даними, що дає їй приблизитися до потрібного результату. НМ свого роду паралельний пристрій за допомого якого можливо швидко обробляти інформацію. Реалізація НМ можлива як в якості ПЗ для звичайного комп’ютера або як АЗ: аналогове, цифрове або змішане. Для найкращої реалізації, що дозволить підвищити швидкість та гнучкість використовуються спеціальні комп’ютери – нейрокомп’ютери на яких уже встановлене відповідне програмне забезпечення.

Рисунок 1.1 – Нейрокомп'ютер Sunway TaihuLight.

На приклад, китайський нейрокомп'ютер Sunway TaihuLight, що зображено на рис. 1.1, був запущений у 2016-му році і є одним із найпотужніших та найдорожчих у світі. Машина використовує 40 тисяч процесорів та коштує близько $ 270 млн. Назва комп'ютеру переводиться як "божественна сила", і він створений для складних обстежень у промисловості, інженерному розробці, медицині, а також для прогнозування переваг і великих даних.

Для оброки НМ великої кількості інформації, її потрібно навчити. Якщо говорити точніше то це свого роду «тренінг», але за історією потрібно вживати термін «навчання». Тож якщо нейронна мережа не навчена всі її реакції на певні дії будуть хаотичними. По суті навчання являє собою багаторазовому наданні потрібних прикладів до тих пір поки не буде отриманий бажаний для користувача відгук. НМ задається певний бажаний відгук, щоб передбачуване значення були передбачувані величини. В більшості випадків цей бажаний відгук встановлено так званим «учителем». Якщо ж передбачення відбулося «учитель» формує вхідні дані, потім вибирає число потрібних кроків передбачення на виході та фільтрує непотрібні компоненти в вихідній частковій послідовності, щоб в результаті отримати більш достовірний відгук. Використання високочастотної флуктації представляти немає великої необхідності, тому і використовують фільтрацію для отримання бажаного відгука у вигляді згладженої вихідної послідовності.

Між бажаним відгуком та реальним виходом НМ повинна бути різниця, яка в свою чергу називається помилкою. Помилка має в собі такі показники як величина та знак, які в свою чергу служать для адаптації вагових коефіцієнтів. Дані коефіцієнти задаються на старті та є випадковими. При зміні вагових коефіцієнтів НМ навчається цей процес називається алгоритмом або правилом навчання. Зменшення с середньоквадратичної помилки є критерієм навченості, який зменшується з кожним кроком нового повтору деякого прикладу. Якщо середньоквадратична помилка досягає заздалегідь заданої на початку величини або коли все бажані відповіді НМ

тобто правильно оброблені данні, тоді можна говорити, що навчання нейронної мережі є завершеним.

Ідея коли НМ є самоорганізованою по своїй архітектурі є не новою. Але для кожного такого досягнення повинен пройти певний час. С кожним днем прогрес не стоїть на місці, що дозволяє впровадження нейронних мереж різноманітні галузі людського життя.

За певним шляхом розвитку НМ є намагання впровадити їх для вирішення великої кількості повсякденних задач. Але розробка даного вирішення завдань є трудомістким процесом та займає велику кількість часу для того щоб досягти потрібного результату. У період 60-х років, коли зароджувалося програмування дало змогу обробляти велику кількість інформації швидше ніж мозок людини. Тому з часом НМ зможуть вирішувати певні проблеми людей.

Наступним етапом в становленні НМ припадає на 80-ті роки. Саме в цей час В Японії був створений перший в своєму роді нейронний комп'ютер, який не мав обмежень в пам'яті та швидкодії. На той час нейронні комп'ютери використовувалися для в розпізнаванні певних образів. На сьогоднішній день для створення НМ використовуються три способи: програмний, апаратний та гібридний – який є поєднанням в собі апаратного та програмного. Варто відзначити те що НМ має собою певний алгоритм, який поєднує в собі послідовність певних дій та певних прийнятих рішень. Але є певний мінус даного алгоритму в тому що прийняті рішення обмежені певною ситуацією і вони в враховують людський фактор. Прогнозування показує нам на високу ступінь ймовірності даного рішення не зважаючи на людський фактор.

1.2 Віртуальний співрозмовник

Віртуальний співрозмовник (англ. Chatterbot) – це програма, яка імітує мовну поведінку одної або декількох людей при спілкуванні з ними. Віртуальний

співрозмовник має також не офіційну назву, яка показує на призначення програми, а саме програма-співрозмовник.

Першим в своєму роді створеним віртуальним співрозмовником була створена в 1966 році Джозефом Вейзенбаумом – програма Еліза. Дана програма була створенна як певного роду пародія психотерапевта. Вона вміла слухати, перепитувати користувача в разі потрібності та дуже часто використовувала фрази типу «Будь ласка, продовжуйте».

Однією з головних ознак, яка вказує на ідеальну програму-співрозмовник це те що вона повинна пройти тест Тюрінга. Також кожного року проводяться конкурси програм-співрозмовників здебільшого тих що на англомовні. Самий відомий – конкурс Лебнера. На рис. 1.2 представлено інтерфейс програми віртуального співрозмовника Еліза (ELIZA).

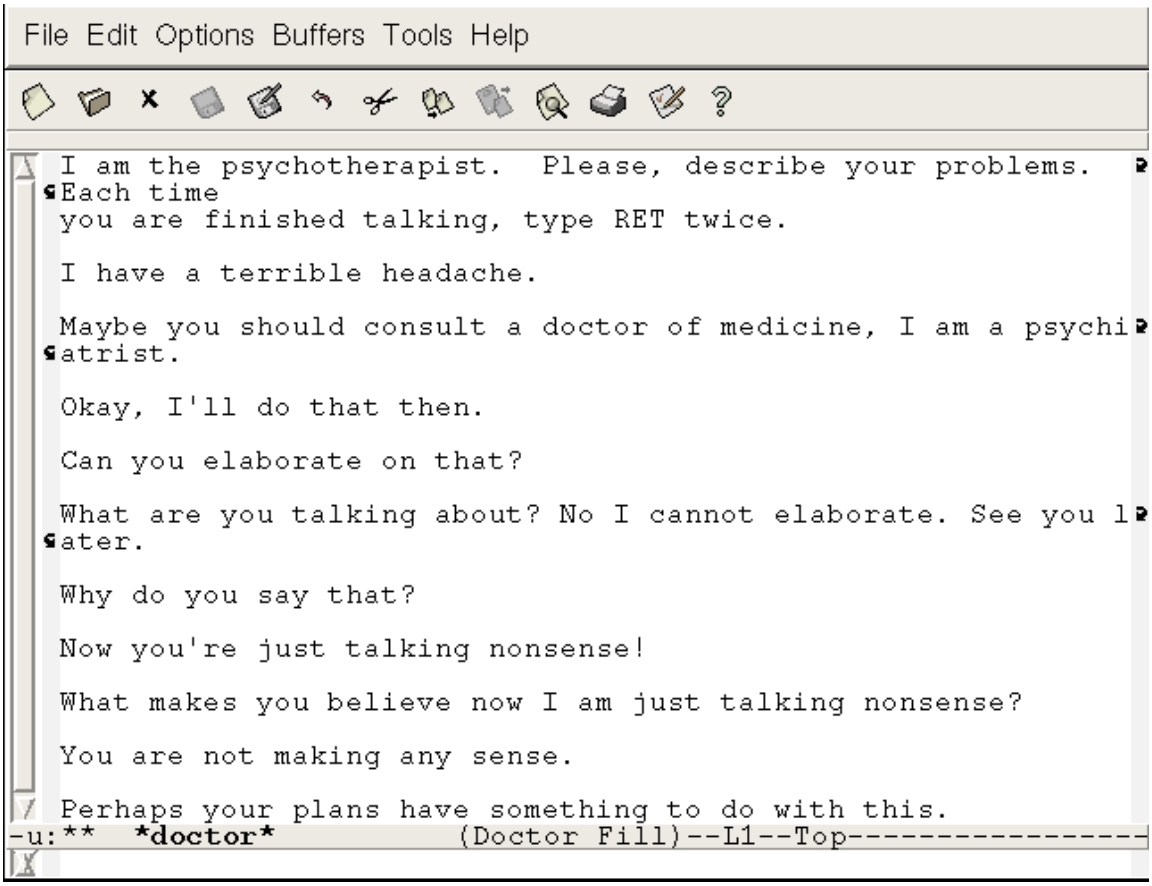

Рисунок 1.2 – Інтерфейс програми віртуального співрозмовника Еліза (ELIZA)

1.2.1 Призначення віртуальних співрозмовників

Визначення віртуального співрозмовника не повністю описує всі його можливості. Цілі, які люди потребують від діалогу з віртуальним співрозмовником часто відрізняються. Можна водночас обговорювати з ним певну важливу проблему, а можна просто поговорити. Для того, щоб програма вирішувала певну поставлену проблему вона повинна мислити як людина.

Тому на даний час більшість віртуальних співрозмовників можуть вести тільки невигадливі бесіди.

Є навіть програми, які розуміють певні висловлювання користувача, що вказує на їх природно-мовний інтерфейс. Одним з прикладів є питально-відповідна система.

Поняття віртуального співрозмовника тісного пов’язане з поняттям штучного інтелекту, тобто при їх поєднанні створюється певна модель інтелектуальної діяльності людини.

1.2.2 Принципи дії діалогових агентів

Віртуальні співрозмоники мають працювати з так званою «живою» мовою. Гострою проблемою штучного інтелекту є те що він не вміє обробляти природну мову тобто особливості мовного стилю. Водночас сучасні програми-співрозмовники це певна спроба імітації розумного діалогу.

Кожний віртуальний співрозмовник, як інтелектуальна система повинна мати бузу знань. У простому випадку в ній присутні набір можливих питань, які може задати користувач та набір оптимальних відповідей. Одні з самих поширених відповідей такі:

1) Реакція на ключові слова: саме цей метод був використаний в першому віртуальному співрозмовнику Елізі. Наприклад, якщо в фразі користувача до віртуального співрозмовника були слова «брат», «сестра», «син» та інші. Еліза могла відповісти: «Будь-ласка розкажіть ще щось про вашу сім'ю».

2) Збіг фрази: це вказує на те що фраза користувача має збіг в базі знань. Також в даному випадку може враховуватися послідовність слів.
3) Збіг контексту: в більшості інструкцій до програм-співрозмовникам зауважують не використовувати ті фрази, які насичені займенниками, наприклад : «А що це таке?». Є навить певні програми, які аналізують всі фрази користувача і вибирають найбільш прийнятні відповіді.

Однією міні-проблемою є ідентифікація форм слова і синонімів.

Як влаштована логіка обробки запитів в діалогових платформах?

Розглянемо процес обробки на прикладі платформи JustAI, але треба відзначити, що на верхньому рівні основні риси однакові якщо не у всіх, то принаймні у відомих нам платформах. Загальну схему роботи платформи зображено на рис. 1.3.

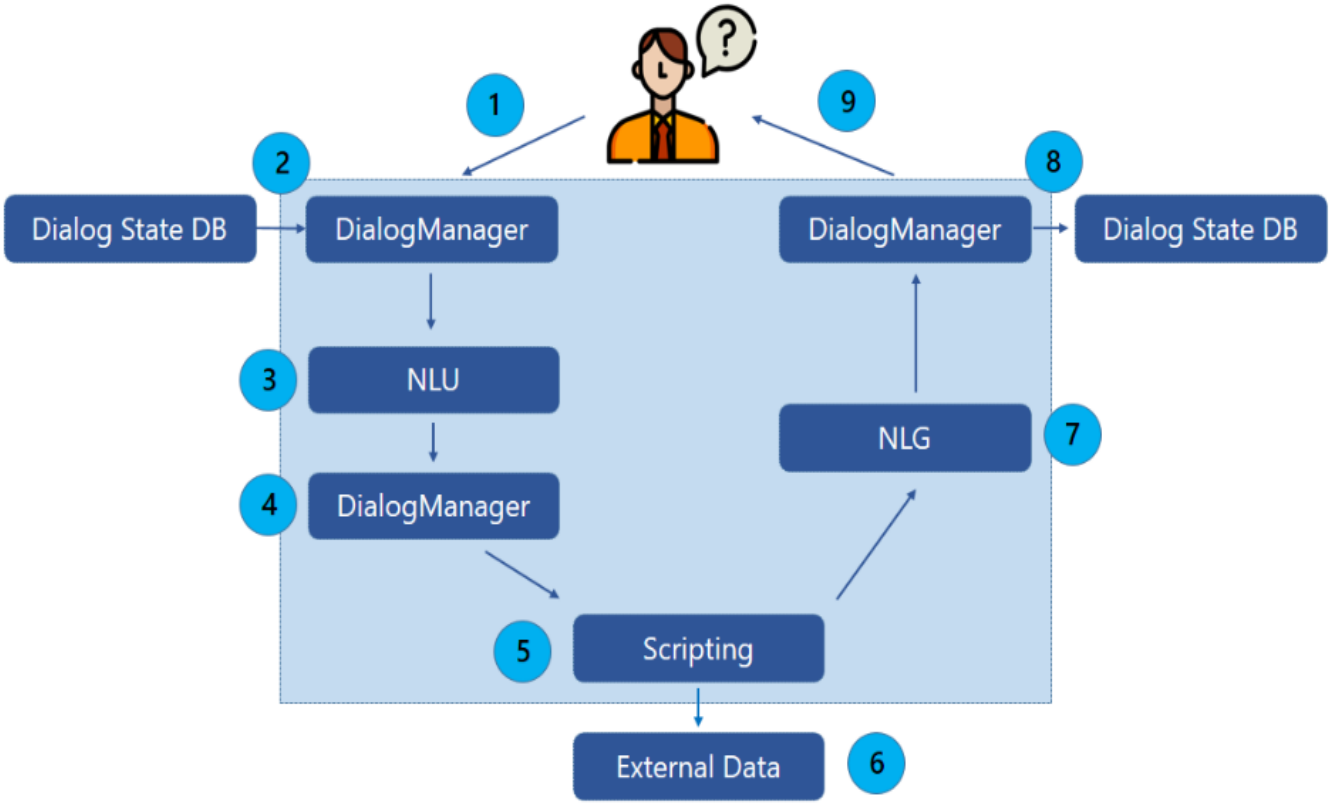

Рисунок 1.3 – Схема роботи діалогової системи JustAI.

Основний цикл обробки запиту клієнта складається з наступних подій і дій:

1) Система отримує запит клієнта в модуль управління діалогом - DialogManager.
2) DialogManager завантажує контекст діалогу з бази даних.
3) Запит клієнта (разом з контекстом) відправляється на обробку в NLU-модуль, в результаті чого визначається намір клієнта і його параметри. У разі обробки не текстових подій (кнопки і т.д.) цей крок пропускається.
4) На основі сценарію діалогу і витягнутих даних, DialogManager визначає найбільш доцільний стан (блок, екран, сторінку діалогу), який повною мірою відповідає запиту клієнта.
5) Виконання бізнес-логіки (скриптів) відповідно до заданого сценарієм чат-бота.
6) Виклик зовнішніх інформаційних систем, якщо такі запрограмовані в бізнес-логіку.
7) Генерація текстової відповіді з використанням макропідстановок і функцій узгодження слів на природній мові.
8) Збереження контексту і параметрів діалогу в DialogState DB для обробки наступних звернень.
9) Відправлення відповіді клієнту.

1.3 Особливості та проблеми віртуального спілкування

Поєднання систем віртуального спілкування на основі ШІ стоїть уже протягом багатьох років. Основною проблемою зараз є швидкий доступ до інформації, який потрібен системі віртуального спілкування. Також є і інші не менш важливі проблеми такі як: можливість роботи в системі багатьох користувачів, обмін інформацією між ними, підтримка навчання, проведення аналітичних досліджень, комунікація з клієнтами та партнерами по бізнесу, збору необхідної інформації та інші.

При створені систем спілкування основним моментом є розробка моделі спілкування, розвиток засобів, моделі учасника спілкування, а одне з головних опису

навколишнього середовища. Тому потрібно приділити не малу кількість часу, щоб вирішити дані питання. Головним чинником, який цьому сприяє є особливості мовної поведінки людини, яка впливає на розуміння того що хоче користувач та розробка моделі спілкування заточена на певного користувача. Серед програм-співрозмовників є також програми, які в своєму роді створені на основі ШІ.

Для того щоб розробляти подібного роду програми необхідно знати психологію та розуміти побудову фраз мови на який створюється програма. Якщо розробник має певні знання мови та предметної області можливо створити систему придатну для спілкування з користувачами даної мови. Необхідно розумітися в теорії мови та спілкування, для того щоб програма могла створювати складні мовні структури та пропозиції. Модель навколишнього середовища має велику кількість обмежень, тому програма не може подати динамічно мінливий світ. Це все пов’язано з проблемами певного сприйняття світу таким, який він є та з точки зору навчання системи.

Метою роботи є аналіз особливостей імітації мовної поведінки людини в процесі спілкування, розробка моделі спілкування, написання чат-бота.

1.3.1 Основні функції і принцип роботи чат-бота

Комп'ютерну програму бот являє собою введення-виведення повідомлень потрібної користувачу інформації, а також використання різноманітних простих функцій. Боти поділяються на такі види: службові, інформаційно розважальні, функції утиліт. Службові функції відповідають за логи обліку прав всіх учасників боту, безпеку користування, можливість з’єднання декількох користувачів в конференцію. Інформаційно-розважальні функції це можуть бути ігри, довідки користування, віртуальні співрозмовники та таке інше. Також в боті можуть біти присутні різного роду утиліти: перекладач, пошук, калькулятор.

Принцип роботи всіх чат-бота схожий та полягає в реалізації етапів: бот приймає вхідні повідомлення, потім аналізує їх і відсилає результат виконання і/або виконує команду.

На рис. 1.4 представлено схему функціювання чат-бота в маркетинговій сфері.

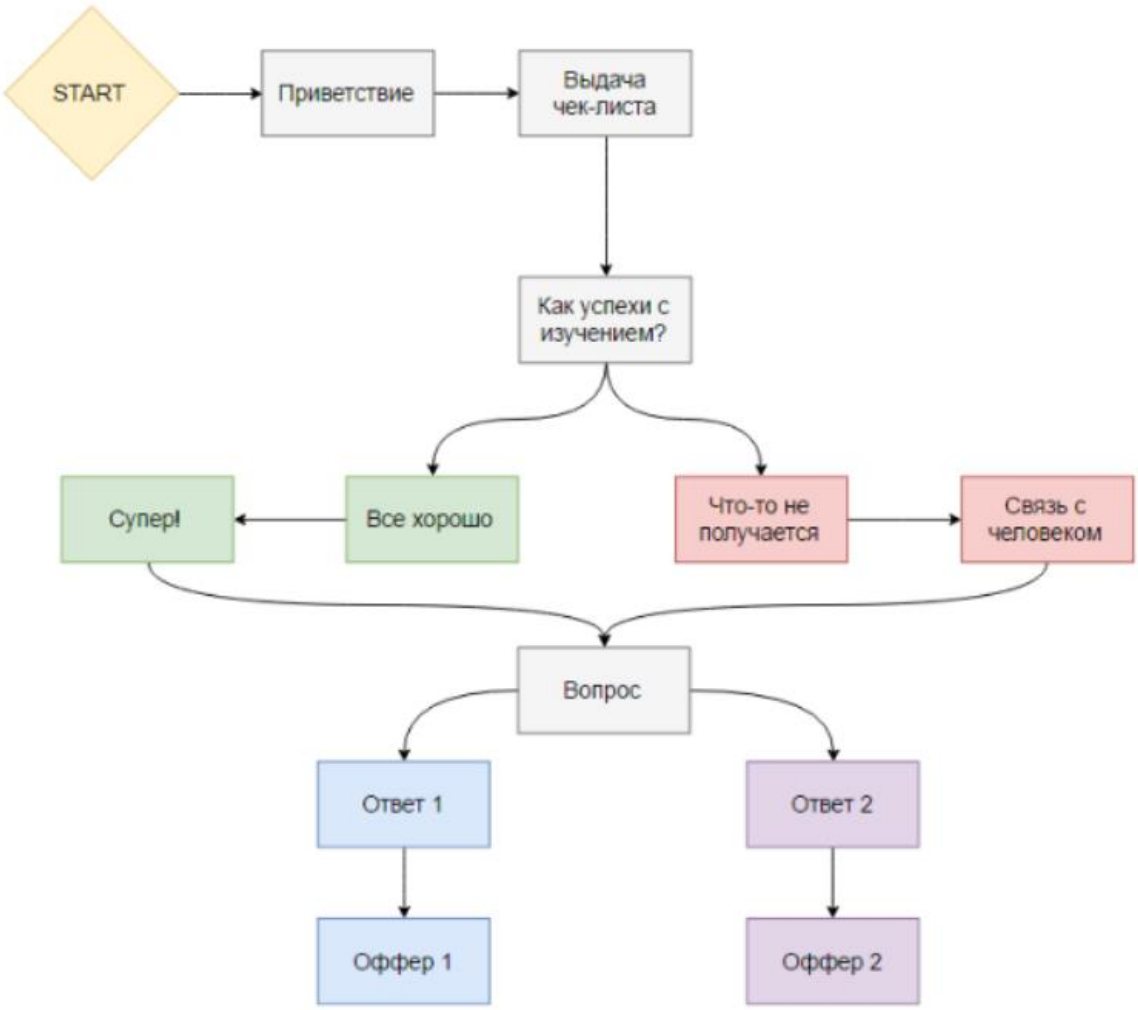

Рисунок 1.4 – Схема функціювання чат-бота в маркетинговій сфері.

Висновки до розділу 1

Таким чином з цього розбілу випливає, що спілкування в чат-ботах здійснюються шляхом введення повідомлення користувачем та отриманнч

очікуваного результату у вигляді виведення відповіді. Можливі два варіанта розмови: обговорення важливих для користувача питань аба звичайна розмова. Але, є одина з головних відмінностей віртуальних співрозмовників це те що вони не володіють гнучким людським інтелектом, тому вирішення складних завдань практично неможливе.

Дані програми відносяться до класу з природним мовним інтерфейсом. Якщо брати до уваги гтучний інтелект, то його головною проблемою є саме обробка прироної мови людини. Моделювання інтелектуальної діяльності людини є проблемою програм співрозмовників на основі штучного інтелекту, яка досі є невирішеною.

На даний момент віртуальні співрозмовники лише частково вирішують питання імітації розмови людини. Для їх повного функціонування повинна бути база знань. База знань являє собою певний набір можливих питань користувача та відповідні відповіді на них.

На сьогоднішній день розроблено велику кількість самарських пошукових роботів. Серед них можна виділити найбільш поширені: ALICE, ChatMaster, Electronic Brain, ELIZA, George, NAI, SkypeTalk і інші.

2 ВИКОРИСТАННЯ РЕКУРЕНТНИХ НЕЙРОННИХ МЕРЕЖ ПРИ МОДЕЛЮВАННІ РОЗМОВИ

2.1 Штучні нейронні мережі

Штучні нейронні мережі - це структури обробки інформації, що забезпечують (часто невідомий) зв’язок між вхідними та вихідними даними шляхом штучного моделювання фізіологічної структури та функціонування структур мозку людини .

Натомість, природна нейронна мережа складається з дуже великої кількості нервових клітин (близько десяти мільярдів у людей), зазначають нейрони, і пов'язані між собою в складну мережу. Розумна поведінка є результатом широкої взаємодії між взаємопов'язаними блоками. Вхід нейрона складається з вихідних сигналів з'єднаних з ним нейронів. Коли внесок цих входів перевищує певний поріг, нейрон - за допомогою відповідної передавальної функції - генерує біоелектричний сигнал, який поширюється через синаптичні ваги до інших нейронів.

Істотними особливостями цієї мережі, яку штучні нейронні моделі мають намір імітувати:

1) Паралельну обробку (завдяки тому, що нейрони одночасно обробляють інформацію).
2) Подвійну функцію нейрона (який одночасно діє як пам’ять і процесор сигналів).
3) Розподілений характер подання даних (тобто. знання розповсюджуються по всій мережі, не обмежуються та не визначаються наперед).
4) Прилади, побудовані на принципах біологічних нейронів, мають перераховані характеристики, які можна вважати суттєвим досягненням в індустрії обробки даних.
5) Здатність мережі вчитися на попередньому досвіді.

Це остання, але фундаментальна здатність дозволяє нейромережам самоорганізовуватися, адаптуватися до нової вхідної інформації та витягувати зв’язки введення-виведення з відомих прикладів, які є основою їх організації. Штучна нейронна мережа фіксує це ставлення на відповідному етапі “навчання”.

Незважаючи на великий успіх, досягнутий штучними нейронними мережами, все ж краще не забувати про межі цієї технології через необхідне скорочення реальної системи, яку потрібно дослідити.

2.1.1 Структура нейронної мережі

Штучні нейронні мережі складаються з елементарних обчислювальних одиниць, званих нейронами, об'єднаних відповідно до різних архітектур. Наприклад, вони можуть бути розташовані шарами (багатошарова мережа), або вони можуть мати топологію з'єднання. Багаторівневі мережі складаються з:

1) Вхідний рівень, виготовлений з n нейронів (по одному на кожен вхід мережі).
2) Прихований шар, що складається з одного або декількох прихованих (або проміжних) шарів, що складаються з m нейронів.
3) Вихідний рівень, що складається з p нейронів (по одному на кожен вихід мережі).

Режим з'єднання дозволяє розрізнити два типи архітектури:

2.1.1 Архітектура зворотного зв'язку із зв’язками між нейронами того самого або попереднього шару.

2.1.2 Архітектура прямого пересилання, без зворотних зв’язків (сигнали надходять лише до нейронів наступного шару).

Кожен нейрон отримує n вхідних сигналів x_i (з вагами з'єднання w_i), які складаються до значення "активації" y, на рис. 2.1 зображено будову штучного нейрона.

Відповідна функція передачі (або активації) *F* перетворює її у вихід *F(y)*. Працездатність мережі (а саме його знання) міститься у вагах з'єднання, які приймають свої значення завдяки фазі навчання.

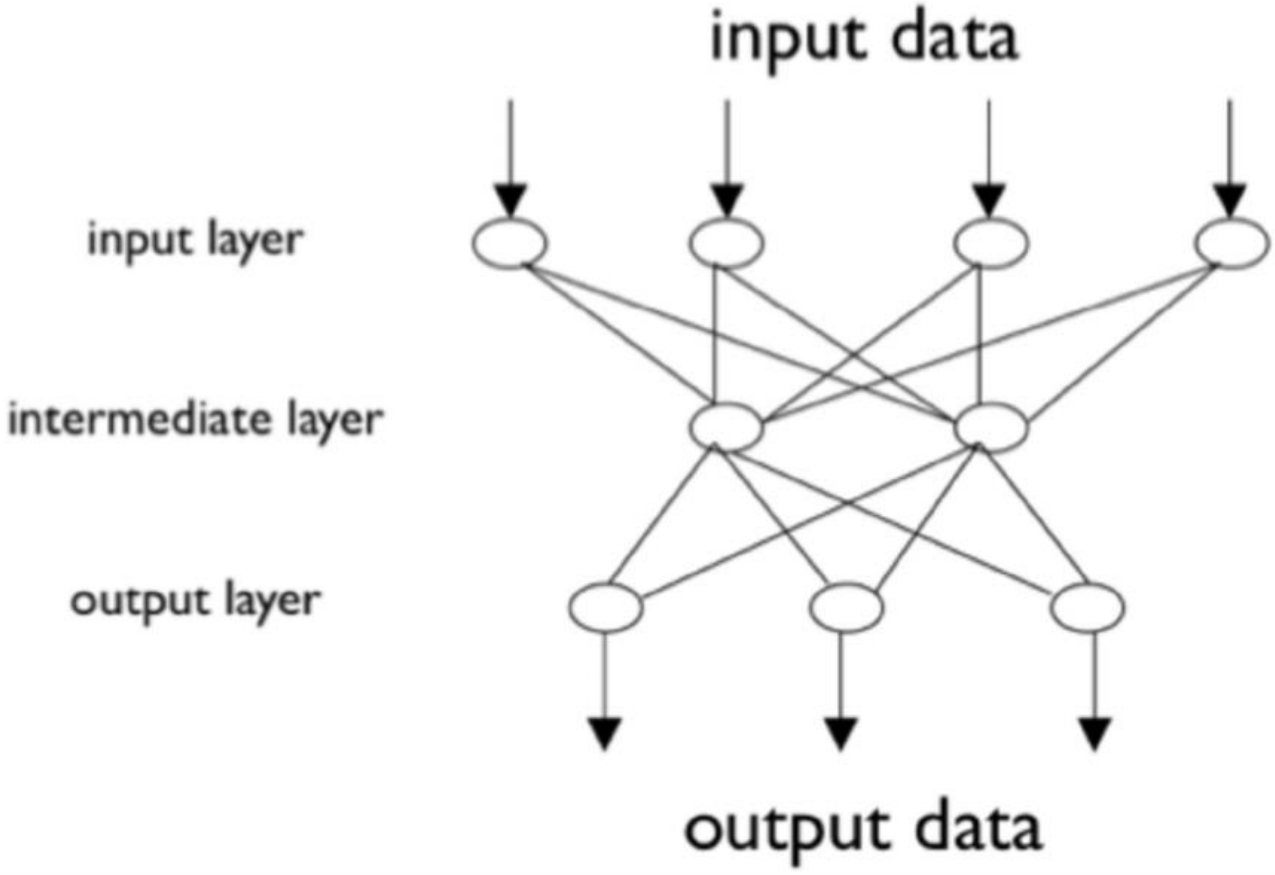

Рисунок 2.1 - Будова штучного нейрона

Останні експериментальні дослідження доводять, що біологічні нейрони структурно складніше, ніж спрощене пояснення існуючих штучних нейронів, які є елементами сучасних штучних нейронних мереж. Оскільки нейрофізіологія надає науковцям розширене розуміння дії нейронів, а технологія обчислень постійно вдосконалюється, розробники мереж мають необмежений простір для поліпшення моделей біологічного мозку.

2.1.1 Архітектура та модель нейронних мереж

Найпростіша мережа складається з одного нейрона з *n* входами та одним виходом. Базовий алгоритм навчання перцептрону аналізує вхідну конфігурацію (шаблон) *i* - зважування змінних через синапси - вирішує, який вихід пов'язаний з конфігурацією. Цей тип архітектури представляє основне обмеження можливості вирішувати лише лінійно відокремлювані проблеми.

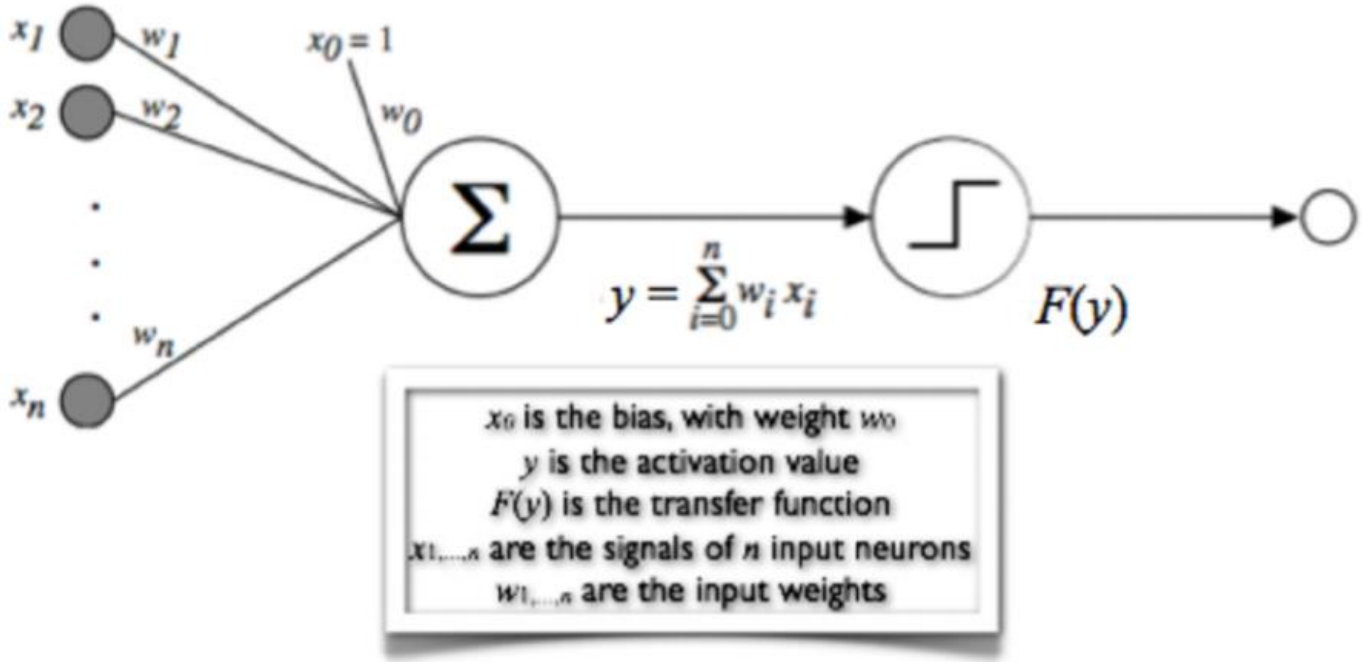

Рисунок 2.2 - Структура прямої багатошарової нейронної мережі

Нейронна мережа з вхідним шаром, одним або декількома проміжними шарами нейронів і вихідним шаром називається багатошаровим перцептроном або MLP(Multi Layer Perceptron). Ця мережа, що зображена на рис. 2.2 має тип прямого пересилання і використовує в більшості випадків, алгоритм навчання зворотного розповсюдження. Він обчислює ваги між шарами, починаючи від випадкових значень і вносячи невеликі поступові та поступові зміни після вихідних помилок мережі, доки алгоритм навчання не сходиться до прийнятного наближення помилок. цілі. Дійсно, архівована архітектура зворотного розповсюдження, що контролюється, є найпопулярнішою і

широко використовується, оскільки цей набір моделей має узагальнювати результати для великої кількості проблем.

2.1.2 Застосування нейронних мереж

Додатки нейронної мережі можна згрупувати за трьома основними областями:

1) Класифікація.
2) Прогнозування часових рядів.
3) Наближення функції.

Апроксимація функції та прогнозування часових рядів

У разі наближення функції (або регресії) мережі застосовуються до всіх ситуацій, у яких відсутня точна функціональна форма, що описує відносини введення-виведення. Конкретна (корисна) область прогнозування часових рядів має на меті передбачити майбутні значення (показники) через минулі доступні періоди даних.

Розглянемо, наприклад, прогнозування обмінного курсу: він може впливати лише на точне значення, а не на тенденцію періоду. У першому випадку вихід нейронної мережі використовується для впровадження торгової системи. У другому випадку буде достатньо знати, що ринок зростає або падає, щоб мати можливість порівняти його з очікуваною динамікою на інших ринках.

У будь-якому випадку, в кожному з можливих застосувань, з операційної точки зору, необхідно розділити серію на дві частини: одна, зроблена з так званих спостережень у вибірці, виступає в якості бази для навчання (навчального набору). Інший, зроблений на основі поза зразкових спостережень, має на меті перевірити його обґрунтованість (набір перевірок). Мережу можна навчити складати прогнози для більш широких горизонтів, використовуючи власні короткострокові прогнози як вхідні дані для довгострокових прогнозів.

Для ефективності цього типу застосування важлива оцінка окремих технічних аспектів, таких як:

1) *Вибір вхідних змінних:* це слід зробити зважаючи на те, що мережа не в змозі забезпечити будь-яку пояснювальну функцію, тому вона може

використовувати несуттєві змінні насправді, відносини між змінними змінюються з часом, а отже, значущі введення сьогодні можуть бути неправдивими в майбутньому.

2) *Оптимальний рівень навчання:* необхідно взяти до уваги, що занадто короткий процес навчання не дозволяє мережі фіксувати взаємозв'язки між змінними, тоді як занадто довгий тренінг може зробити мережу нездатною до узагальнення (надмірне).
3) *Вибір часового горизонту для прогнозу:* це є важливим фактором того, що дуже короткі горизонти прогнозу збільшують кількість правильних прогнозів, тому на відміну від цього, ознаки тривалих часових горизонтів прогнозування в середньому менш правильні, але правильні призводять до вищої середньої вигоди.

2.1.3 Класифікація та дискримінація нейронних мереж

Нейронні мережі також можуть бути використані для класифікації даних. Неподібні проблеми регресії, де метою є отримання результату, що відповідає даному введенню, проблеми класифікації вимагають позначення кожної точки даних як належності до одного з n заданих класів. Нейронні мережі можна навчити забезпечувати дискримінанту функцію, що розділяє класи. У типовій класифікаційній задачі з двома класами пряма лінія може бути використана як дискримінаційна, класи називаються так званими лінійно розділимі. Ці проблеми можна вивчити без прихованих одиниць, але іноді для забезпечення відокремлення класів потрібна нелінійна функція: це може бути вирішене лише нейронною мережею з одним або декількома прихованими рівнями.

Типовими додатками для класифікації є, наприклад, кредитні рейтинги, рішення про довіру або оцінка біологічного ризику. У цих випадках мережа має завдання призначити вхід до відповідного виходу серед ряду заздалегідь визначених категорій.

Щоб використовувати нейромережу для класифікації, вам потрібно побудувати еквівалентну задачу наближення функції, присвоївши цільове значення кожному класу. Для двійкової задачі (два класи) ви можете використовувати нейронну мережу з одним виходом у та бінарним цільовим значенням: 0 для одного класу, 1 для іншого. Тому ви можете інтерпретувати результати роботи мережі як оцінку ймовірності належності даної вибірки до одного з двох класів. У цих випадках часто використовується функція активації, яка насичує два цільові значення, тобто логістичну функцію:

$$f(x) = 1/(1 + e^{-x}) \quad (2.1)$$

2.1.4 Навчання штучної нервової мережі

Нейронна мережа не програмується безпосередньо, але вона явно навчається за допомогою алгоритму навчання для вирішення даного завдання, процесу, що веде до «навчання через досвід». Алгоритм навчання допомагає визначити конкретну конфігурацію нейронної мережі, а отже, обумовлює та визначає здатність самої мережі надавати правильні відповіді на конкретні проблеми.

Ми розрізняємо щонайменше три типи навчання: без нагляду, під наглядом та підкріплення. У першому випадку мережа навчається лише на основі набору входів, не забезпечуючи відповідних виходів. Для навчання під наглядом необхідно натомість визначити набір прикладів, що складаються з відповідних входів та відповідних результатів, які повинні бути представлені мережі, щоб вона «вчилася» на них. Навчання за допомогою підкріплення використовується в тих випадках, коли неможливо визначити схеми введення-виведення. Підсилення забезпечується системою, яка інтерпретує її як позитивний / негативний сигнал про її поведінку та відповідно регулює налаштування.

Набір даних, що використовується для мережевого навчання, становить так званий навчальний або навчальний набір.

Модифіковані входи передаються на функцію підсумовування, яка переважно тільки підсумовує твори. Можна вибрати різні операції, такі як середнє арифметичне, максимальне, найменше, OR, AND, і т.д., що виробляють різні значення. Більшість комерційних програм дозволяють інженерам-програмістам створювати власні функції суматора за допомогою підпрограм, закодованих на мові високого рівня. Іноді функція підсумовування ускладняється додаванням функції активації, роздільною функції підсумовування діяти в часі.

У будь-якому з цих випадків, вихід функції підсумовування проходить через передавальну функцію на вихід (0 або 1, -1 або 1, або яке-небудь інше число) за допомогою певного алгоритму. В існуючих нейромережах як передавальних функцій можуть бути використані сигмоїда, синус, гіперболічний тангенс і інше.

Алгоритм зворотного поширення

Алгоритм зворотного розповсюдження використовується під контролем навчання. Це дозволяє змінювати ваги з'єднання таким чином, що мінімізує певні помилки функцію *E*. Ця функція залежить від і-го виходу мережі (вектора) неті з урахуванням і-го входу (вектора) x_i та і-го (вектора) бажаного виводу y_i. Таким чином, навчальний набір - це набір з *N* пар вхідних / вихідних даних. Функцію помилки, яку слід мінімізувати, можна записати як:

$E(w) = \frac{1}{2} \cdot \sum i_{\sum} k \cdot (netik_{\_y} \cdot ik)^2$, де: (2.2)

k - являє собою значення, що відповідає k-му вихідному нейрону;

$E(w)$ - є функцією залежно від ваг (які, як правило, змінюються з часом);

Щоб звести його до мінімуму, ви можете використовувати алгоритм градієнт-спуска, який починається з загальної точки даних і обчислює градієнт, який дає напрямок руху, щоб досягти максимального збільшення (або зменшення, якщо врахувати -∇). Одного разу визначивши напрямок, ви рухаєтеся на попередньо встановлену η відстань і знаходите нову точку, на якій перераховуєте градієнт. Алгоритм продовжується ітеративно, поки градієнт не дорівнює нулю.

Алгоритм зворотного просування можна розділити на два етапи:

1) *Крок вперед*: вхідні дані в мережу переносяться на наступний рівень тощо. Потім обчислюється помилка мережі $E(w)$.
2) *Крок назад:* помилка, допущена мережею, поширюється назад, і ваги відповідно оновлюються.

Логічними етапами навчання нейронної мережі з контрольованим навчанням є наступні:

1. Створіть набір шаблонів введення та відповідний набір (бажаних) шаблонів виводу.
2. Ініціалізуйте ваги нейронної мережі до випадкових значень.
3. Цикл навчання (цей цикл закінчується лише тоді, коли помилка, як правило, менше заздалегідь визначеного порогу або після заданої кількості ітерацій).
    1) Фаза прямої передачі (від вхідного до вихідного шару):
        a) випадково витягнути шаблон введення з наявних;
        b) обчислити значення всіх наступних нейронів;
        c) відняти від результату значення порогової активації нейрона (якщо це значення ще не змодельовано з додаванням фіксованої одиничної вхідної величини, зміщення);
        d) відфільтруйте вихід нейрона, застосовуючи логістичну функцію, щоб зробити це значення входом наступного нейрона.
    2) Порівняйте результат мережі з відповідним шаблоном виводу та отримайте поточну помилку мережі.
    3) Фаза зворотного розповсюдження (від вихідного до вхідного шару):
        a) обчислити поправку на ваги відповідно до обраного мінімального правила розміщення;
        b) застосувати корекцію до ваг шару.

Для гарної підготовки набір вхідних моделей повинен бути повністю вивчений (епоха). Отже, після випадкового вибору візерунка, при витягуванні наступного

старий не повинен розглядатися. Таким чином, на кожному кроці в навчанні беруть участь лише ще не оброблені вхідні дані, таким чином здійснюючи повну обробку всього навчального набору.

2.1.5 Побудова нейронної мережі

Побудова нейронної мережі обов'язково передбачає певні кроки, що вимагають встановлення відповідних параметрів:

1. Розбиття бази даних.

Для того, щоб отримати оптимальну підготовку та впровадження мережі, необхідно розділити набір даних на підмножини, які визначають середовище навчання: навчальний та перевірочний набір. Останній, у свою чергу, поділяється на набір тестів та узагальнення.

По суті, мережа вчиться, намагаючись розпізнати динаміку навчального набору, перевіряє, наскільки він відповідає тестовому набору, а потім застосовує себе до набору даних (узагальнення), який ніколи раніше не спостерігався.

Не існує універсально діючих правил для підрозділу набору даних: прийнятими рішеннями є (60% - 20% - 20%) та (60% - 30% - 10%) відповідно для навчального, тестового та узагальнювальних наборів.

2. Кількість прихованих шарів та нейронів.

Є численні емпіричні дослідження, в яких використовується один прихований шар, оскільки достатньо наблизити нелінійні функції з високим ступенем точності. Однак цей підхід вимагає великої кількості нейронів, тому обмежує процес навчання. Іноді мережі з двома прихованими рівнями є більш ефективними, особливо для прогнозування високочастотних даних. Цей вибір, окрім того, що пропонується конкретною теорією, підтверджується досвідом, який показує, як, з іншого боку, більше двох прихованих шарів не дають значних поліпшень у результатах, отриманих мережею.

Слід також зазначити, що надмірна кількість нейронів може генерувати перенавчання, тобто нейрони не здатні генерувати надійний прогноз, оскільки вони зменшують внесок вхідних даних; у цих випадках це так, ніби мережа “зберегла” правильні відповіді, не маючи можливості узагальнити. Навпаки, занадто мала кількість нейронів зменшує навчальний потенціал мережі.

Формули, запропоновані в літературі, дуже різні, а в деяких випадках суперечливі:

$$h = 2 \cdot n + 1 \quad (2.3)$$

$$h = 2 \cdot n \quad (2.4)$$

$$h = n \quad (2.5)$$

$$h = \frac{n+m}{2+t^{1/2}} \quad (2.6)$$

де:

h – кількість прихованих нейронів;

n – кількість вхідних нейронів;

m – кількість вихідних нейронів;

t – кількість спостережень, що містяться в навчальному наборі.

Емпіричні результати показують, що жодне з цих правил не здається узагальненим для кожної проблеми, хоча, здається, не безглузда кількість позитивних результатів.

2. Механізми з'єднання.

Існує кілька режимів з'єднання між різними шарами:

1. *Стандартні підключення*: забезпечують пряме з’єднання без зворотного зв’язку між входом і виходом, які проходять через один або декілька прихованих шарів.
2. *Перехідні з'єднання:* мережа може призначати сполучні ваги навіть між нейронами, які не знаходяться в сусідніх шарах.

3. *Множинні підключення*: забезпечує можливість того, що нейрони, присвоєні прихованих шарів можуть повернутися до вхідних змінних з ітераційних процесів для того, щоб точно визначити кількість ваги з'єднання.
3. Функції активації.

Ми можемо виділити кілька типів різних функціональних форм, що зображено в таб.2.1, які впливають на зв'язування нейронів (лінійні, синусоїдальні, гаусові та інші). Ви можете визначити різну функцію активації для кожного шару.

Таблиця 2.1 – Функції активації

Функція	Формула
Лінійна	$f(x) = x$ (2.7)
Логістична (сигмоподібна)	$f(x) = \frac{1}{1+e^{-x}}$ (2.8)
Логістично- симетрична	$f(x) = \frac{1+e^{-x}}{2}$ (2.9)
Гіперболічний тангенс	$f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$ (2.10)
Виправлена дотична	$f(x) = tg(\mathrm{c} \cdot x)$ (2.11)
Синусоїдальна	$f(x) = sin(x)$ (2.12)
Гауссова	$f(x) = e^{-x^2}$ (2.13)
Обернений Гауссова	$f(x) = 1 - e^{-x^2}$ (2.14)

Немає теоретично прийнятного правила для визначення функції активації різних шарів. Хоча багато досліджень представляють різні функції для окремих шарів, деякі мають однакову функцію для вхідного, прихованого та вихідного шарів. Лінійна функція зазвичай використовується для вихідного рівня. Однак його використання менш ефективно в прихованих шарах, особливо якщо вони характеризуються великою кількістю нейронів.

Логістична та симетрична логістичні функції мають характеристику, яка змінюється відповідно в інтервалах [0, 1] та [−1, 1]. Деякі проблеми мають динамічні характеристики, які більш точно визначаються симетричною функцією, особливо у вхідному та прихованому шарах. Більша частина емпіричної літератури показує використання цієї функції у прихованому шарі, хоча і без сильної теоретичної мотивації.

Гіперболічна дотична функція дозволяє надійно адаптувати мережу в прихованих шарах (особливо в мережах із трьома шарами), особливо у випадку, коли аналітик обрав логістичну або лінійну функцію для виведення.

Синусоїдальна функція, як правило, використовується в дослідженнях, і пропонується нормалізувати вхідні та вихідні дані в межах [-1, 1].

Гауссова функція піддається ідентифікації конкретних динамічних процесів, зафіксованих двома прихованими архітектурами паралельних шарів, з функцією дотичних у другому шарі.

4. Правила навчання.

Після визначення початкових характеристик нейронної мережі необхідно визначити критерії припинення навчання. Якщо ви хочете побудувати нейронну мережу з метою прогнозування, переважно оцінювати її засвоєння на наборі тестів. Якщо ви хочете адекватно описати досліджуване явище, переважно оцінювати навчання за навчальним набором. Далі параметри навчання пов’язані з показниками помилок мережі (середня помилка, максимальна помилка, кількість разів без покращення помилок).

5. Оновлення ваг з’єднання.

Подальшою проблемою є вибір правила навчання: зокрема, необхідно визначити швидкість, з якою мережа змінює ваги порівняно з помилкою. Розгляньте малюнок 3. Зліва ви бачите прогрес як у навчанні, так і в коливаннях, з частотою, яка швидше досягає бажаного результату; праворуч мережа навчається, змінюючи свій

«шлях» з меншою швидкістю і досягає цілі за вищий час. Швидкість навчання η зазвичай спочатку встановлюється на значення:

$$\eta = \frac{[\max(x) - \min(x)]}{N} \quad (2.15)$$

де x – являє собою навчальний набір;

N – число вхідних записів.

Через компенсацію вхідних значень x та відносного числа N це початкове значення η повинно бути точним для багатьох проблем класифікації, незалежно від кількості навчальних зразків та діапазону їх значень. Однак, хоча найвищі значення η можуть прискорити процес навчання, це може спричинити коливання, які можуть уповільнити збіжність.

В якості альтернативи можна використовувати momen-tum, тобто частку, з якою варіація останньої ваги, досягнута нейронною мережею, додається до нової ваги. Таким чином, мережа може навчитися навіть з вищою швидкістю, але вона, швидше за все, не похитнеться, оскільки отримує велику частку останньої втрати ваги.

Потім аналітик повинен визначити рівень початкової ваги зв’язку між нейронами та оцінити, чи характеризують високі рівні шуму спостереження. У цьому випадку вам слід підтримувати значення імпульсу. Процес навчання мережі переходить природним чином із поступового визначення найбільш підходящих значень для цих параметрів. Тому це вимагає тривалої серії спроб, які можуть привести до суттєво різних результатів. Пропозиція полягає в тому, щоб уникнути радикальних змін параметрів.

2.2 Нейронна розмовна модель

2.2.1 Моделювання розмови

Досягнення наскрізного навчання нейронних мереж призвело до значних успіхів у багатьох сферах, таких як розпізнавання мовлення, комп’ютерний зір та обробка мовлення. Останні досягнення свідчать про те, що нейронні мережі можна

використовувати не тільки для виконання завдань класифікації, але також можна використовувати для порівняння деяких складних структур з іншими складними структурами. Прикладом цього є послідовність порівнянь з іншими послідовностями, які безпосередньо пов'язані з розумінням природної мови. Основна перевага цієї структури полягає в тому, що в порівнянні з нею потрібно мало функціональних технологій та галузевих специфікацій. Ця перевага дозволяє дослідникам працювати над можливостями в цій галузі.

2.2.2 Модель

Підхід для вирішення задачі моделювання діалогу використовує наступну структуру послідовностей (seq2seq). Модель заснована на обертовій нейронній мережі, яка зчитує вхідну послідовність одного маркера за раз і забезпечує вихідну послідовність t, по одному знаку за раз. Під час тренування вихідна послідовність визначається як шаблон, який проводиться шляхом відтворення.

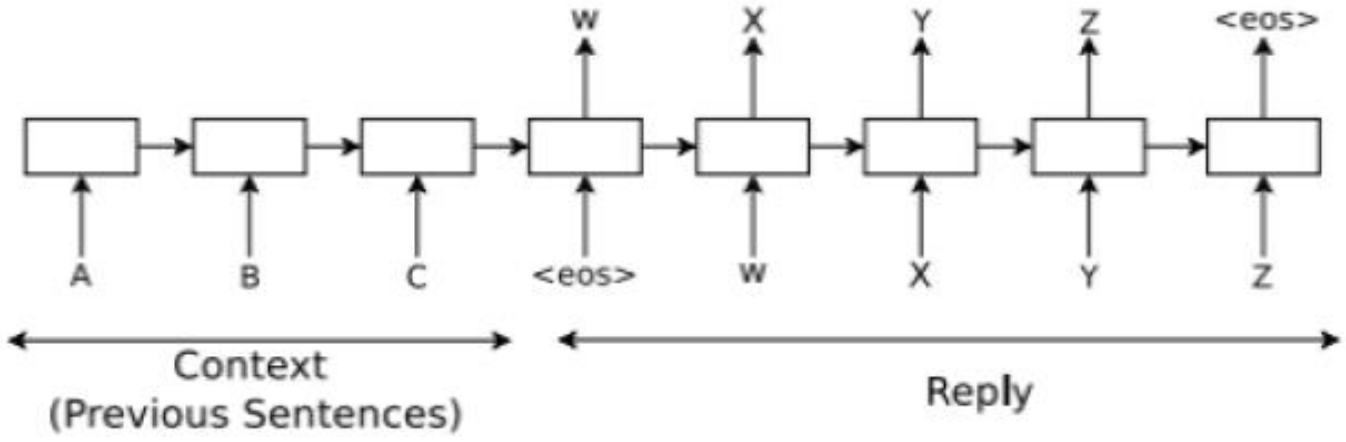

Рисунок 2.7 – Використовуйте Seq2seq для імітації діалогу

Модель вчиться максимізувати поперечну ентропію правильної послідовності відповідно до її змісту. Під час виходу, оскільки жодної справжньої вихідної послідовності не спостерігається, ми просто надаємо передбачений вихідний маркер

як вхідний сигнал для передбачення наступного виходу. Прогнозовану послідовність можна вибрати на основі ймовірності послідовності.

Зокрема, припустимо, що ми проводимо розмову: перша людина говорить ‘ABC”, а друга – WXYZ. Ми можемо використовувати повторювану нейронну мережу і навчити її відображати ‘ABC’ на ‘WXYZ’, як показано на фіг. 2. 7 вище.

Прихований стан моделі можна розглядати як вектор відбиття, оскільки вона зберігає інформацію про речення або думку «ABC», коли вона отримує кінець знака послідовності. Перевагою цієї моделі є її простота та послідовність. Ми можемо використовувати цю модель для машинного перекладу, запитань / відповідей та бесід без значних змін в архітектурі. Застосування цієї методики до розмовного моделювання також є простим: вхідна послідовність може бути об’єднанням того, про що вже йшлося (контекст), а вихідна послідовність є відповіддю. На відміну від більш простих завдань, таких як переклад, послідовно подібна модель не зможе успішно «вирішити» проблему моделювання діалогу через низку очевидних спрощень.

2.3 Рекурентні нейронні мережі

Рекурентні нейронні мережі – це нейронна мережа, призначена для аналізу потоків даних за допомогою прихованих блоків. У деяких додатках, таких як обробка тексту, розпізнавання мови та послідовності ДНК, результат залежить від попередніх обчислень. Оскільки РНМ мають справу з послідовними даними, вони добре підходять для домену інформатики в галузі охорони здоров’я, де доступні для обробки величезні обсяги послідовних даних.

Ідея РНМ полягає у використанні послідовної інформації. У традиційній нейронній мережі ми припускаємо, що всі входи (і виходи) не залежать один від одного. Але для багатьох завдань це дуже погана ідея. Якщо ви хочете передбачити наступне слово у реченні, вам краще знати, які слова стояли перед ним. РНМ називаються рекурентними, оскільки вони виконують одне і те ж завдання для

кожного елемента послідовності, при цьому вихідні дані залежать від попередніх обчислень. Інший спосіб думати про РНМ – це те, що вони мають “пам’ять”, яка фіксує інформацію про те, що нарахувано до цього часу. Теоретично РНМ можуть використовувати інформацію у довільно довгих послідовностях, але на практиці вони обмежуються оглядом лише кількох кроків (про це далі). Ось як виглядає типовий рекурентна нейронна мережа, що зображена на рис. 2.8.

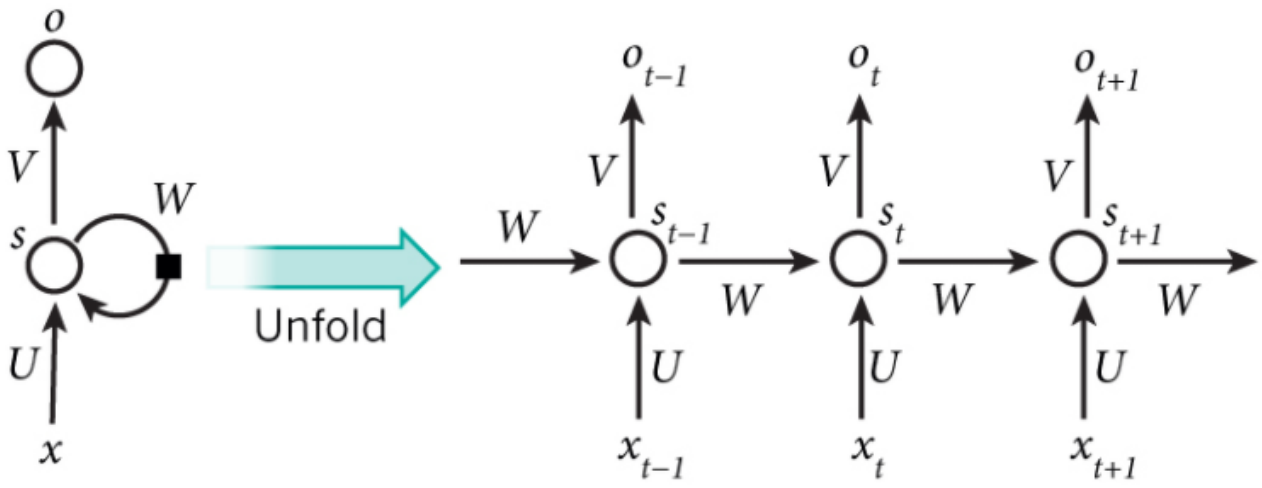

Рисунок 2.8 – Модель рекурентної нейронної мережі

На наведеній вище схемі показано, як РНМ розгортається (або розгортається) у повну мережу. Під розгортанням ми просто маємо на увазі, що ми виписуємо мережу для повної послідовності. Наприклад, якщо послідовність, яку ми турбуємо, є реченням із 5 слів, мережа буде розгорнута в 5-шарову нейронну мережу, по одному шару на кожне слово. Формули, що регулюють обчислення, що відбуваються в РНМ, такі:

1) x_t – вхід на кроці часу t, наприклад, x_1 може бути гарячим вектором, що відповідає другому слову речення.

2) s_t – прихований стан на кроці часу t. Це «пам'ять» мережі. s_t обчислюється на основі попереднього прихованого стану та вводу на поточному кроці:

$$s_t = f(U_{x_t} + W_{s_{t-1}}) \quad (2.16)$$

3) O_t - вихід на кроці t. Наприклад, якби ми хотіли передбачити наступне слово у реченні, це був би вектор імовірностей у нашому словниковому запасі.

$$O_t = softmax(V_{s_t}) \quad (2.17)$$

Тут слід зазначити кілька речей:

1. Ви можете уявити прихований стан s_t як пам'ять мережі. s_t фіксує інформацію про те, що сталося за всі попередні кроки часу. Вихід на етапі O_t обчислюється виключно на основі пам'яті в момент часу t. Як коротко згадувалося вище, на практиці це дещо складніше, оскільки s_t, як правило, не може захоплювати інформацію із занадто багато часових кроків тому.
2. На відміну від традиційної глибокої нейронної мережі, яка використовує різні параметри на кожному рівні, РНМ використовує однакові параметри (U, V, W вище) на всіх етапах. Це відображає той факт, що ми виконуємо одне і те ж завдання на кожному кроці, лише з різними входами. Це значно зменшує загальну кількість параметрів, які нам потрібно вивчити
3. Наведена діаграма має результати на кожному кроці часу, але в залежності від завдання це може не знадобитися. Наприклад, прогнозуючи почуття речення, ми можемо дбати лише про кінцевий результат, а не про почуття після кожного слова. Подібним чином, нам може не знадобитися введення даних на кожному кроці часу. Головною особливістю РНМ є її прихований стан, який фіксує деяку інформацію про послідовність.

РНМ показали великий успіх у багатьох завданнях НВП. На цьому етапі я повинен згадати, що найбільш часто використовуваними типами РНМ є LSTM, які набагато краще фіксують довгострокові залежності, ніж стандартні РНМ. Але не

хвилюйтеся, LSTM – це, по суті, те саме, що РНМ, який ми розробимо в цьому посібнику, вони просто мають інший спосіб обчислення прихованого стану. Більш детально ми розглянемо LSTM у наступній публікації. Ось кілька прикладів застосувань РНМ в НВП (необов'язково вичерпний список).

У останній публікації Google розробники детально описують багато рішень, які допомагають їм оптимізувати поточну версію Google Translate, але проблема досі залишається. Наприклад, рідкісні слова, сленг, чи навмисне спотворення слова.

На рис 2.12 зображено принцип роботи машинного перекладача Google Translate, заснованого на комбінації кілька рекурентних нейромереж.

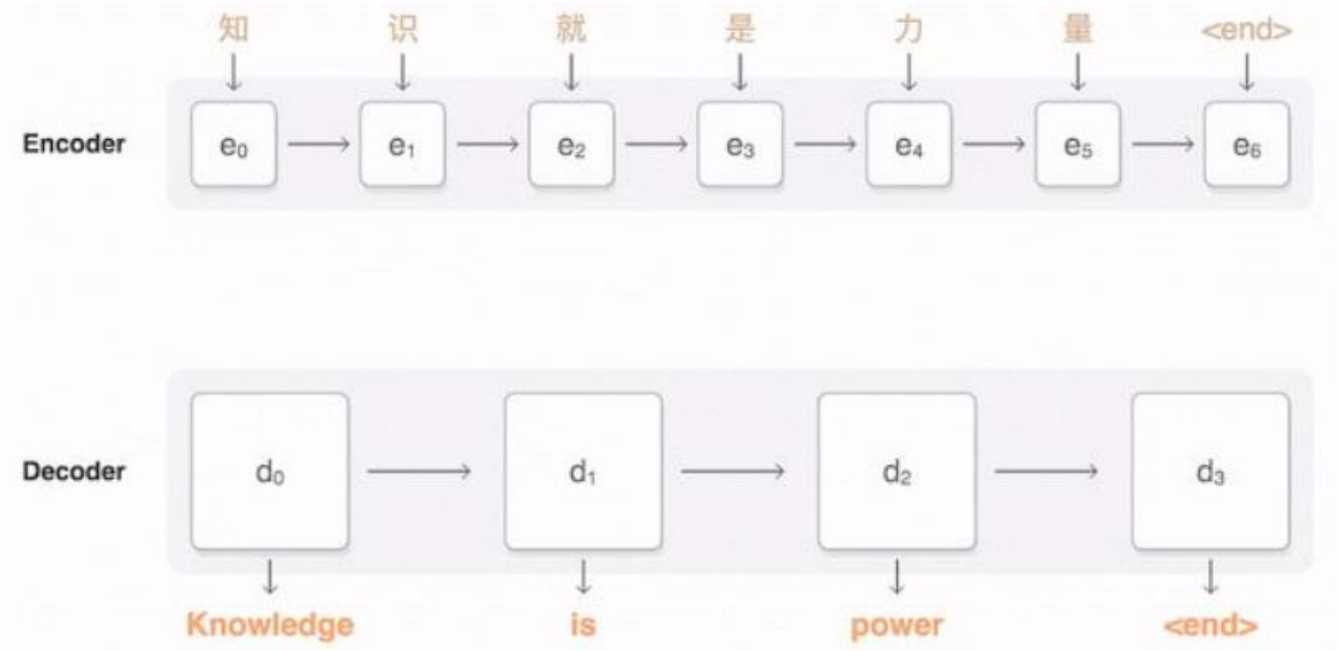

Рисунок 2.9 – Принцип роботи машинного перекладача Google Translate

2.4 Навчання послідовність-в-послідовність з нейронними мережами

Глибокі нейронні мережі (DNN) – це потужні моделі, які досягли успіху у складних навчальних завданнях. Хоча DNN працюють добре, коли доступні навчальні набори з великою міткою, їх не можна використовувати для узгодження послідовностей. У даній роботі представлено загальний підхід до послідовного

навчання, що робить мінімальні припущення щодо структури послідовності. Цей Метод оснований на багатошаровій довго чи короткостроковій пам’яті (LSTM) для зміщення вхідної послідовності за вектором фіксованого розміру, а потім інший глибокої пам’яті LSTM для декодування цільової послідовності з вектора. Нарешті, ми виявили зміну порядку слів у всіх початкових реченнях, що в повній мірі поліпшило продуктивність LSTM. Це призвело до появи нових короткострокових залежностей між вихідним і цільовим пропозиціями, що значно скоротило оптимізацію завдання.

2.4.1 Технічні складності розробки агентів

Розмовні діалогові агенти повинні задовольняти два набори вимог. По-перше, вони повинні забезпечити достатні можливості обробки мови, щоб вони могли вести продуктивні бесіди з користувачами. Вони повинні вміти розуміти питання і заяви користувачів, використовувати ефективні методи управління діалогом і точно реагувати на кожен «розмовний хід». По-друге, вони повинні ефективно працювати на підприємстві. Вони повинні бути масштабованими і надійними, і вони повинні чітко інтегруватися в існуючі бізнес-процеси та інфраструктуру підприємства. Ми обговоримо кожне з цих вимог в свою чергу.

Висновки до розділу 2

Нейронна мережа – це складна статистична система (паралельно функціонує) з хорошою стійкістю до шуму: якщо якийсь блок системи вийде з ладу, мережа в цілому матиме певні скорочення, але навряд чи Моделі, створені нейронними мережами, хоча і дуже ефективні, не можуть бути пояснені людською символічною мовою: результат повинен бути прийнятий «таким, як є» (чорний ящик). Іншими словами – на відміну від алгоритмічної системи – нейронна мережа здатна генерувати дійсний результат (або, принаймні, з великою ймовірністю бути прийнятною), але неможливо пояснити,

як і чому цей результат був створений Як і в будь-якому алгоритмі моделювання, нейронні мережі ефективні лише в тому випадку, якщо змінні предиктора вибираються обережно і не можуть добре мати справу з категоріальними змінними (наприклад, назвою міста) з багатьма різними значеннями. Вони вимагають етапу навчання, який фіксує ваги індивідуальних зв'язків, і це може зайняти деякий час, якщо кількість аналізованих зразків та змінних дуже велика. Немає теорем або моделей для визначення оптимальної мережі, тому успіх мережі дуже залежить від досвіду її виробника.

Діалоговий агент повинен відповісти відповідними текстом. Відповідний текст може в себе включати складання заявок, подання інформації, таких як зображення продуктів, або прийняття подій, таких як ведення електронного журналу інформації в базі даних. Наприклад, якщо метою дії було надати користувачеві замовлення на купівлю та попросити його виконати, агенту потрібно було б отримати або створити заяву типу «Будь ласка, заповніть форму нижче, щоб завершити покупку», або «створіть чи завантажте відповідну веб сторінку».

Мовленнєвий аналіз полягає у визначенні взаємозв'язку між кількома реченнями. Важливою частиною аналізу дискурсу є рішення стандартів, завдання визначення об'єкта, позначеного стандартом, наприклад «Я хотів би купити його зараз». У цьому випадку проблема полягає в інтерпретації трьох пунктів, тобто матеріал вказується в інструкції, але не в контексті діалогу.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Проектування діалогового агенту

Розробки діалогового агента відбувалася на дистрибутиві Linux Ubuntu 17.4. Для функціювання ненииної мережі необхідно встановити наступні бібліотеки:

1) Python 3.5.3 .
2) Tensorflow 0.13.0.
3) cuDNN v6.1.
4) Cuda Toolkit 8.0.

3.2 Операційна система Ubuntu

Ubuntu – це безкоштовний дистрибутив Linux, опублікований у жовтні 2004 року, заснований на класичному Debian GNU / Linux. Засновником проекту з відкритим кодом став південноафриканський підприємець Марк Шаттлворт. Він мріяв розробити операційну систему, яка була доступна практично всім людям, і яка сформувала основну філософію програмного пакету – слово «Ubuntu» походить з мов бантусів на Zulu і Xhosa, і означає «спільнота» або «гуманність». Ubuntu розміщено з більшим набором програмного забезпечення для серверів та робочих станцій.

Система містить усі необхідні додатки – веб-браузер, текстовий редактор, інструменти роботи з електронними таблицями та презентаціями, клієнтами невідомого обміну повідомленнями та багатьма іншими польовими програмами.

Версії LTS, що видаються на 2 роки, термін підтримки 5 років. На інших дистрибутивах LTS сімейства Ubuntu діє повна підтримка протягом 3 років, а для основних систем (ядро, Xorg та прочі компоненти) – 5 років.

Інсталювання дистрибутиву Ubuntu 17.3:

1) Для початку перейти на офіційний сайт Ubuntu.com, в розділі «Завантаження» вибрати останню версію операційної системи.

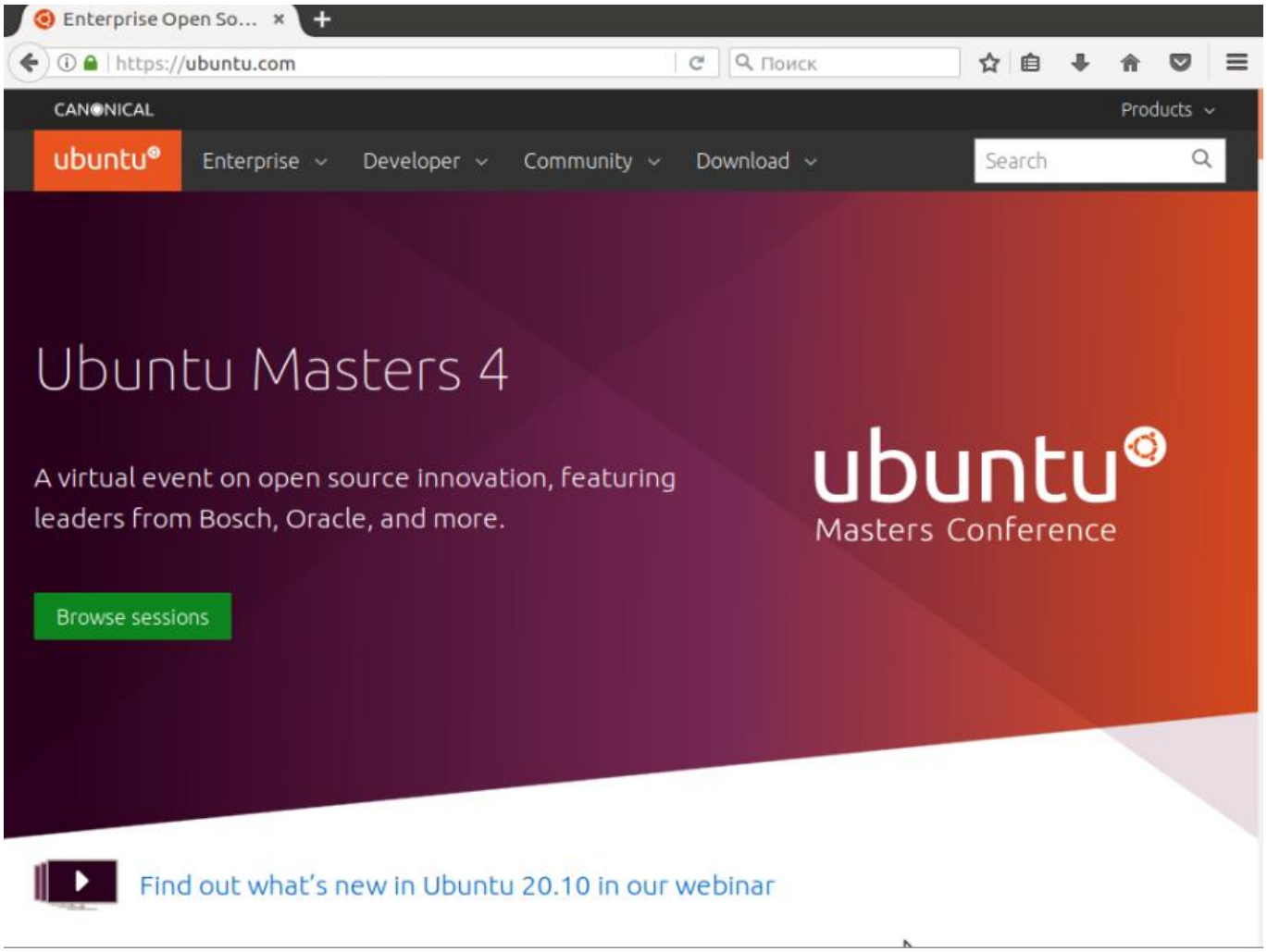

Рисунок 3.1 – Головна сторінка сайту Ubuntu.com

2) Наступний крок це завантаження та інсталяція програми WinSetupFromUSB, для створення завантажувального носія с нашою операційною системою. Завантажити її можна тут http://www.winsetupfromusb.com/downloads/ .

Після завантаження необхідно запустити програму та виконати наступні дії:

1. Вибрати необхідний USB носій.

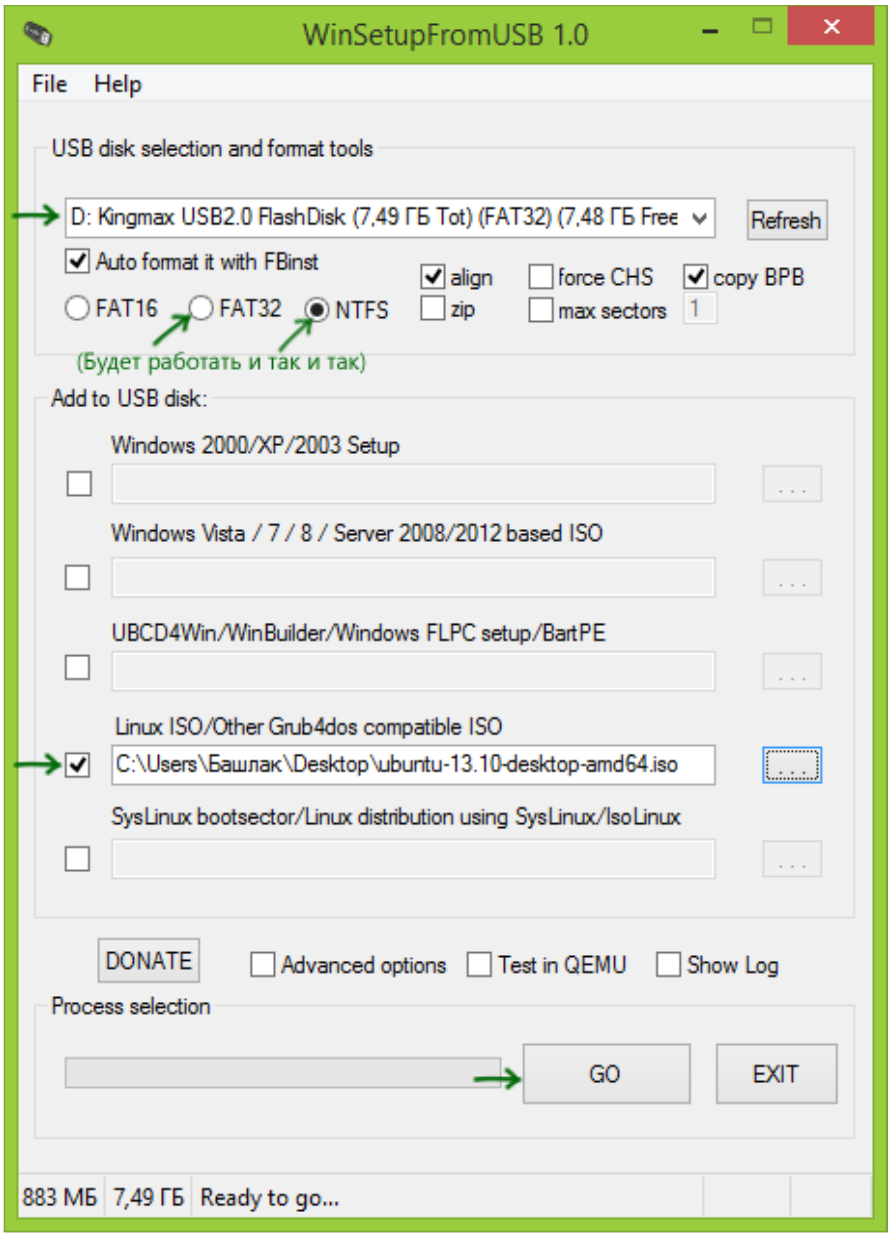

Рисунок 3.2 – Інтерфейс програми WinSetupFromUSB

2. Вибрати пункт *Auto format*.
3. Вибрати пункт *Linux ISO* де вказати шлях до образу диска Ubuntu.
4. З’явиться діалогове вікно, де нам потрібно вказати назву носія, наприклад Ubuntu 17.04

5. Натиснути кнопку «Go», щоб підтвердити, що всі дані з USB накопичувача будуть видалені і почнеться процес створення завантажувальної флешки.

3) Наступним кроком буде налаштування завантажування носіїв операційної системи в меню BIOS.

Після успішного зчитування завантажувального носія, розпочнеться встановлення ОС Ubuntu:

1. Запустити Ubuntu.

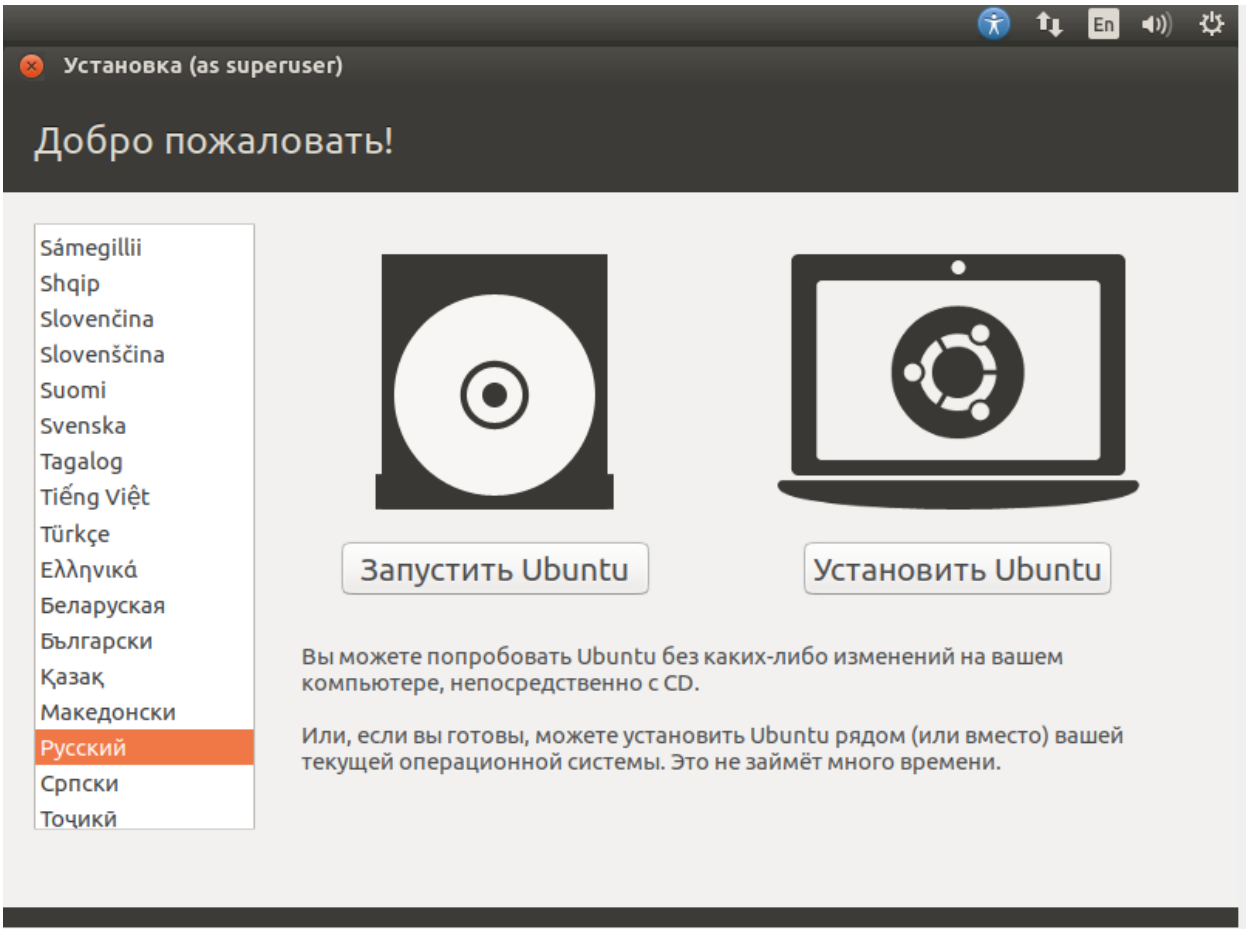

Рисунок 3.3 – Вікно «Ласкаво просимо»

2. Встановити Ubuntu 17.3.

3. Вибираємо пункт «Встановити Ubuntu».

Вибираємо другий варіант, не забуваючи попередньо вибрати мову системи за замовченням.

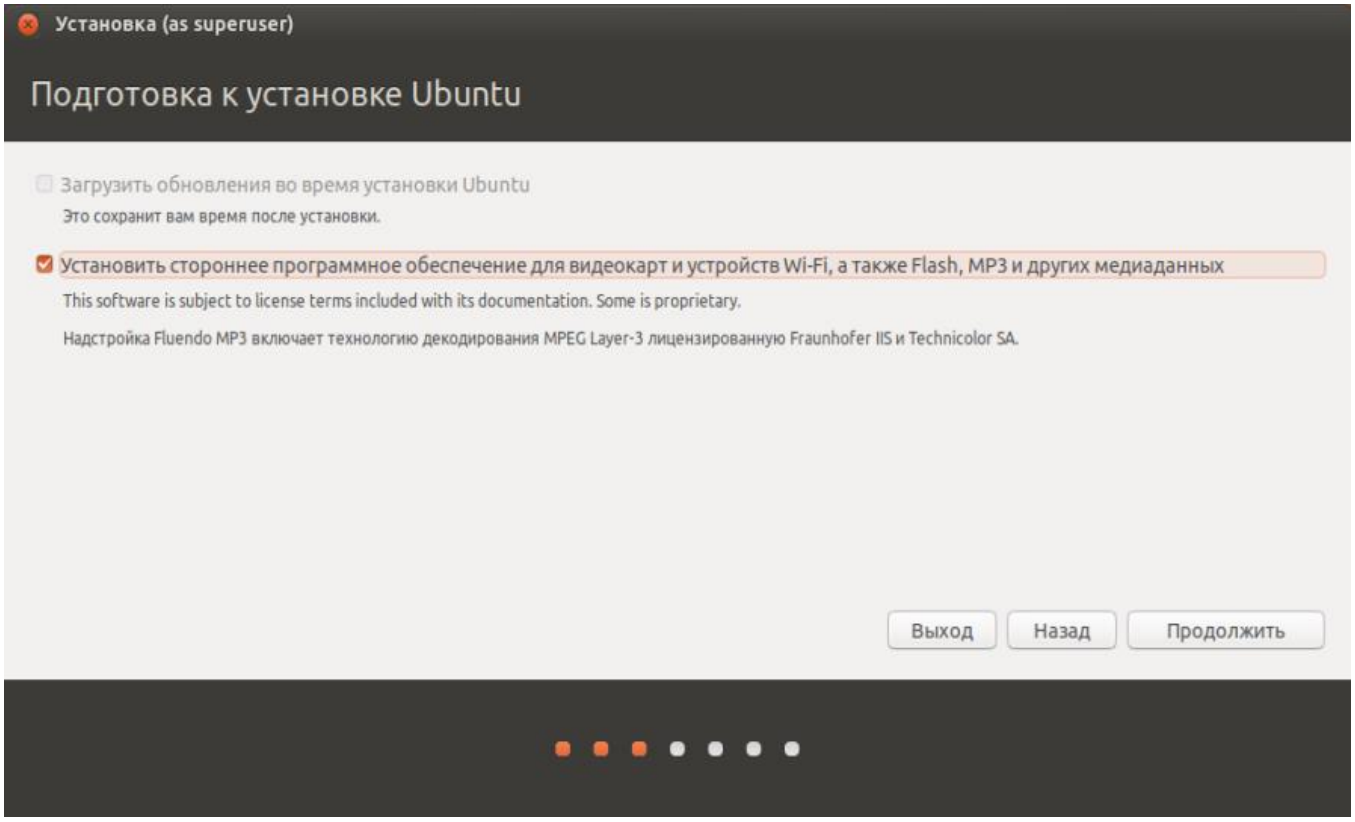

Рисунок 3.4 – Вікно «Підготовка до встановлення ОС Ubuntu 17.3»

Наступне вікно буде називатися «Підготовка до встановлення ОС Ubuntu 17.3». У цьому вікні вам буде запропоновано переконатися, що комп'ютер має досить вільного місця на носії для встановлення системи. Також ви побачите пункт «Встановити стороннє програмне забезпечення для відеокарт та пристроїв Wi-Fi, а також Flash, MP3 чи інших пристроїв».

4) Наступний крок – вибір встановлення Ubuntu:

У наступному вікні будуть відображені розділи вашого носія. Далі нам необхідно визначити розмірність віртуального диску на який буде встановлена наша система Ubuntu. Є можливість самостійно виконати розбивання диску за допомогою

редактора розділів. Але не рекомендовано для не звертатися до цього пункту, якщо ви користувач.

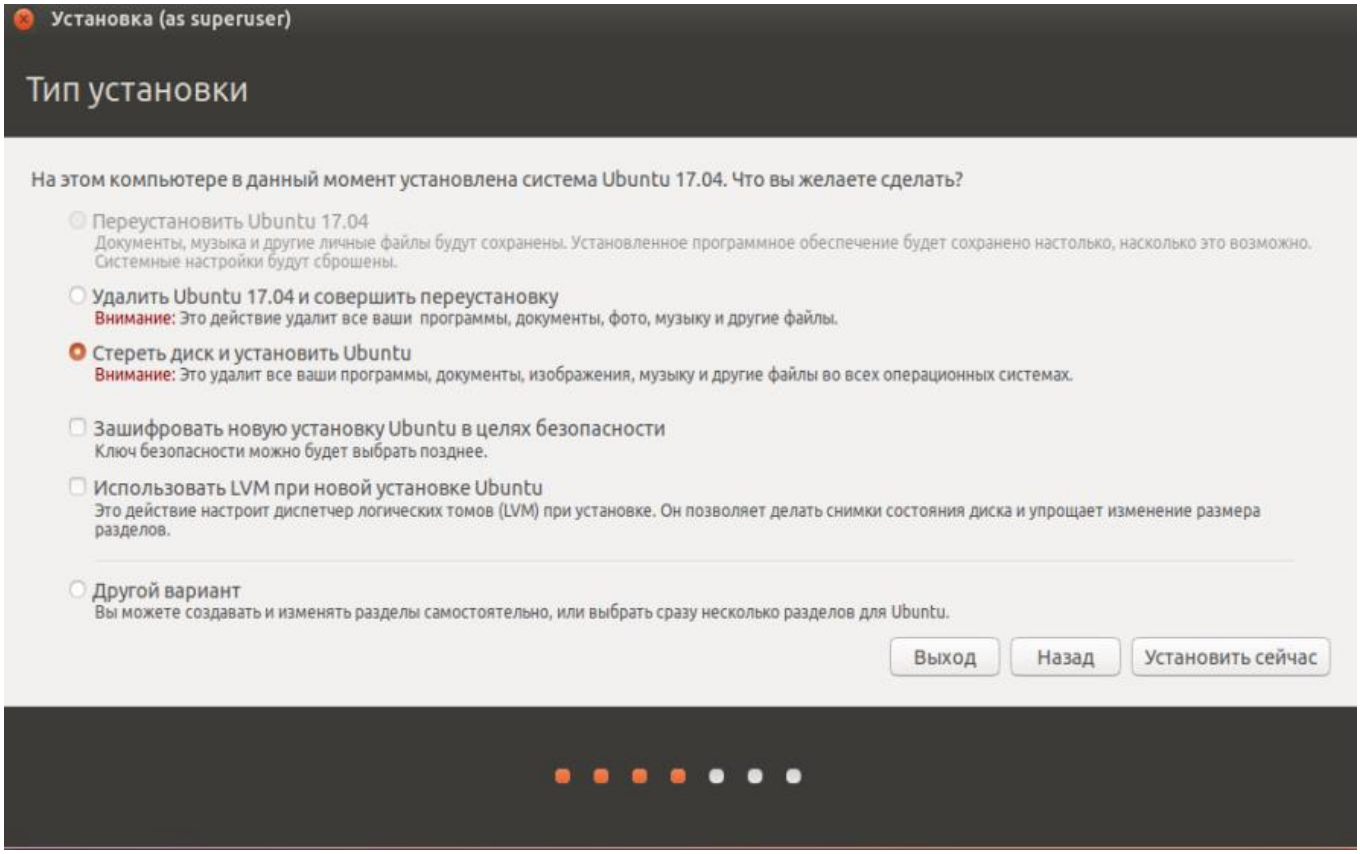

Рисунок 3.5 – Вікно «Встановлення»

На рис 3.5 зображено вікно «Встановлення», де нам потрібно вибрати носій.

Далі нам потрібно тип встановлення операційної системи Ubuntu. Ми вибираємо «Стерти диск і встановити Ubuntu», для того щоб наша система була встановлена на носій після його попереднього форматування. Також можна вибрати пункт «Зашифрувати нову установку Ubuntu з метою безпеки», для того щоб захистити носій за допомогою ключа, котрий ми задаємо пізніше і пункт «Використовувати LVM під час встановлення Ubuntu з метою безпеки». Але ми ці пункти пропускаємо, так я вони є необов’язковими.

Наступним кроком буде налаштування регіонального стандарту для Ubuntu – часовий пояс і розкладку клавіатури.

Рисунок 3.6 – Вікно «Вибору регіона»

На рис 3.7 зображено вікно «Вибір розкладки клавіатури» для налаштування мови за упомовчення для введення.

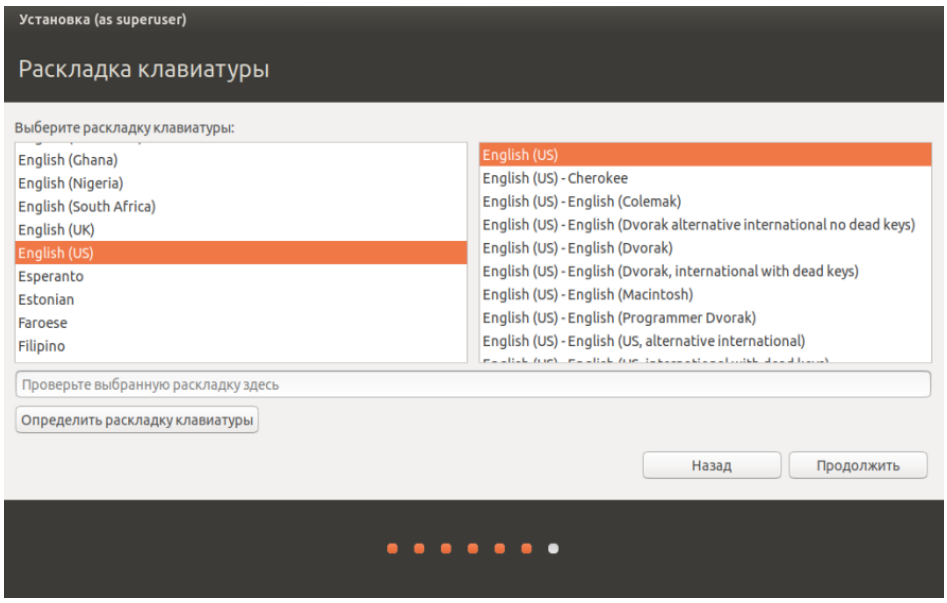

Рисунок 3.7 – Вікно «Вибір розкладки клавіатури»

5) Наступний етап – створення нового користувача і пароля для авторизації в Ubuntu.

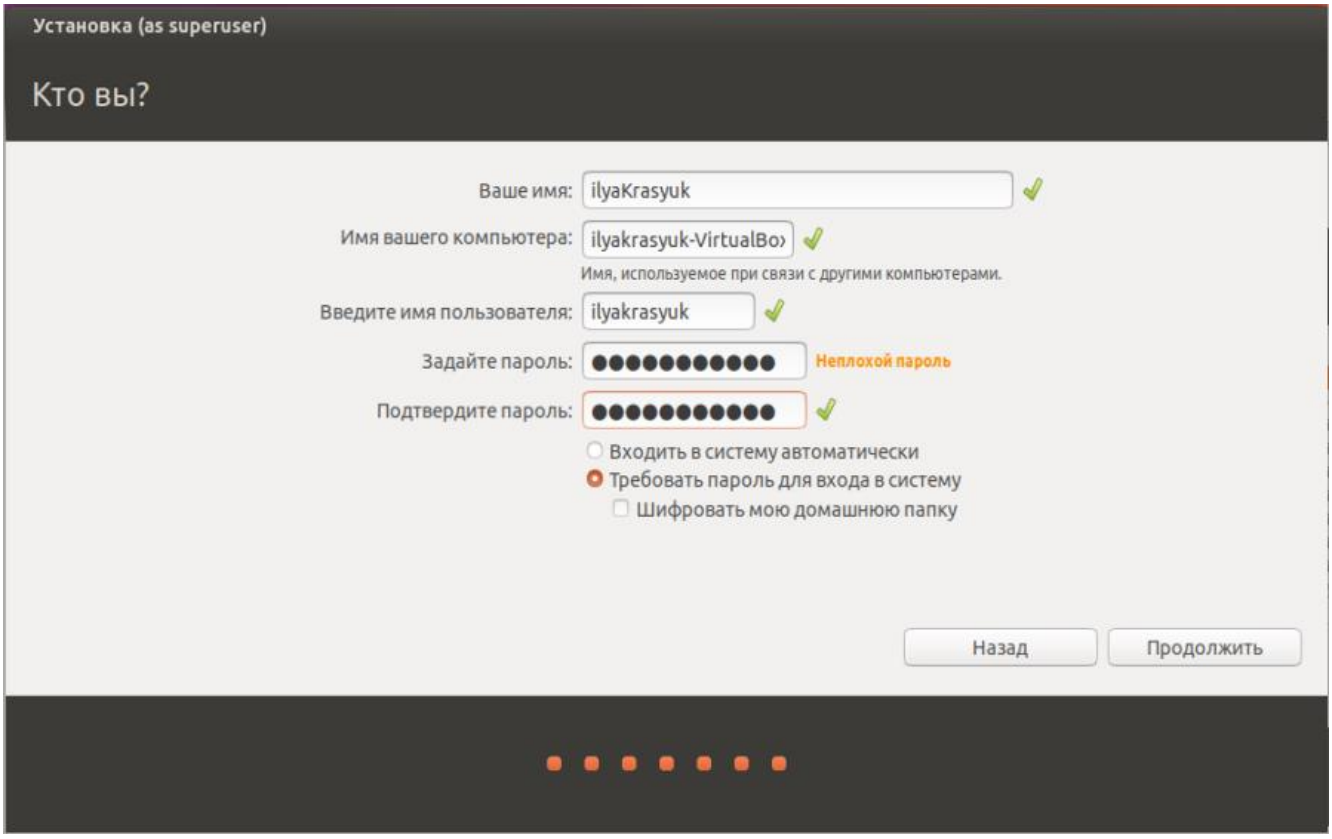

Рис. 3.8. Вікно «Створення нового користувача»

Після заповнення даних, потрібно натиснути на кнопку «Продовжити» і почнеться процес встановлення ОС Ubuntu на комп’ютер. Після деякого проміжку часу з’явиться повідомлення про успішне встановлення і пропозицію перезавантажити комп’ютер

Ubuntu 18.04 Bionic Beaver отримала оновлене ядро з корисними новими функціями, нову версію інтерфейсу Gnome і масу оновлень вбудованих додатків.

За замовчуванням встановлено додаток для ведення нотаток (To Do).

За замовчуванням включена відправка статистики конфігурацій системи на сервери Canonical, відключити яку можна в налаштуваннях приватності.

Ubuntu поставляється з новітніми версіями ядра Linux, в якому реалізований захист від вразливостей Spectre і Meltdown.

6) Після попередніх налаштування почнеться процес встановлення операційної системи.

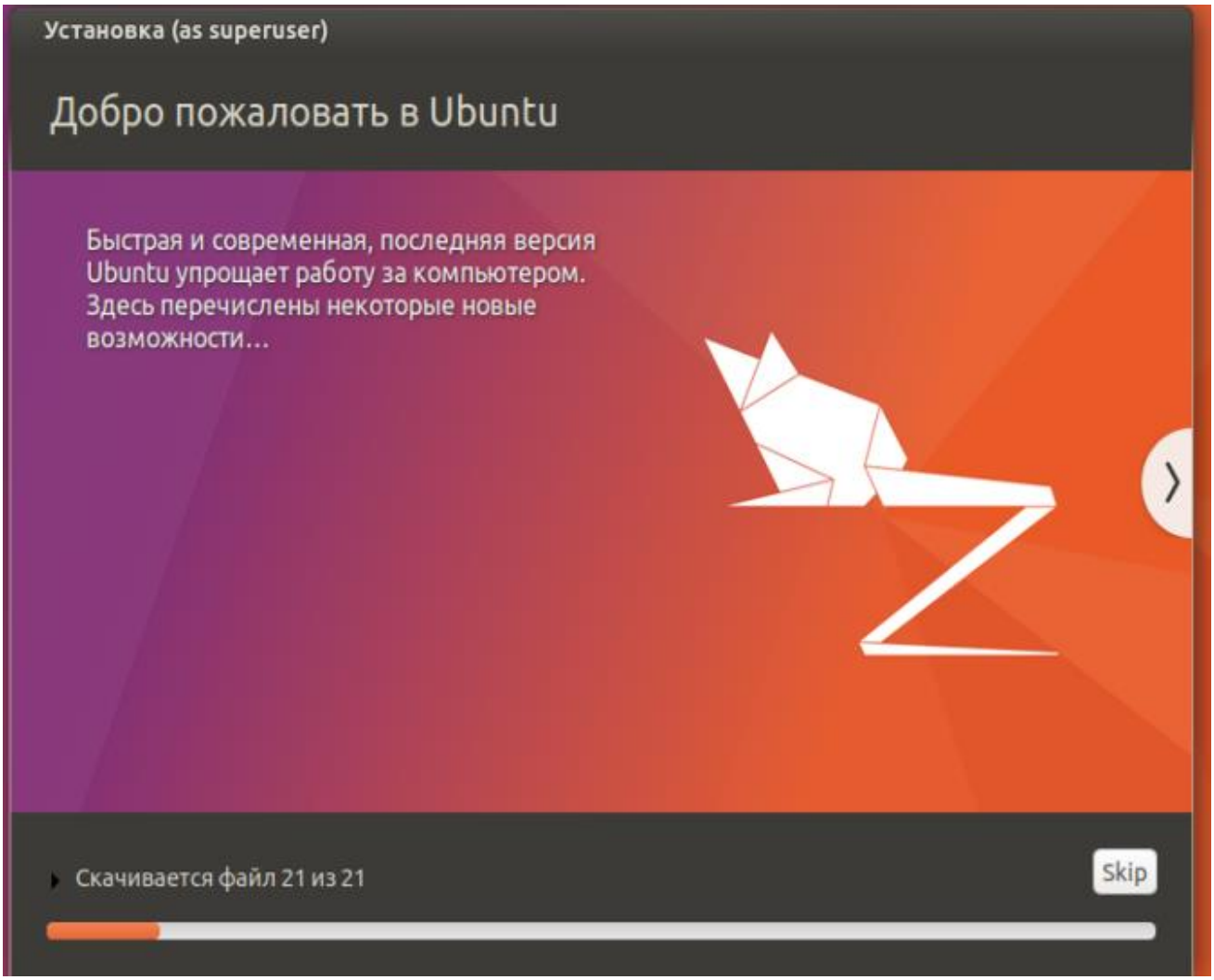

Рисунок 3.9 – Вікно «Встановлення Ubuntu »

У дистрибутиві за умовою використовується протокол графічного сервера X.Org Server, замість очікуваного Wayland. Ймовірно в Ubuntu 20.04 LTS буде повернутий Wayland.

У установці доданий режим встановлення Minimal Install, який дозволяє встановити лише мінімальний набір додатків. Робоче оточення Gnome оновлено до версії

Після встановлення операційної системи Ubuntu з'явиться інтерфейс робочого столу в ОС Ubuntu, що зображений на рис 3.10.

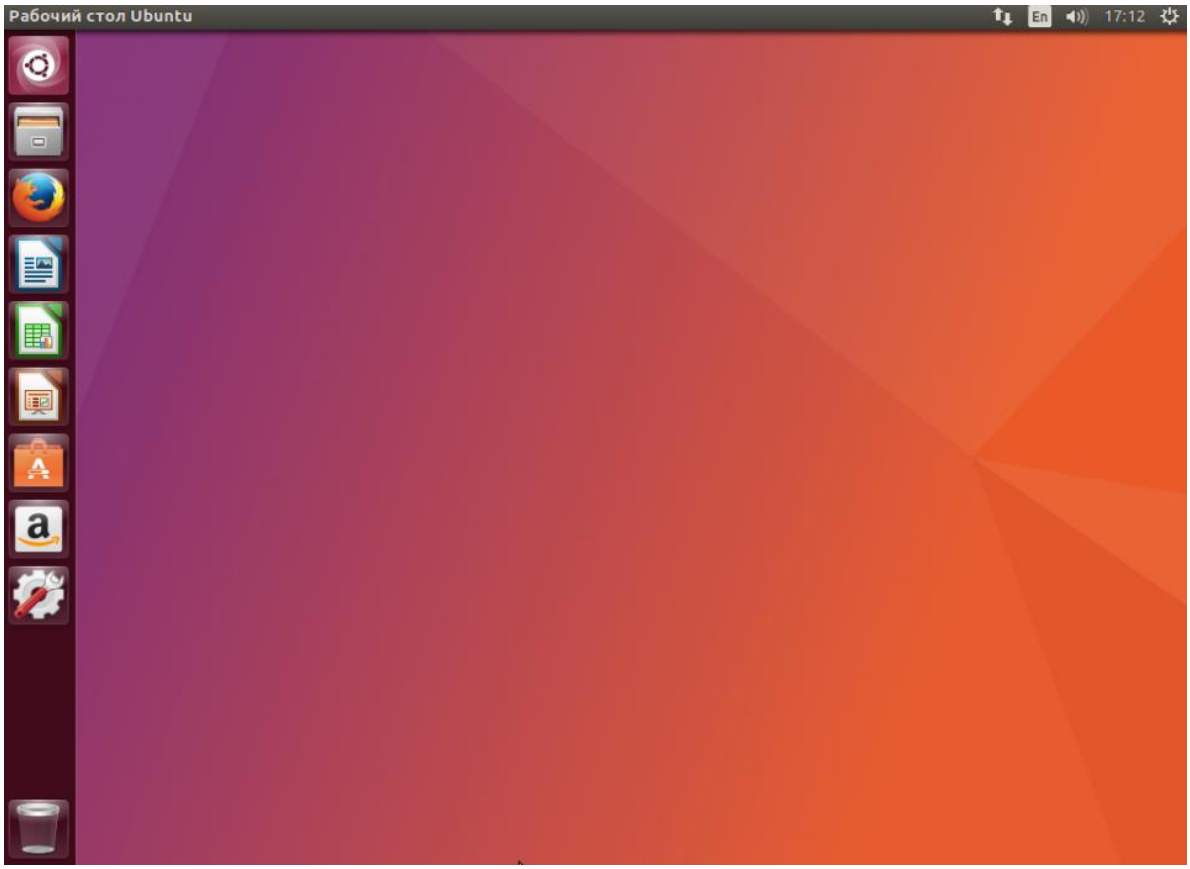

Рисунок 3.10 – Робочий стіл операційної системи Ubuntu 17.4

3.3 Мова програмування Python

Python – популярна високорівнева мова програмування, який призначений для створення додатків різних типів. Це і веб-додатки, і гри, і настільні програми, і робота з базами даних. Досить велике поширення пітон отримав в області машинного навчання і досліджень штучного інтелекту.

Вперше мова Python був анонсована в 1991 році голландським розробником Гвідо Ван Россум. З тих пір ця мова виконав великий шлях розвитку. У 2000 році була видана версія 2.0, а в 2008 році – версія 3.0. Незважаючи на такі великі проміжки між версіями постійно виходять підверсії. Так, поточною актуальною версією на момент написання даного матеріалу є 3.9. Більш детальну інформацію про всіх випусках, версіях і зміни мови, а також власне інтерпретатори і необхідні утиліти для роботи можна знайти на офіційному сайті https://www.python.org/.

Основні особливості мови програмування Python:

1. Скриптова мова. Код програм визначається у вигляді скриптів.
2. Підтримка самих різних парадигм програмування, в тому числі об'єктно-орієнтованої та функціональної парадигм.
3. Інтерпретація програм. Для роботи зі скриптами необхідний інтерпретатор, який запускає і виконує скрипт.

Python – дуже проста мова програмування, яка має лаконічний і в той же час досить простий і зрозумілий синтаксис. Відповідно його легко вивчати, і власне це одна з причин, по якій він є одним з найпопулярніших мов програмування саме для навчання. Зокрема, в 2014 році він був визнаний найпопулярнішим мовою програмування для навчання в США.

Python також популярний не тільки в сфері навчання, але в написанні конкретних програм в тому числі комерційного характеру. У чималому ступені тому для цієї мови написано безліч бібліотек, які ми можемо використовувати.

Крім того, у даного мови програмування дуже велике суспільство, в інтернеті можна знайти по даному мови безліч корисних матеріалів, прикладів, отримати кваліфіковану допомогу фахівців.

Вимоги від системи:

1) Сервер Ubuntu 17.03 або Debian 9.
2) Sudo – доступ користувача.

3.3.1. Інсталяція мови програмування Python 3.5.3

У системах починаючи з Ubuntu 17.03 та Debian 9 мова програмування Python 3/Python 2 встановлені за замовчуванням. Для того щоб оновити пакети системи, необхідно ввести наступну команду, що зображена на рис 3.11.

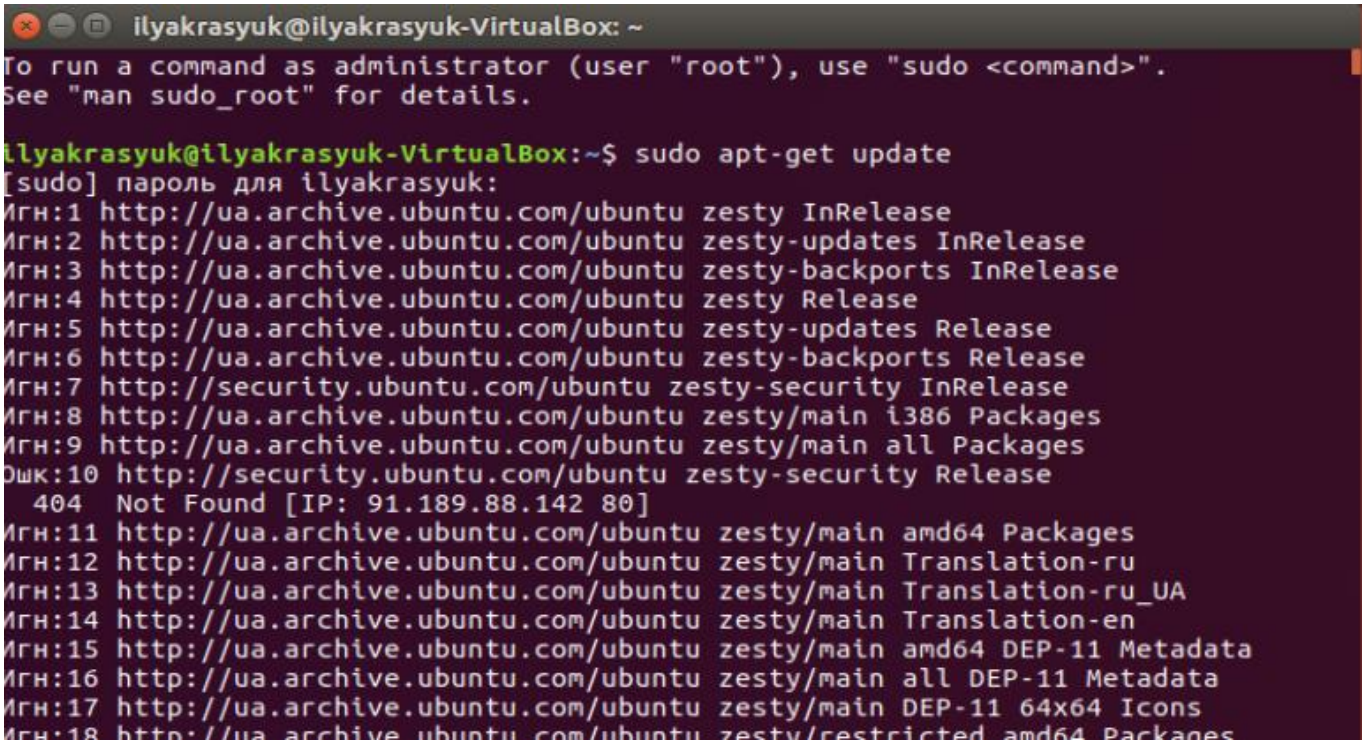

Рисунок 3.11- Оновлення пакетів системи

Для того щоб дізнатися, яка версія Python встановлена в системі, слід ввести наступну команду, що зображена на рис 3.12.

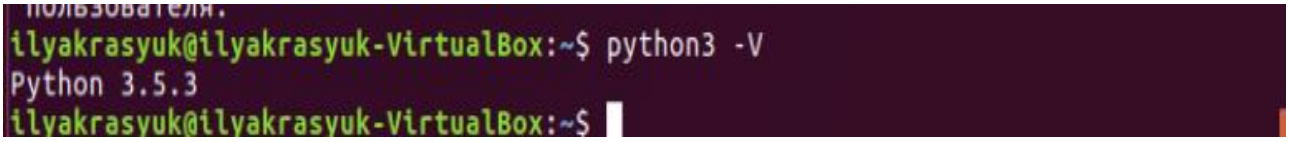

Рисунок 3.12 – Перевірка встановлення Python

У терміналі можна дізнатися номер поточної версії мови програмування Python.

Потім потрібно встановити пакетний менеджер PIP для Python, для цього нам необхідно вести наступну команду в терміналі, що зображена рис. 3.13.

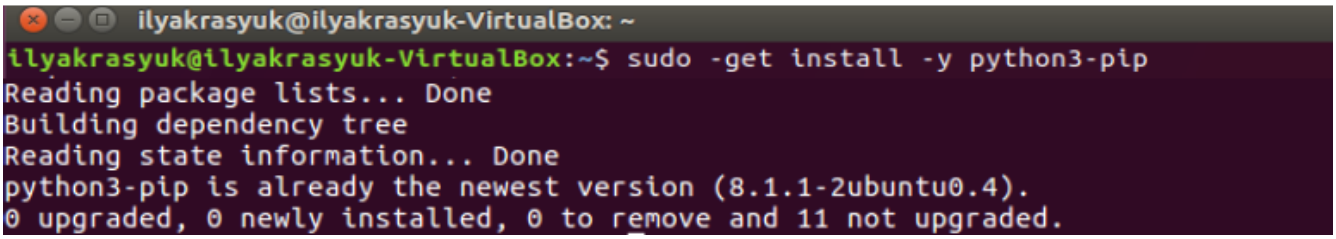

Рисунок 3.13 – Встановлення пакетного менеджера PIP

Менеджер pip дозволяє нам управляти та встановлювати пакети в Python. Для того що встановити пакет, необхідно використати наступний синтаксис:

sudo pip3 install package_name.

Замість *package_name* слід вказати ім’я вашого пакету чи бібліотеки. Для того щоб встановити бібліотеку NumPy, нам необхідно ввести наступну команду, що зображена на рис. 3.14.

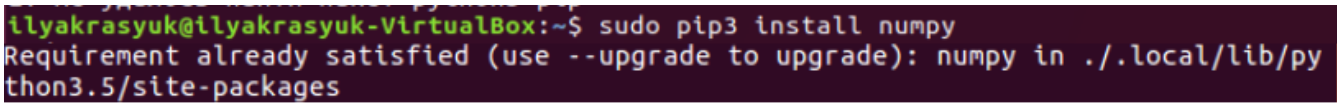

Рисунок 3.14 – Встановлення пакету NumPy

3.4 Бібліотека Tensorflow

Машинне навчання – складна дисципліна. Але впровадження моделей машинного навчання набагато менш страшне і складне, ніж було раніше, завдяки механізмам машинного навчання, таким як Google TensorFlow, які полегшують процес отримання даних, навчальних моделей, обслуговування прогнозів та уточнення майбутніх результатів.

Створений командою Google Brain, TensorFlow – це бібліотека з відкритим кодом для чисельних обчислень та масштабного машинного навчання. TensorFlow поєднує в собі безліч моделей машинного навчання та глибокого навчання (він же нейронних мереж) та алгоритмів, і робить їх корисними за допомогою загальної метафори. Він використовує Python, щоб забезпечити зручний інтерфейсний API для побудови додатків з фреймворком, виконуючи ці програми в високопродуктивному C++.

TensorFlow може навчати та запускати глибокі нейронні мережі для рукописної класифікації цифр, розпізнавання зображень, вбудовування слів, періодичних нейронних мереж, моделей послідовності в послідовність для машинного перекладу, обробки природних мов та моделювання на основі PDE (рівняння часткових диференціальних процесів). Найкраще, TensorFlow підтримує масштабне прогнозування виробництва, використовуючи ті самі моделі, що використовуються для навчання.

Як працює TF? TensorFlow дозволяє розробникам створювати графіки потоку даних – структури, які описують, як дані рухаються через графік або серію вузлів обробки. Кожен вузол на графіку представляє математичну операцію, а кожне з'єднання або ребро між вузлами є багатовимірним масивом даних або тензором.
TensorFlow забезпечує все це для програміста за допомогою мови Python. Python простий у вивченні та роботі з ним, і він пропонує зручні способи висловити, як абстракції високого рівня можна поєднати. Вузли та тензори в TensorFlow – це об'єкти Python, а програми TensorFlow – самі програми Python.

Однак фактичні математичні операції в Python не виконуються. Бібліотеки перетворень, доступні через TensorFlow, записані як високопродуктивні двійкові файли C ++. Python просто спрямовує трафік між фрагментами та забезпечує абстракції програмування високого рівня, щоб з'єднати їх між собою.

Програми TensorFlow можна запускати на будь-якій зручній цілі: на локальній машині, кластері у хмарі, пристроях iOS та Android, центральних процесорах або

графічних процесорах. Якщо ви використовуєте власну хмару Google, ви можете запустити TensorFlow на спеціальному кремнію Google TensorFlow Processing Unit (TPU) для подальшого прискорення. Отримані в результаті моделі, створені TensorFlow, можуть бути розгорнуті на більшості будь-яких пристроїв, де вони будуть використовуватися для обслуговування прогнозів.

TensorFlow 2.0, випущений у жовтні 2019 року, багато в чому переробив структуру, базуючись на відгуках користувачів, щоб полегшити роботу (наприклад, за допомогою порівняно простого API Keras для навчання моделей) та підвищив продуктивність. Розподілене навчання легше запускати завдяки новому API, а підтримка TensorFlow Lite дає змогу розгортати моделі на більшій кількості різноманітних платформ. Однак код, написаний для попередніх версій TensorFlow, повинен бути переписаний – іноді лише незначно, іноді суттєво – щоб максимально скористатися новими можливостями TensorFlow 2.0.

Найбільшою перевагою TensorFlow для розвитку машинного навчання є абстракція. Замість того, щоб займатися дрібницею деталей реалізації алгоритмів або з'ясовувати правильні способи приєднання виходу однієї функції до входу іншої, розробник може зосередитись на загальній логіці програми. TensorFlow піклується про деталі за кадром.

TensorFlow Lite – це набір інструментів, які допомагають розробникам запускати моделі TensorFlow на мобільних, вбудованих та IoT-пристроях. Це дозволяє зробити висновок машинного навчання на пристрої з низькою затримкою та невеликим двійковим розміром.

TensorFlow Lite складається з двох основних компонентів:

1) Інтерпретатор TensorFlow Lite, який використовує спеціально оптимізовані моделі на багатьох різних типах обладнання, включаючи мобільні телефони, вбудовані пристрої Linux та мікроконтролери.

2) Перетворювач TensorFlow Lite, який перетворює моделі TensorFlow в ефективну форму для використання інтерпретатором, і може запровадити оптимізацію для поліпшення бінарного розміру та продуктивності.

TensorFlow Lite розроблений, щоб спростити машинне навчання на пристроях, «на краю» мережі, замість того, щоб надсилати дані вперед і назад із сервера. Для розробників виконання машинного навчання на пристрої може допомогти покращити:

1) Затримка: немає зворотного шляху до сервера.
2) Конфіденційність: дані не повинні залишати пристрій.
3) Підключення: підключення до Інтернету не потрібно.
4) Споживання енергії: мережеві з'єднання потребують енергії.
5) TensorFlow Lite працює з величезним набором пристроїв, від крихітних мікроконтролерів до потужних мобільних телефонів.

Встановлення бібліотеки Tensorflow:

1. Для встановлення та оновлення необхідної версії TensorFlow нам необхідно виконати наступну команду, що зображена на рис. 3.15.

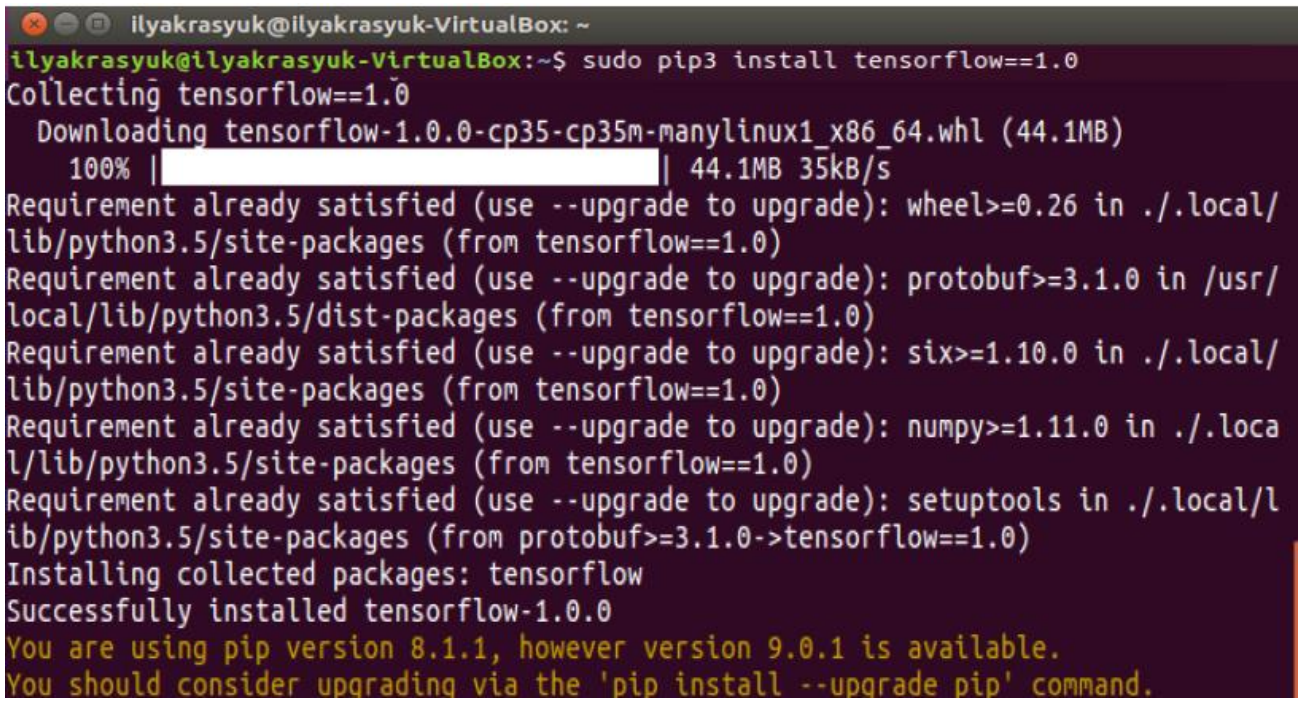

Рисунок 3.15 – Вікно встановлення бібліотеки TensorFlow для CPU

Наступний крок це встановлення бібліотеки TensorFlow для GPU, для цього нам необхідно ввести наступну команду, що зображено на рис. 3.16.

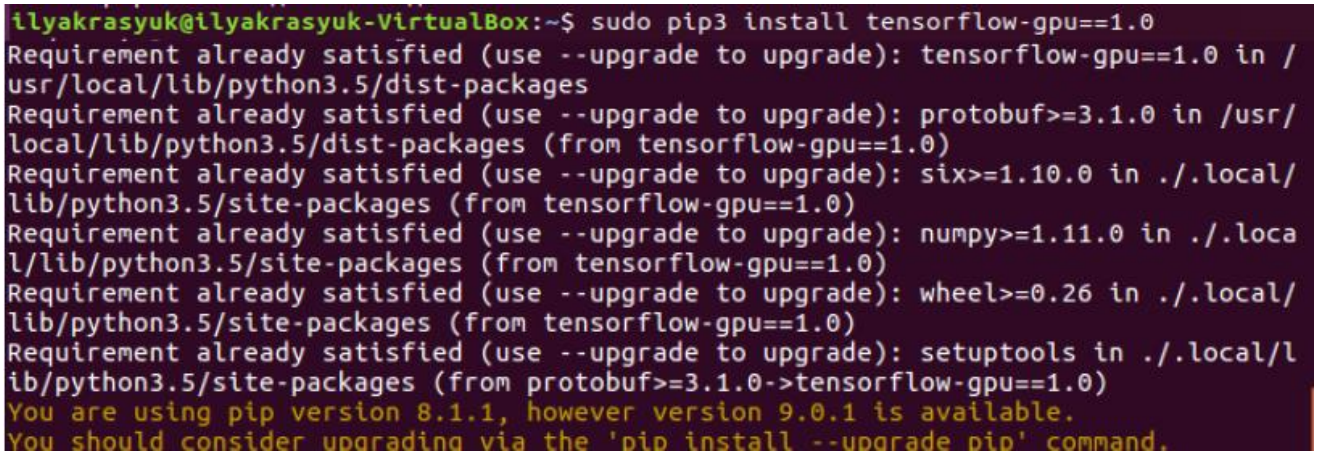

Рисунок 3.16 – Вікно інсталяції TensorFlow-GPU

Після успішного встановлення, нам необхідно переконатись що бібліотека TensorFlow дійсно працює.

2. Перевірка встановлення бібліотеки TensorFlow.

Для того, щоб перевірити чи встановлена бібліотека TensorFlow, нам необхідно скласти програму та запустити її з TensorFlow. Використаємо приклад простої програми «Hello Word!». Не потрібно створювати файл Python, достатньо створити лише програму, за допомогою консолі Python.

Спочатку необхідно запустити інтерпретатор Python, для створення нашої, ввівши наступну команду, що зображена на рис. 3.17.

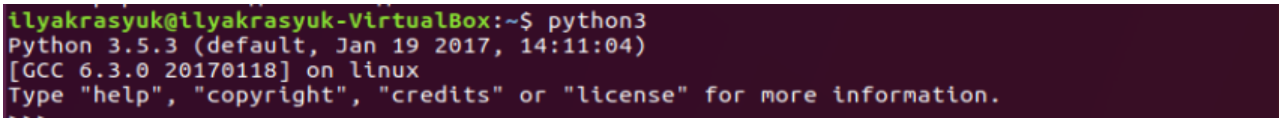

Рисунок 3.17 – Інтерпретатора Python

Після успішного запуску інтерпретатора в терміналі буде наступне повідомлення «>>>».

Це запрошення для інтерпретатора Python, і він вказує на те, що він готовий, щоб почати вводити деякі оператори Python.

По-перше, введіть наступний рядок, щоб імпортувати пакет TensorFlow і зробити його доступним як локальної змінної *tf*. Натисніть ENTER після введення в рядку коду (рис. 3.18).

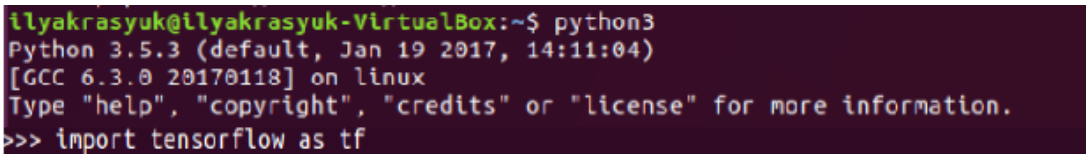

Рисунок 3.18 – Імпорт пакета TensorFlow

Потім необхідно додати наступний рядок коду, щоб вивести повідомлення «Привіт, світ!», зображено на рис. 3.19.

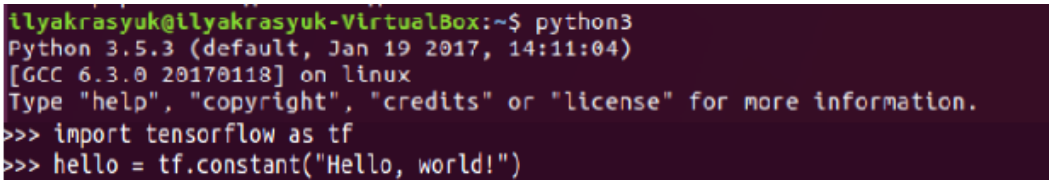

Рисунок 3.19 – Привласнення змінної hello

У будь-якій мові програмування найпростіша програма “Hello World” використовується для демонстрації синтаксису мови програмування. Але ми створюємо програму “Hello World” на Python 3.5.3 як тестову перевірку, що мова програмування вже встановлена в систему.

Серед нових основних нових функцій та змін у серії 3.5 випуску:

- PEP 441, покращена підтримка додатків Python zip;
- PEP 448, додаткові узагальнення розпакування;
- PEP 461, «% -форматування» для байтів і байтових об’єктів;
- PEP 465, новий оператор (@) для множення матриць;

- PEP 471, os.scandir (), нова функція швидкого обходу каталогу;
- PEP 475, додавши підтримку автоматичних спроб перерваних системних дзвінків;
- PEP 479, змініть обробку StopIteration всередині генераторів.

Потім необхідно створити новий сеанс TensorFlow і привласнити його змінної, зображено на рис. 3.20.

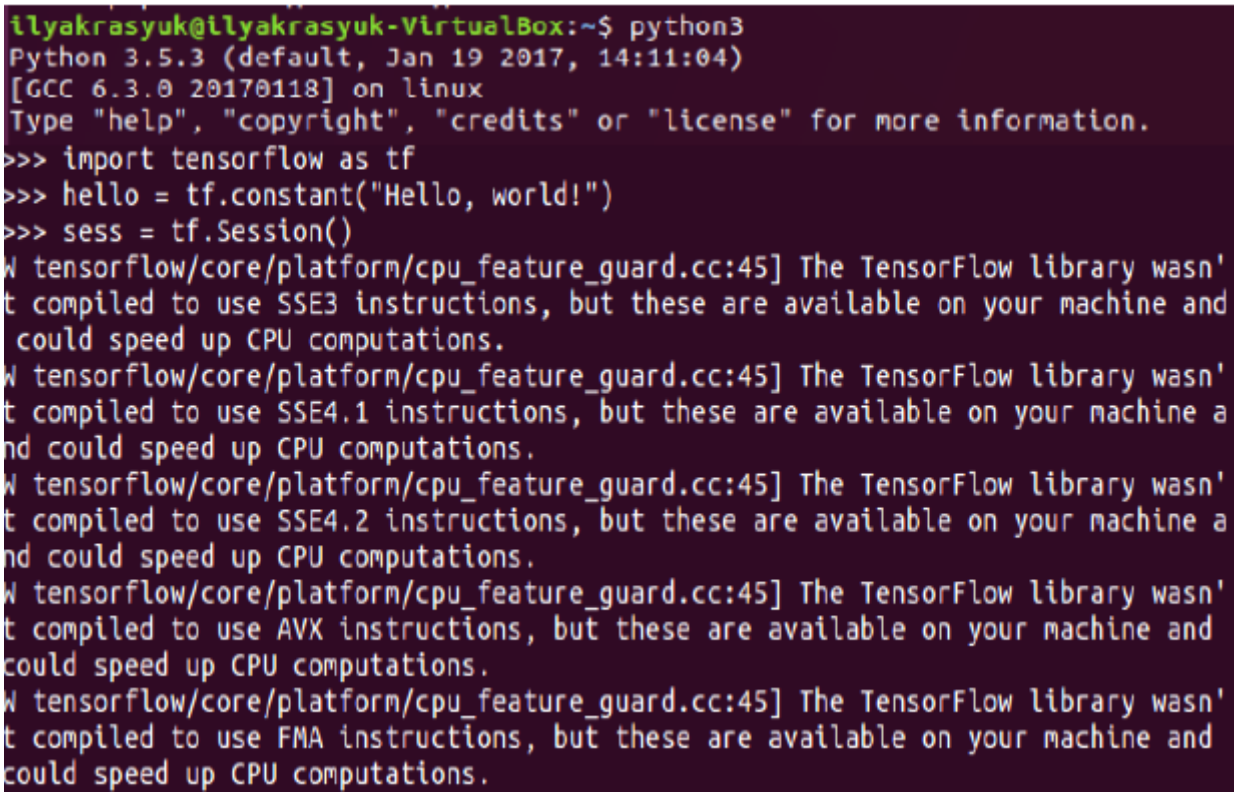

Рисунок 3.20 – Створення нового сеансу TensorFlow

Нарешті, щоб вивести результат запуску сеансу hello в TensorFlow, необхідно ввести такі рядки коду, зображено на рис 3.21

Рисунок 3.21 - Вивід повідомлення «Привіт, світ!»

Це вказує на те, що все працює, і що можна почати використовувати TensorFlow, щоб зробити щось більш цікаве.

3.5 Бібліотека NVIDIA CUDA 8.0

NVIDIA CUDA (Compute Unified Device Architecture) – це сукупність апаратних та програмних засобів, які можуть виробляти на GPU загальні рахунки. В даний час є готовий SDK для програм програмування Fortran, C та C ++

Для скорочення часу прорахунку кадру, можна використовувати потужності графічних процесорів відеокарти, і зробити це досить просто.

Удосконалення сучасних графічних процесорів призвело до того, що потужність відеокарти (GPU) зазвичай перевищує потужність центрального процесора системи (CPU). Мультипроцесор з декількома десятками ядр, становить основу в відеочіпах NVIDIA. Так само, сама карта має швидкісну глобальну пам’ять, з доступом мультипроцесорів, спеціальну пам’ять для констант і власну локальну пам’ять в кожному мультипроцесорі. Основне, що ті самі кілька ядр процесора в GPU є SIMD ядрами (безліч потоків даних і одиночний потік команд). Всі ці ядра виконують однакові інструкції одночасно, даний стиль є звичайним в програмуванні графічних алгоритмів і різних наукових завдань.

Платформа сумісна з 32- і 64-бітними операційними системами Windows XP, Windows Vista, MacOS X і різними версіями Linux. Платформа відкрита і на сайті, крім спеціальних драйверів для відеокарти, можна завантажити програмні пакети CUDA Toolkit, CUDA Developer SDK, що включають компілятор, відладника, стандартні бібліотеки та документацію.

Навчання нейронної мережі можна прискорити за допомогою графічного процесору (GPU). Було прийнято рішення використовувати Tensorflow для навчання нейронної мережі з допомогою GPU.

Процес встановлення бібліотеки NVIDIA CUDA.

1. Встановлення NVIDIA. На рис 3.22 зображено команду для додавання репозиторія.

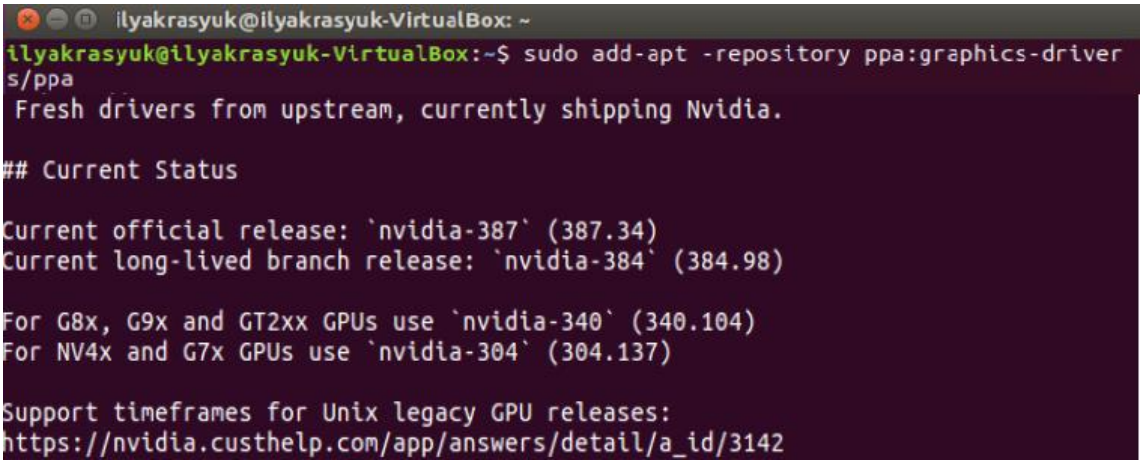

Рисунок 3.22 – Додання репозиторію

Якщо в системі є версія CUDA чи драйвера графічного процесора NVIDIA, то спочатку їх потрібно видалити з системи. Якщо у вас є карта серії GeForce 300 або старіша (випущена в 2010 році або раніше), Nvidia більше не підтримує драйвери для вашої картки. Це означає, що ці драйвери не підтримують поточну версію Xorg. Таким чином, може бути простіше, якщо ви використовуєте драйвер Nouveau, який підтримує старі картки з поточним Xorg.

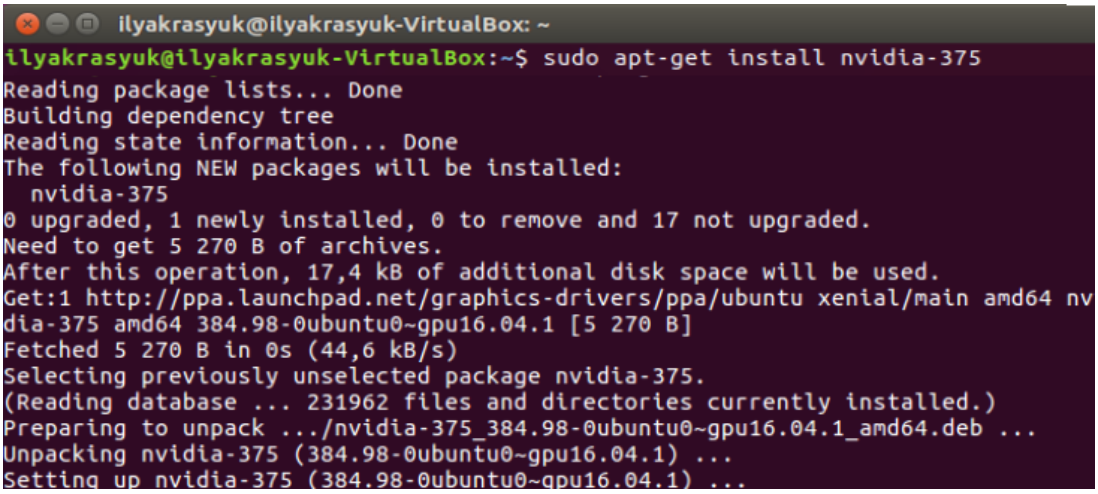

Рисунок 3.23 - Інсталяція драйверу NVIDIA

На рис 3.23 зображено встановлення відео драйверу Nvidia для дистрибутиву Ubuntu.

2. Встановлення CUDA 8.0

Репозиторії Ubuntu містить тільки CUDA 7. Нам потрібна версія CUDA 8, тобто на потрібно встановлювати вручну. Завантаження файлу відбувається за допомогою команди runfile CUDA 9 з офіційного сайту NVIDIA.

Рисунок 3.24 – Розташування NVIDIA CUDA

Для встановлення необхідно запустити файл, виконавши наступною командою:

sudo dpkg -I cuda-repo-ubuntu1604-8-0-local_8.0.44-1_amd64.deb

Бібліотека CUDA 8.0 встановиться в каталог /usr/local/cuda-8.0, також буде створено символічне посилання /usr/local/cuda. Слід використовувати саме це посилання, для швидкої зміни версії бібліотеки CUDA 8.0.

3. Налаштування параметрів середовища

Після того як відбудеться встановлення бібліотеки, нам необхідно прописати шляхи до виконуваних файлів. Створюємо файл за наступним розміщенням /etc/profile.d/cuda.sh і записаємо наступне:

export PATH = $ PATH: / usr / local / cuda / bin

export CUDADIR = / usr / local / cuda

export GLPATH = / usr / lib

Також необхідно створити файл /etc/ld.so.conf.d/cuda.conf для бібліотек з одним рядком:

/ Usr / local / cuda / lib64

для налаштування бібліотек слід викликати команду: *sudo ldconfig*

Процес встановлення завершено, перезавантажуємо комп'ютер для використання бібліотеки CUDA.

3.6 Середовище програмування PyCharm

PyCharm – інтегроване середовище розробки для мови програмування Python. Надає засоби для аналізу коду, графічний відладник, інструмент для запуску юніт-тестів. PyCharm розроблена чеською компанією JetBrains на основі IntelliJ IDEA.

PyCharm працює під операційними системами Windows, Mac OS X і Linux.

Переваги PyCharm

1) Статичний аналіз коду , підсвічування синтаксису і помилок.
2) Навігація серед проектів і сирцевого коду: відображення файлової структури проекту, швидкий перехід між файлами, класами, методами і використання методів.
3) Рефракторінг : перейменування, витяг методу, введення змінної, введення константи, підняття і опускання методу тощо.
4) Виділення помилок в реальному часі.
5) Вбудований відладник для Python;
6) Легка навігація по коду і пошук.
7) Вбудовані інструменти для юніт-тестування.
8) Рефракторінг Python.
9) Документація.
10) Підтримка систем контролю версій: загальний користувацький інтерфейс для Mercurial, Git, Subversion, Perforce і CVS з підтримкою списків змін та злиття.

Репозиторій підтримує різні версії Ubuntu, починаючи від Ubuntu 14.04.

Для додавання репозиторію необхідно ввести наступну команду, що зображена на рис. 3.25.

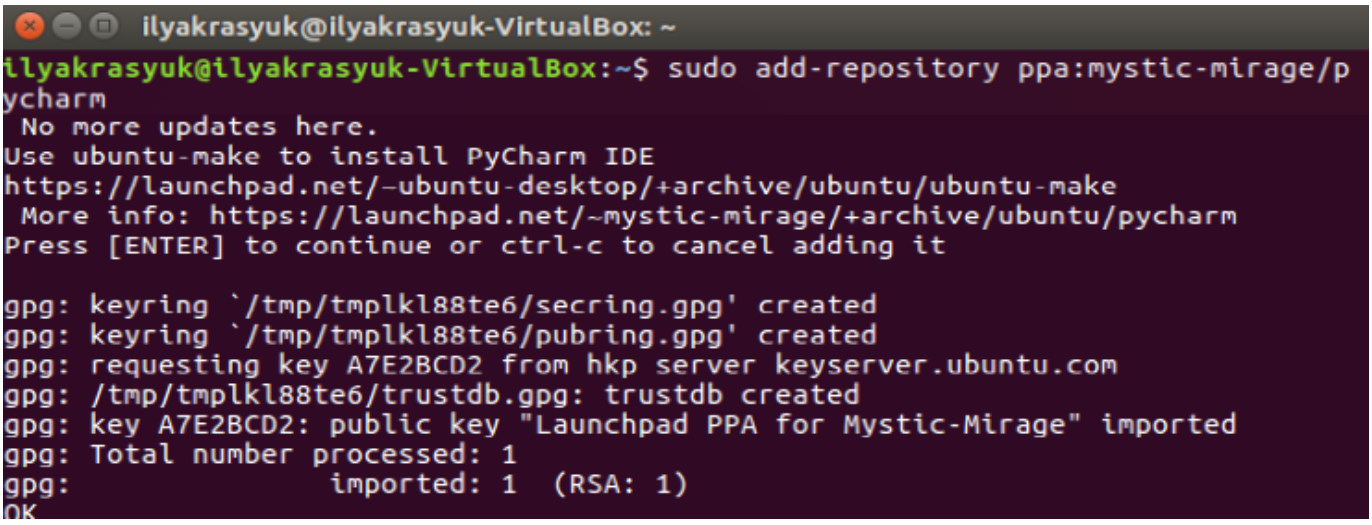

Рисунок 3.25 – Додавання репозиторію де знаходиться PyCharm

Для того, щоб встановити безкоштовну версію, слід ввести наступну команду, що зображена на рис. 3.25.

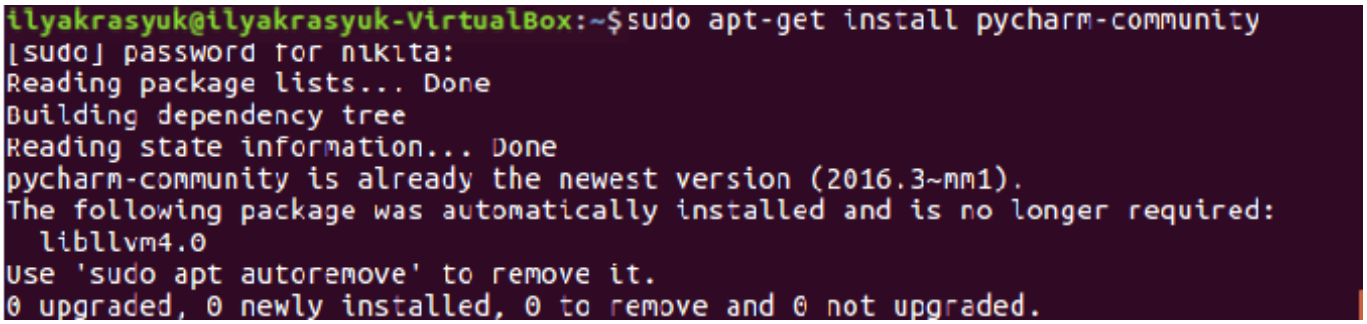

Рисунок 3.26 – Встановлення середовища розробки PyCharm

В процесі встановлення, відбудеться завантаження файлів приблизно 240 Мб даних, після їх розпакування, процес встановлення завершиться і відкриється головне вікно середовища.

На рис. 3.27 зображено головне вікно середовища розробки PyCharm.

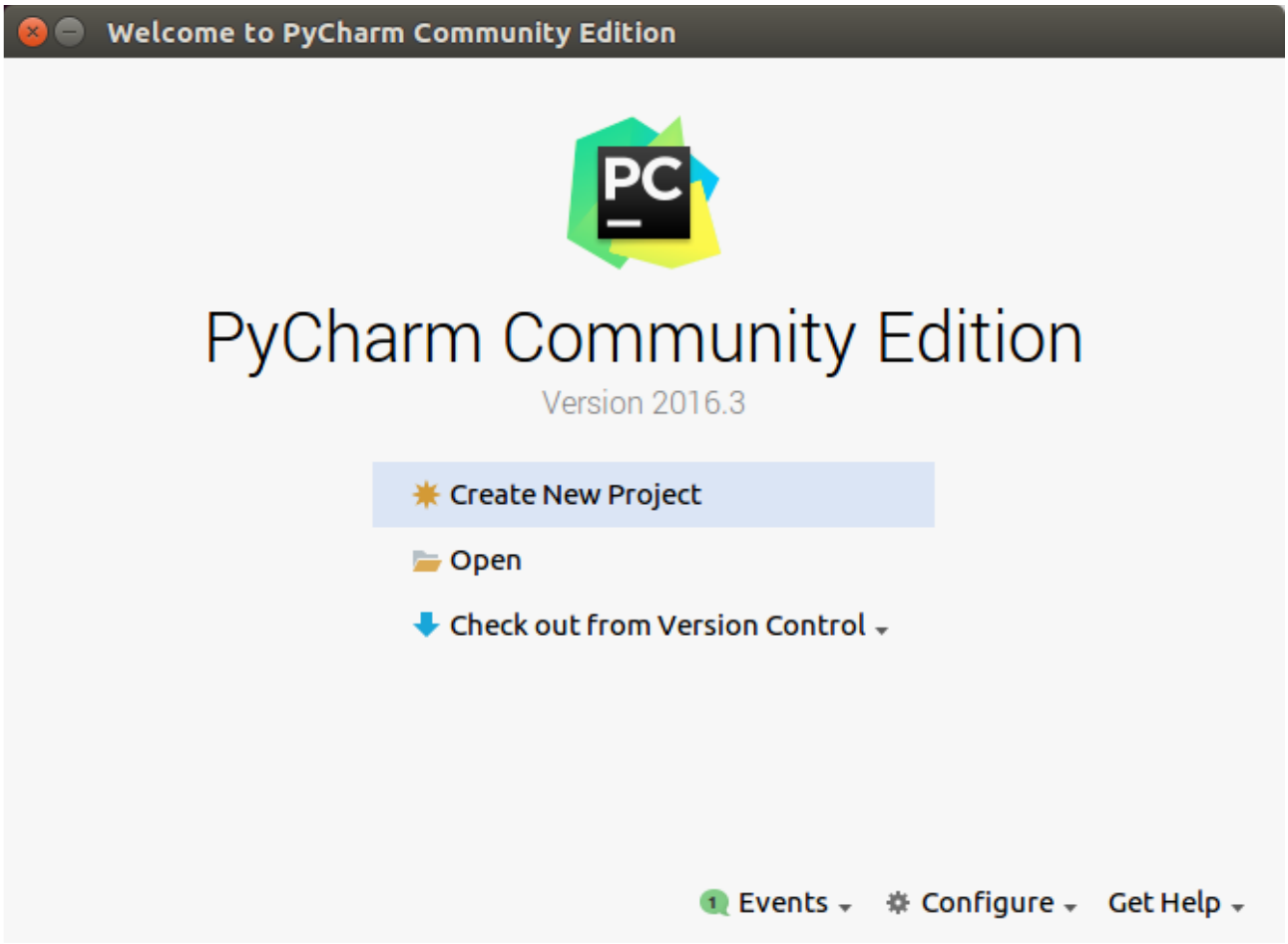

Рисунок 3.27 – Головне вікно середовища розробки PyCharm

3.7 Розробка нейронної діалогової моделі

Для того щоб почати процес навчання моделі потрібно спочатку прописати наступні параметри у скрипт-файлі main.py:

-corpus_opensubs – означає вибір корпусу даних;

-modelTag_OpenSubtitles – вибір моделі яку треба навчити/протестувати.

Для того щоб ввести параметри потрібно спочатку перейти до вкладку Run, а потім Edit Configurations.

На рис. 3.28 зображено інтерфейс редагування параметрів Edit Configurations

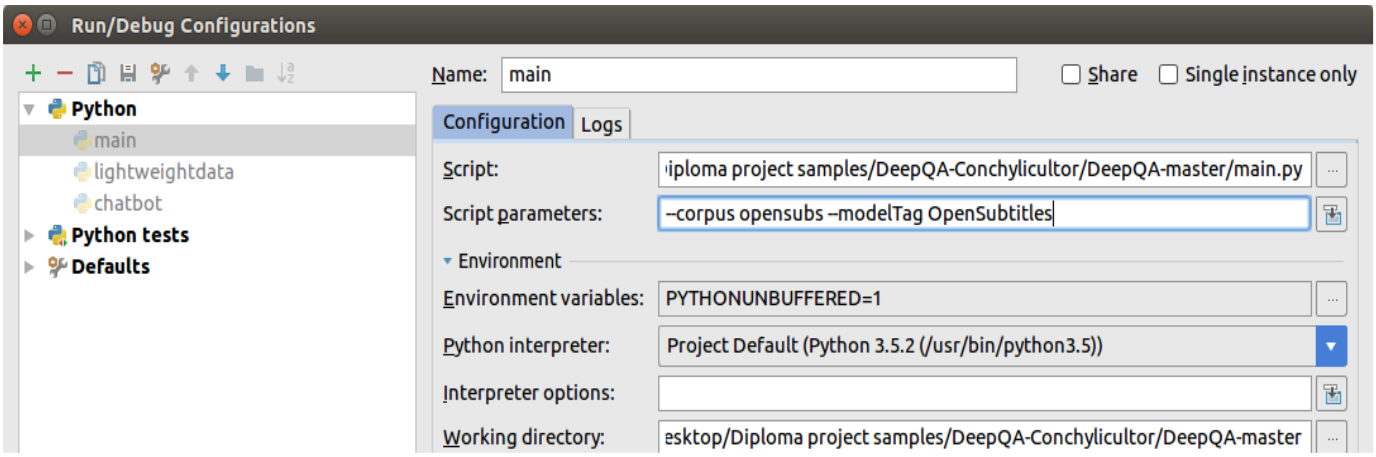

Рисунок 3.28 – Параметри навчання скрипт-файлу main.py

Для підтвердження зміни і збереження та запуску скрипт- файлу необхідно скористатись комбінацією клавіш Alt+Shift+F10 у середовищі програмування PyCharm. Після виконання цих дій розпочнеться процес навчання нейронної моделі, що зображено на рис. 3.29.

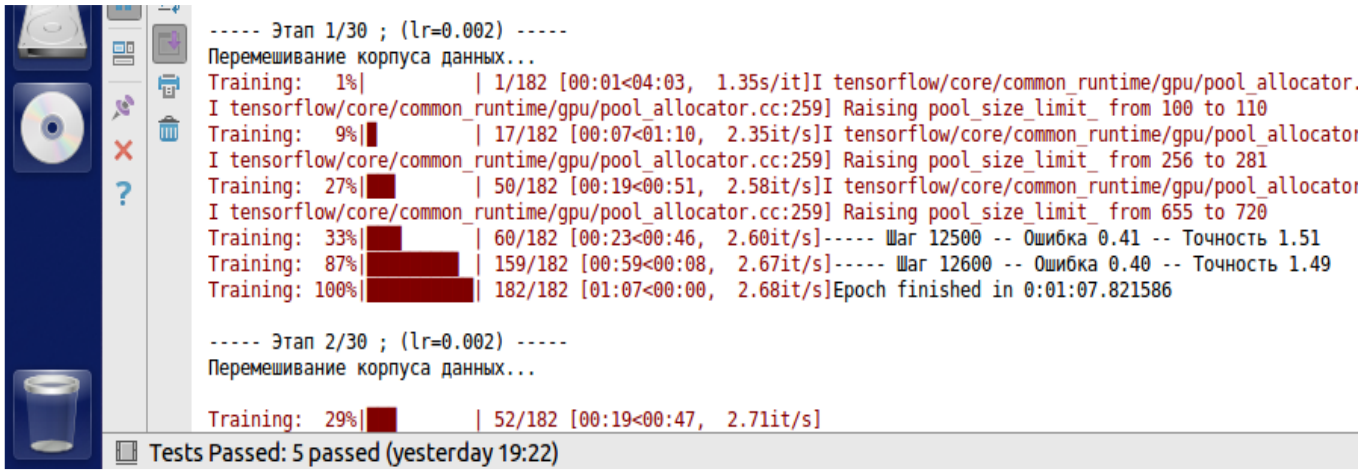

Рисунок 3.29 - Початок навчання діалогової моделі

Для того щоб побачити результат, спочатку необхідно почекати приблизно 5 годин, під час яких модель буде самонавчатися.

На рис 3.30 зображено результат завершення навчання діалогової моделі.

Рисунок 3.30 – Кінець навчання діалогової моделі.

Після того як модель збереглася, треба знову запустити скрипт main.py, але з іншими параметрами:

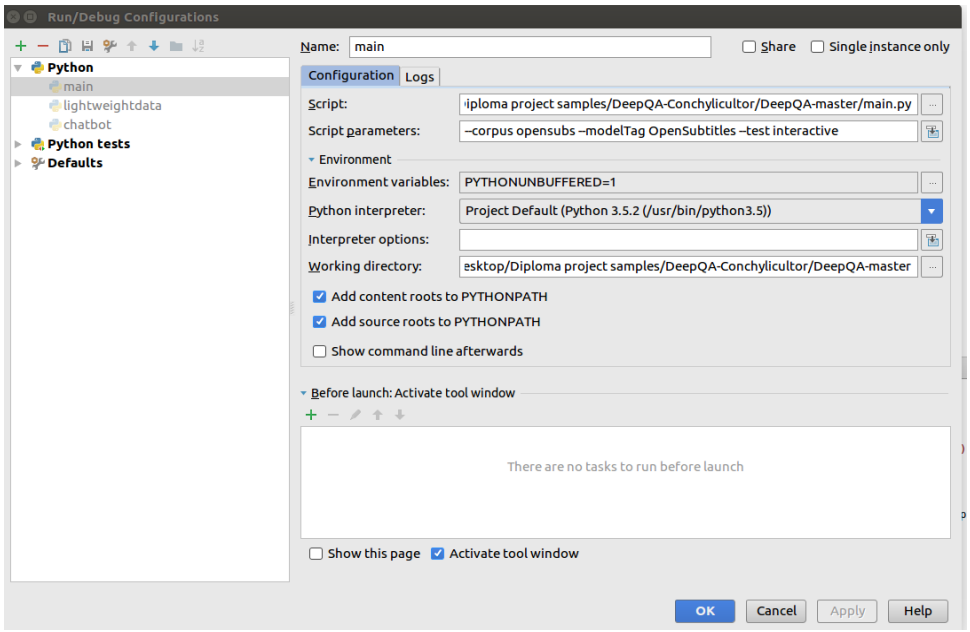

Рисунок 3.31 – Параметри скрипту main.py

Наступний крок- ведення бесіди з розробленою системою, для переконання в роботі алгоритмів закладених в основу цієї моделі, таких як *sequence to sequence*.

Далі наведено тестові результати роботи програми:

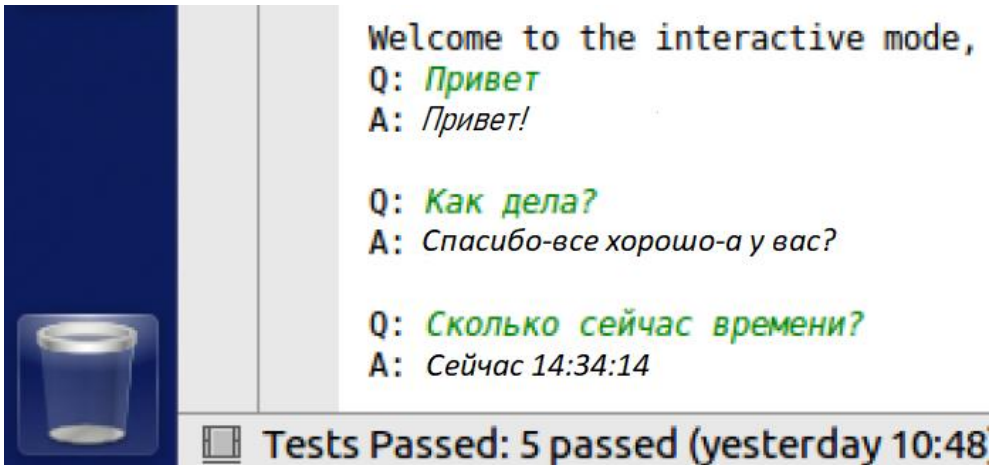

Рисунок 3.32 – Результат роботи програми

На рис 3.32 зображено відповіді на елементарні питання, що для системи не викликає труднощів.

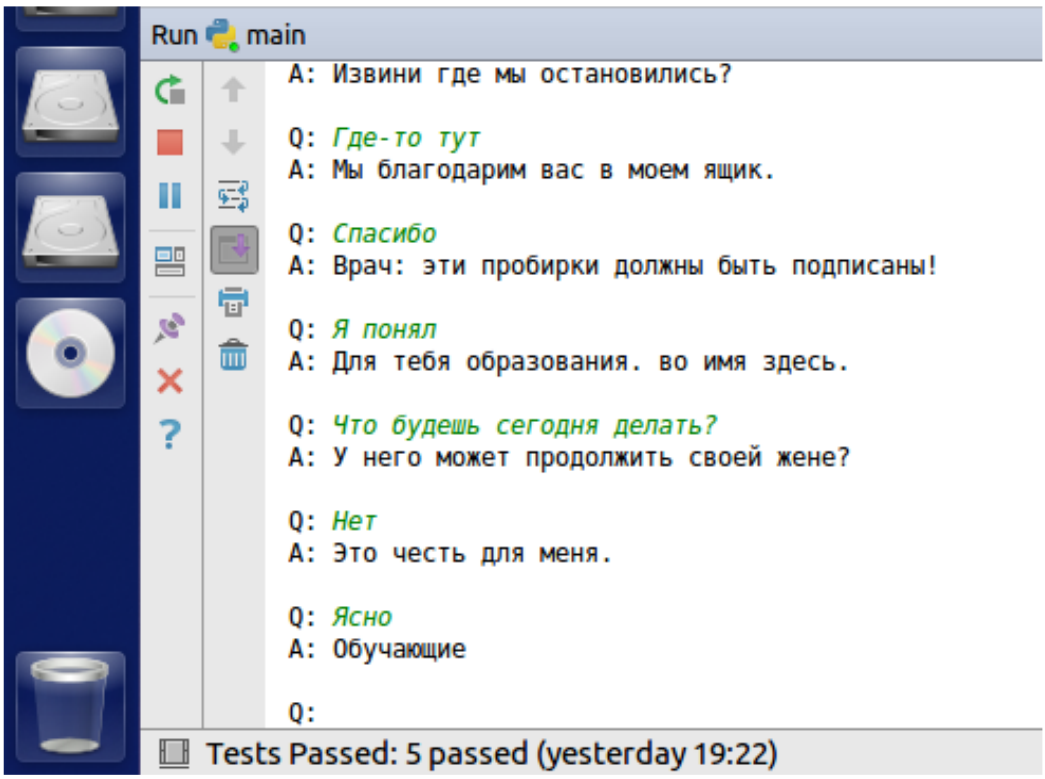

Рисунок 3.33 – Результат роботи програми

У процесі пізнання будь-якого питання описується як вихідне знання, яке виступає його основою, виконуючи роль пропозицій питання. З огляду на судження питання не містить ні затвердження, ні заперечення, тому вираженої в ній інформації не оцінюється в термінах істинності та хибності, тобто питання не говорять як про істинне або хибне.

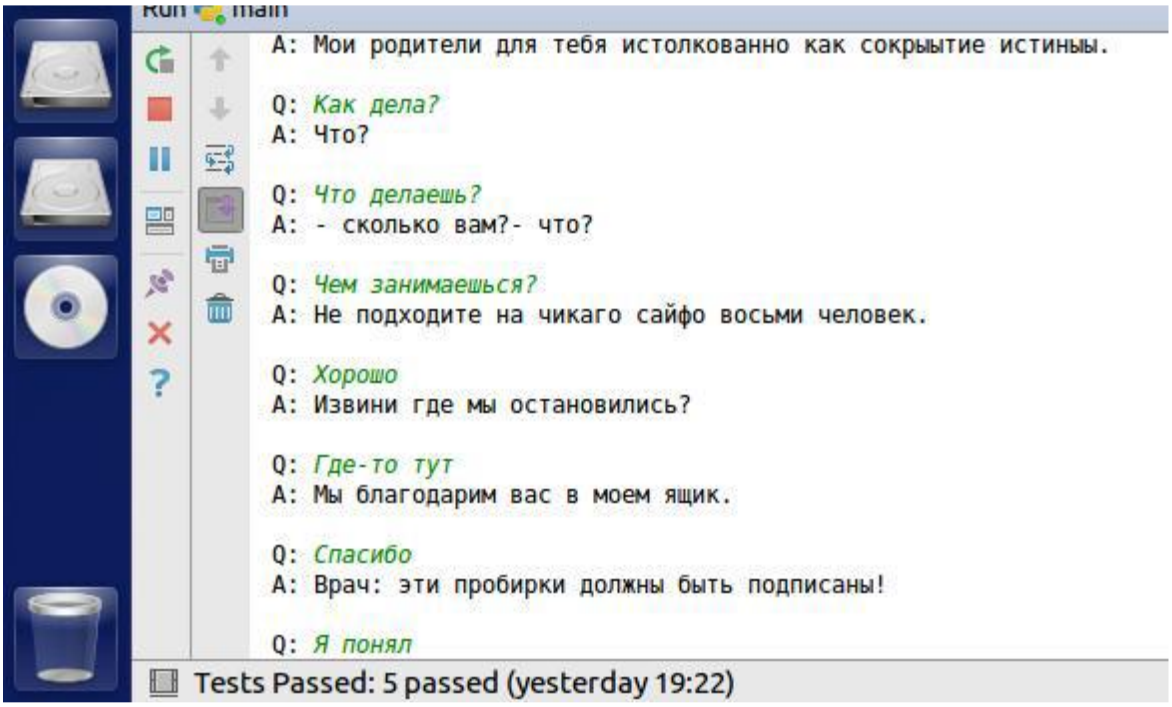

Рисунок 3.34 – Результат роботи програми

За своїм складом питання можуть бути:

1) Простими – називають питання, не включаючи у якісний склад інших питань.
2) Складними – називають питання, включаючи у якісний склад інших питань, об’єднані логічними зв’язками.

У залежності від типу зв’язка, складні питання можуть бути:

1) Сполучними (кон’юнктивними);
2) Розділювальним(диз’юнктивними);
3) Змішаними (сполучно-розділювальними).

Якщо в якості відповіді призводять хоча і справжні, але змістовно не зв'язані з питанням судження, то їх розцінюють як відповіді не по суті питання і зазвичай виключають з розгляду. Поява таких відповідей в дискусії коли відповідач не вловив сенсу питання, але все ж намагається якось відповідати на нього.

Серед відповідей розрізняють наступні види:

1) Справжні і несправжні;
2) Прямі і непрямі.
3) Короткі ж розгорнуті.
4) Повні і неповні.
5) Точні (певні) і неточні (невизначені).

По відношенню до дійсності, відповіді можуть бути істинними або помилковими. Відповідь розцінюється як істинна, якщо вона виражає в ньому судження правильно, або адекватно відображає дійсність. Відповідь розцінюється як помилкова, якщо вона виражає в ньому судження невірно, або неадекватно відображає стан справ в дійсності.

Висновки до розділу 3

В даному розділі розглянуто всі аспекти програмної реалізації розробки нейронної діалогової моделі. Операційна система розробки Ubuntu 17.3, розкриті всі нюанси а також показано детальна інструкція встановлення ОС. Мова програмування- Python 3.5.3, освітлені всі деталі та аспекти даної мови програмування, а також було встановлено в ОС Ubuntu 17.3. Для реалізації нейронної мережі були встановлені необхідні інструменти, такі як: пакетний менеджер PIP для Python 3.5.3, бібліотека нейронної мережі Tensorflow, та інструменти для використання ресурсів графічного процесора NVIDIA CUDA 8.0. Також було встановлено найефективніше середовище розробки PyCharm для мови програмування Python. Розроблено рекуренту нейронну діалогову модель та інтегровано у вигляді чат-боту.

4 ЕКОНОМІЧНА ЧАСТИНА

Під час розробки дипломного проекту був проведений аналіз систем на основі нейронної мережі. На основі отриманої інформації було спроектована та реалізована діалогова модель, в основі якої лежить рекурентна нейронна мережа, для ведення діалогу з ШІ. Система відлагоджена, може вести осмислену бесіду та самонавчатися.

Показникові дані, для розрахунку економічної складової проекту:

1) Число операторів – 960.
2) Показник складності програми – 1,4.
3) Показник корекції програми – 0,2.
4) Показник збільшення витрат праці – 1,3.
5) Показник кваліфікації програміста – 0,9.
6) Середня заробітна плата програміста за годину – 40 грн/год.
7) Вартість використання ЕОМ – 15 грн/год.

4.1 Розрахунок трудомісткості розробки програмного забезпечення

Трудомісткість розробки програмного забезпечення розраховується за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д \text{ , людино- години, де:} \quad (4.1)$$

t_o – витрати на підготовку поставленої задачі (за замовченням – 50);

t_u – витрати на дослідження алгоритму рішення задачі;

t_a – витрати на розробку блок-схеми алгоритму;

t_n – витрати на програмування по створеній блок-схемі;

$t_{отл}$ – витрати на налагодження ПЗ на ЕОМ;

$t_д$ – витрати на підготовку документації проекту.

$$t = 50 + 34 + 96 + 96 + 513 + 228 = 1017 \text{ (людино} - \text{годин)} \quad (4.2)$$

Витрати праці вираховується через умовний показник операторів у програмному забезпеченні.

Умовний показник операторів:

$Q = q \cdot C \cdot (1 + p)$, де (4.3)

q – очікуване число операторів;

C – показник складності програми (1.25…2.0);

p – показник корекції програми під час її розробки (0.05…1).

$Q = 940 \cdot 1{,}4\,(1 + 0{,}1) = 1344$ (4.4)

Витрати на вивчення постановки завдання t_u визначається з урахуванням кваліфікації програміста:

$t_u = \frac{Q \cdot B}{(75 \ldots 85) \cdot k}$, людино − годин, де: (4.5)

B – показник збільшення витрат через недостатнього опису завдання (1.2…1.5);

k – показник кваліфікації програміста, залежно від досвіду роботи в даній галузі.

Таблиця 4.1 – Показники кваліфікації програміста

Стаж програміста	Значення k
до 2-х років	0,7
від 2 до 3 років	0,9
від 3 до 4 років	1 … 1,1
від 4 до 7 років	1,1 … 1,2
понад 7 років	1,2 … 1,5

$t_u = \frac{1344 \cdot 1{,}2}{80 \cdot 1{,}2} = 25$ (людино − годин) (4.6)

Витрати на проеткування та розбку алгоритму вирішення задачі:

$$t_a = \frac{Q}{(20...25)\cdot k}, \text{людино} - \text{ годин} \quad (4.7)$$

$$t_a = \frac{1334}{23\cdot 0{,}8} = 73 \text{ (людино} - \text{годин)} \quad (4.8)$$

Витрати на розбку програми при розробленій блок-схемі:

$$t_n = \frac{Q}{(20...25)\cdot k} \quad (4.9)$$

$$t_n = \frac{1344}{22\cdot 0.8} = 76 \text{ (людино} - \text{годин)} \quad (4.10)$$

Витрати налагодження програми :

1. За умови автономного коригування завдання:

$$t_{\text{отл}} = \frac{Q}{(4...5)\cdot k}, \text{людино} - \text{ годин} \quad (4.11)$$

$$t_{\text{отл}} = \frac{1344}{4\cdot 0{,}8} = 420 \text{ (людино} - \text{ годин)} \quad (4.12)$$

2. За умови всебічного коригування завдання:

$$t_{\text{отл}}^k = 1{,}5 \cdot t_{\text{отл}}, \text{людино} - \text{ годин} \quad (4.13)$$

$$t_{\text{отл}}^k = 1{,}5 \cdot 420 = 630 \text{ (людино} - \text{ годин)} \quad (4.14)$$

Витрати на підготовання документації:

$$t_\partial = t_{\partial p} + t_{\partial o}, \text{людино} - \text{ годин} \quad (4.15)$$

$$t_\partial = 130 + 98 = 228 \text{ (людино} - \text{ годин)} \quad (4.16)$$

де t_{do} – складність редагування, друку та збереження документів

$$t_{\partial o} = 0{,}75 \cdot t_{\partial p}, \text{людино} - \text{ годин} \quad (4.17)$$

$$t_{\partial o} = 0{,}75 \cdot t_{\partial p}, \text{людино} - \text{ годин} \quad (4.18)$$

4.2 Оцінка вартості програмного забезпечення

Витрати на створення програмного забезпечення $К_{ПО}$ включають вартість заробітної плати виконавців програми $З_{ЗП}$ та вартість машинного часу, необхідного для налаштування програми на комп'ютері.

$$К_{ПО} = З_{ЗП} + З_{МВ}, грн \quad (4.19)$$

$$К_{ПО} = 33120 + 5040 \text{ (грн)} \quad (4.20)$$

Зарплата програміста визначається за формулою:

$$З_{ЗП} = t \cdot C_{пр}, грн \quad (4.21)$$

де t – загальна трудомісткість;

$C_{пр}$ – середня погодинна зарплата програміста, грн/годин

$$З_{ЗП} = 828 \cdot 40 = 33120 \text{ (грн)} \quad (4.22)$$

Вартість часу, необхідного для налагодження комп'ютерної програми :

$$З_{МВ} = t_{отл} \cdot C_{мч}, грн \quad (4.23)$$

де $t_{отл}$ – складність налаштування програми на комп’ютері, год.

$C_{мч}$ – погодинна вартість ЕОМ, грн/год.

$$З_{МВ} = 630 \cdot 8 = 5040 \text{ (грн)} \quad (4.24)$$

Визначені таким чином витрати на ПЗ є частиною одноразових капітальних витрат на створення системи.

Очікуваний період розробки програмного забезпечення:

$$Т = \frac{t}{B_k \cdot F_p}, міс. \quad (4.25)$$

де B_k – кількість виконавців;

F_p – щомісячний операційний фонд (З 40-годинним робочим тижнем F_p=175).

$$Т = \frac{828}{1 \cdot 175} = 4{,}7 \text{ (міс.)} \quad (4.26)$$

4.2. Маркетингове дослідження розробленого програмного продукту та ринків збуту

Розмовне моделювання є важливим завданням для розуміння природної мови та машинного інтелекту. Ця стаття досліджує підхід моделювання, який використовує нещодавно запропоновану послідовність для структурування послідовностей. (seq2seq). Ця конструкція може передбачити таке речення з урахуванням

попереднього речення або речень у розмові. Суть цієї структури полягає в тому, що її можна навчити end-to-end. Таким чином, ця структура може генерувати розмови з великим набором даних. Він здатний отримати знання як із набору даних, характерних для області розмови, так і з широкого, загального набору даних. Наприклад, у наборі даних IT-довідкової служби модель може знайти рішення технічної проблеми за допомогою діалогу. Дипломна робота стосується моделі діалогу та повторюваної нейронної мережі, яка може бути використана для змістовного діалогу зі штучним інтелектом та самонавчання.

Актуальність даної праці продиктована умовами постійного розвитку штучного інтелекту та машинного навчання у світі. Віртуальний співрозмовник - це комп'ютерна програма, призначена для імітації поведінки людської мови у спілкуванні з одним або кількома співрозмовниками. Назва програми співрозмовника також використовується щодо віртуальних співрозмовників.

Практичне застосування отриманих результатів полягає в проектуванні нейронної діалогової моделі в основі якої лежить машинного навчання бібліотеки Tensorflow, для оцінки переваг проектування системи діалогу.

Основними конкурентами цього ПЗ будуть віртуальні співрозмовники чату, які можуть навчитися самі. Неяскравішим прикладом з них є GoogleAssistant та Yandex Аліса. Вивчивши ринок на плагіати, можу сказати, що існують приклади використання різних типів рекурентних нейронних мереж для ведення. Вивчивши ринок на плагіати, можу сказати, що існують приклади використання різних типів рекурентних нейронних мереж для ведення осмисленого діалогу, однак немає розроблених або вдосконалених продуктів, які можуть бути покладені в основу рекурентних нейронних мереж.

4.3. Економічна ефективність впровадження даного програмного забезпечення

Таблиця 4.2 – Розрахування чистого грошового потоку після запуску програми

Індекси, грн	По місяцям							Усього за 6 міс.	Середнє значення за 6 міс.
		1	2	3	4	5	6		
1. Вкладення в ПЗ	20625	-	-	-	-	570	-	21925	3654
2. Витрати на запуск ПЗ:		17000	785	796	805	276	777	20439	3407
- на закупівлю необхіної техніки		19900	-	-	-	-	-	19900	3317
- на закупівлю розхідних матеріалів		1100	-	-	-	400	-	1500	250
- на просування		1700	1700	1700	1700	1700	1700	10200	1700
- на спожиту електроенергію		200	185	196	205	176	188	1150	192
3. Витрати після введення в есплуатацію даного ПЗ		1570	1570	1570	1570	1570	1570	9420	1570
- на оренду приміщення		1000	1000	1000	1000	1000	1000	6000	1000
- на спожиту електроенергію в приміщенні		180	190	180	170	180	175	1075	179
- на рекламування ПЗ		900	9000	900	900	900	900	5400	900

Продовження таблиця 4.2 – Розрахування чистого грошового потоку після запуску програми

- на використання інтернеу		150	150	150	150	150	150	900	150
4. Заощадження		15325	295	316	256	795	315	17276	2880
5. Амортизування ПЗ		5772,4	5772,4	5772,4	5772,4	5772,4	5772,4	34635	5772,4
6. Чистий грошовий потік		17537.5	5057,5	5095,5	5100,5	5748,5	5080,5	43620	2370
7. Показник знижки		0,750	0,755	0,645	0,484	0,475	0,543	-	-
8. Дисконтовані грошові находження		13153	3832	3287	2469	2731	2759	28231	4705

Економічна оцінка ефективності запропонованого впровадження оцінюється системою показників, що використовуються у міжнародній та національній практиці. Для оцінки економічної ефективності використовувались наступні показники:

1) Чиста теперішня вартість (NPU).
2) Рентабельність капіталовкладень.
3) Показник прибутковості ПЗ.
4) Показник ефективності вкладень в розробку ПЗ.

Вирішуючи питання щодо доцільності реалізації проекту, необхідно враховувати значення всіх показників. Розрахунок наведено в таблиці 4. 2

Показники економічної ефективності:

1) Чиста теперішня вартість:

$$NPU = 28231 - 21925 = 6306 \text{ грн} > 0 \quad (4.27)$$

2) Рентабельність капіталовкладень:

$$Т = 21925/6306 = 3{,}5 \text{ місяців} \quad (4.28)$$

3) Показник прибутковості (ПП):

$$ПП = 28231/21925 = 1{,}29 \quad (4.29)$$

Індекс економічної ефективності (NPU - чиста вартість реалізації доходу склала 6306 грн, можна сказати, що умови ефективності повністю виконуються NPU > 0. Середній термін окупності капітальних вкладень буде 3,5 місяців. Шестимісячний індекс прибутковості буде 1,29, тобто ПП> 1, проект повинен бути прийнятий. Отже, показник ефективності показує, що таке впровадження є вигідним.

Висновки до розділу 4

В дипломному проекті виконано «Розробка системи діалогової моделі з використанням рекурентної нейронної мережі за допомогою Tensor Flow».

При розробці програмного забезпечення рекурентної нейронної діалогової моделі витрати на розробку становили 21925 грн., термін розробки – 6 місяців і визначається складність розробленої інформаційної системи (1015, людино-годин). Коефіцієнт економічної склав 6306 грн, що означає відповідність умовам ефективності, де NPU > 0. Прогнозований час для окупності інвестицій становить 3,5 місяців.

Зручний інтерфейс розробленої програми пропонує відчуття простоти, зрозумілості та інформативності.

ВИСНОВКИ

Отже, у дипломній роботі ми проаналізували характеристики імітації мовної поведінки людини в процесі спілкування, визначивши основні функції та принципи функціювання чат-ботів з метою розробки моделі спілкування та інтегрували її у вигляді чат-бота. Сучасні технології дозволяють використовувати мережу як засіб спілкування а також відкривають можливість створювати інтерактивні форми спілкування, такі як: чати, форуми, конференції, електронну пошту. Існуючі співрозмовники замінюють програми зі штучним інтелектом, такі як чати, консультанти, помічники та інші. Але, на відміну від людського діалогу, програма не має гнучкого розумового інтелекту. На жаль, існуючі віртуальні співрозмовники не в повній мірі вирішують потрібність в імітації розмови людини. Словниковий запас потребує постійного оновлення, характер бесід, зазвичай не передає повного змісту розмови, тому більшість систем віртуальних співбесідників запрограмовані для простої розмови. Основна проблема штучного інтелекту це обробка природної мови людини.

Основою функціонування віртуальних співрозмовників є сучасна база знань. Зазвичай вона містить набори можливих питань і відповідних до них відповідей. В деякі програми можна інтегрувати процес самонавчання, а саме: змінювати словниковий запас відповідно до деяких особливостей мови. Проблема розробки сучасних систем на основі штучного інтелекту, які прогнозують інтелектуальну діяльність людини, на теперішний час в повній мірі не вирішена. Незважаючи на переваги, сучасні системи зі штучного інтелекту в даний час не можуть пройти тест Тюрінга , перевірки комп'ютерного інтелекту на інтелект подібно людині.

Даний дипломний проект пройшов апробацію на XI НАУКОВО-ТЕХНІЧНІЙ КОНФЕРЕНЦІЇ «СУЧАСНІ ІНФОКОМУНІКАЦІЙНІ ТЕХНОЛОГІЇ» у ДУТ, м. Київ, яка пройшла 12 грудня 2020 року.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate., arXiv:1409.0473v1, 2014.

2. Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. A neural probabilistic language model. The Journal of Machine Learning Research, pp. 1137–1155, 2003.

3. Hochreiter, S. and Schmidhuber, J. Long short-term memory. Neural Computation, 1997.

4. Jean, S., Cho, K., Memisevic, R., and Bengio, Y. On using very large target vocabulary for neural machine translation. CoRR, abs/1412.2007, 2014.

5. Jurafsky, D. and Martin, J. Speech and language processing. Pearson International, 2009.

6. Kalchbrenner, N. and Blunsom, P. Recurrent continuous translation models. In EMNLP, 2013.

7. Lester, J., Branting, K., and Mott, B. Conversational agents. In Handbook of Internet Computing. Chapman & Hall, 2004.

8. Luong, T., Sutskever, I., Le, Q. V., Vinyals, O., and Zaremba, W. Addressing the rare word problem in neural machine translation. arXiv preprint arXiv:1410.8206, 2014.

9. Mikolov, T. Statistical Language Models based on Neural Networks. PhD thesis, Brno University of Technology, 2012.

10. Mikolov, T., Karafi´at, M., Burget, L., Cernock`y, J., and Khudanpur, S. Recurrent neural network based language model. In INTERSPEECH, pp. 1045–1048, 2010.

11. Shang, L., Lu, Z., and Li, H. Neural responding machine for short-text conversation. In Proceedings of ACL, 2015.

12. Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. In NIPS, 2014.

13. Sordoni, A., Galley, M., Auli, M., Brockett, C., Ji, Y., Mitchell, M., Gao, J., Dolan, B., and Nie, J.-Y. A neural network approach to context-sensitive generation of conversational responses. In Proceedings of NAACL, 2015.

14. Tiedemann, J. News from OPUS - A collection of multilingual parallel corpora with tools and interfaces. In Nicolov, N., Bontcheva, K., Angelova, G., and Mitkov, R. (eds.), Recent Advances in Natural Language Processing, volume V, pp. 237–248. John Benjamins, Amsterdam/Philadelphia, Borovets, Bulgaria, ISBN 978 90 272 4825 1, 2009.

15. Turing, A. M. Computing machinery and intelligence. Mind, pp. 433–460, 1950.

16. Vinyals, O., Kaiser, L., Koo, T., Petrov, S., Sutskever, I., and Hinton, G. Grammar as a foreign language. arXiv preprint arXiv:1412.7449, 2014.

17. Тема 5. Искусственные нейронные сети. [Електронний ресурс] - Режим доступу: http://www.victoria.lviv.ua/html/oio/html/theme5_rus.htm – (дата звернення 27.10.2020 р).

18. Азбука ИИ: «Машинное обучение». [Електронний ресурс] - Режим доступу: https://nplus1.ru/material/2016/09/06/mistakesflow – (дата звернення 10.11.2020).

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

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

ДОДАТКИ

Приклад Python - коду.

Текст програми *Chatbot.py*

```
import argparse # Command line parsing
import configparser # Saving the models parameters
import datetime # Chronometer
import os # Files management
import tensorflow as tf
import numpy as np
import math
from tqdm import tqdm # Progress bar
from tensorflow.python import debug as tf_debug
from chatbot.textdata import TextData
from chatbot.model import Model
class Chatbot:
"""
Main class which launch the training or testing mode
"""
def __init__(self):
"""
"""
# Model/dataset parameters
self.args = None
# Task specific object
self.textData = None # Dataset
self.model = None # Sequence to sequence model
# Tensorflow utilities for convenience saving/logging
self.writer = None
self.saver = None
self.modelDir = '' # Where the model is saved
self.globStep = 0 # Represent the number of iteration for the current model
# TensorFlow main session (we keep track for the daemon)
self.sess = None
# Filename and directories constants
self.MODEL_DIR_BASE = 'save' + os.sep + 'model'
self.MODEL_NAME_BASE = 'model'
self.MODEL_EXT = '.ckpt'
self.CONFIG_FILENAME = 'params.ini'
self.CONFIG_VERSION = '0.5'
self.TEST_IN_NAME = 'data' + os.sep + 'test' + os.sep + 'samples.txt'
self.TEST_OUT_SUFFIX = '_predictions.txt'
self.SENTENCES_PREFIX = ['Q: ', 'A: ']

def main(self, args=None):
"""
Launch the training and/or the interactive mode
"""
print('Welcome to DeepQA v0.1 !')
print()
print('TensorFlow detected: v{}'.format(tf.__version__))
# General initialisation
self.args = self.parseArgs(args)
if not self.args.rootDir:
self.args.rootDir = os.getcwd() # Use the current working directory
#tf.logging.set_verbosity(tf.logging.INFO) # DEBUG, INFO, WARN (default), ERROR, or FATAL
self.loadModelParams() # Update the self.modelDir and self.globStep, for now, not used when loading Model (but need to be called before _getSummaryName)
self.textData = TextData(self.args)
# TODO: Add a mode where we can force the input of the decoder // Try to visualize the predictions for
# each word of the vocabulary / decoder input
# TODO: For now, the model are trained for a specific dataset (because of the maxLength which define the
# vocabulary). Add a compatibility mode which allow to launch a model trained on a different vocabulary (
# remap the word2id/id2word variables).
if self.args.createDataset:
print('Dataset created! Thanks for using this program')
return # No need to go further
# Prepare the model
with tf.device(self.getDevice()):
self.model = Model(self.args, self.textData)
# Saver/summaries
```

```
self.writer                                =
tf.summary.FileWriter(self._getSummaryN
ame())
self.saver                                 =
tf.train.Saver(max_to_keep=200)
# TODO: Fixed seed (WARNING: If dataset
shuffling, make sure to do that after
saving the
# dataset, otherwise, all which cames
after the shuffling won't be replicable
when
# reloading the dataset). How to restore
the seed after loading ??
# Also fix seed for random.shuffle (does
it works globally for all files ?)
# Running session
self.sess                                  =
tf.Session(config=tf.ConfigProto(
allow_soft_placement=True,    #    Allows
backup device for non GPU-available
operations (when forcing GPU)
log_device_placement=False)    #    Too
verbose ?
) # TODO: Replace all sess by self.sess
(not necessary a good idea) ?
if self.args.debug:
self.sess                                  =
tf_debug.LocalCLIDebugWrapperSession(se
lf.sess)
self.sess.add_tensor_filter("has_inf_or
_nan", tf_debug.has_inf_or_nan)
print('Initialize variables...')
self.sess.run(tf.global_variables_initi
alizer())
# Reload the model eventually (if it
exist.), on testing mode, the models are
not loaded here (but in predictTestset)
if           self.args.test           !=
Chatbot.TestMode.ALL:
self.managePreviousModel(self.sess)
# Initialize embeddings with pre-trained
word2vec vectors
if self.args.initEmbeddings:
self.loadEmbedding(self.sess)
if self.args.test:
if           self.args.test           ==
Chatbot.TestMode.INTERACTIVE:
self.mainTestInteractive(self.sess)
elif          self.args.test           ==
Chatbot.TestMode.ALL:
print('Start predicting...')
self.predictTestset(self.sess)
print('All predictions done')
elif          self.args.test           ==
Chatbot.TestMode.DAEMON:
print('Daemon    mode,    running    in
background...')
else:
raise RuntimeError('Unknown test mode:
{}'.format(self.args.test))   #   Should
never happen
else:
self.mainTrain(self.sess)
if           self.args.test           !=
Chatbot.TestMode.DAEMON:
self.sess.close()
print("The End! Thanks for using this
program")
def mainTrain(self, sess):
""" Training loop
Args:
sess: The current running session
"""
# Specific training dependent loading
self.textData.makeLighter(self.args.rat
ioDataset) # Limit the number of training
samples
mergedSummaries = tf.summary.merge_all()
# Define the summary operator (Warning:
Won't appear on the tensorboard graph)
if self.globStep == 0: # Not restoring
from previous run
self.writer.add_graph(sess.graph)      #
First time only
#  If  restoring  a  model,  restore  the
progression bar ? and current batch ?
print('Start training (press Ctrl+C to
save and exit)...')
try: # If the user exit while training,
we still try to save the model
for e in range(self.args.numEpochs):
print()
print("----- Epoch {}/{} ; (lr={}) -----
".format(e+1,       self.args.numEpochs,
self.args.learningRate))
batches = self.textData.getBatches()
# TODO: Also update learning parameters
eventually
tic = datetime.datetime.now()
for    nextBatch    in    tqdm(batches,
desc="Training"):
# Training pass
ops,              feedDict              =
self.model.step(nextBatch)
assert len(ops) == 2 # training, loss
_,   loss,   summary   =   sess.run(ops   +
(mergedSummaries,), feedDict)
self.writer.add_summary(summary,
self.globStep)
self.globStep += 1
# Output training status
if self.globStep % 100 == 0:
perplexity  =  math.exp(float(loss))  if
loss < 300 else float("inf")
tqdm.write("----- Step %d -- Loss %.2f -
-  Perplexity  %.2f"  %  (self.globStep,
loss, perplexity))
```

```
# Checkpoint
if self.globStep % self.args.saveEvery == 0:
self._saveSession(sess)
toc = datetime.datetime.now()
print("Epoch finished in {}".format(toc-tic)) # Warning: Will overflow if an epoch takes more than 24 hours, and the output isn't really nicer
except (KeyboardInterrupt, SystemExit): # If the user press Ctrl+C while testing progress
print('Interruption detected, exiting the program...')
self._saveSession(sess) # Ultimate saving before complete exit
```

Model.py

```
import tensorflow as tf
from chatbot.textdata import Batch
class ProjectionOp:
""" Single layer perceptron
Project input tensor on the output dimension
"""
def __init__(self, shape, scope=None, dtype=None):
"""
Args:
shape: a tuple (input dim, output dim)
scope (str): encapsulate variables
dtype: the weights type
"""
assert len(shape) == 2
self.scope = scope
# Projection on the keyboard
with tf.variable_scope('weights_' + self.scope):
self.W_t = tf.get_variable(
'weights',
shape,
#
initializer=tf.truncated_normal_initializer() # TODO: Tune value (fct of input size: 1/sqrt(input_dim))
dtype=dtype
)
self.b = tf.get_variable(
'bias',
shape[0],
initializer=tf.constant_initializer(),
dtype=dtype
)
self.W = tf.transpose(self.W_t)
def getWeights(self):
""" Convenience method for some tf arguments
"""
return self.W, self.b
def __call__(self, X):
""" Project the output of the decoder into the vocabulary space
Args:
X (tf.Tensor): input value
"""
with tf.name_scope(self.scope):
return tf.matmul(X, self.W) + self.b
class Model:
"""
Implementation of a seq2seq model.
Architecture:
Encoder/decoder
2 LTSM layers
"""
def __init__(self, args, textData):
"""
Args:
args: parameters of the model
textData: the dataset object
"""
print("Model creation...")
self.textData = textData # Keep a reference on the dataset
self.args = args # Keep track of the parameters of the model
self.dtype = tf.float32
# Placeholders
self.encoderInputs = None
self.decoderInputs = None # Same that decoderTarget plus the <go>
self.decoderTargets = None
self.decoderWeights = None # Adjust the learning to the target sentence size
# Main operators
self.lossFct = None
self.optOp = None
self.outputs = None # Outputs of the network, list of probability for each words
# Construct the graphs
self.buildNetwork()
def buildNetwork(self):
""" Create the computational graph
"""
# TODO: Create name_scopes (for better graph visualisation)
# TODO: Use buckets (better perfs)
# Parameters of sampled softmax (needed for attention mechanism and a large vocabulary size)
outputProjection = None
if 0 < self.args.softmaxSamples < self.textData.getVocabularySize():
outputProjection = ProjectionOp(
(self.textData.getVocabularySize(),
self.args.hiddenSize),
scope='softmax_projection',
```

```
dtype=self.dtype
)
def sampledSoftmax(labels, inputs):
labels = tf.reshape(labels, [-1, 1]) # Add one dimension (nb of true classes, here 1)
# We need to compute the sampled_softmax_loss using 32bit floats to
# avoid numerical instabilities.
localWt = tf.cast(outputProjection.W_t, tf.float32)
localB = tf.cast(outputProjection.b, tf.float32)
localInputs = tf.cast(inputs, tf.float32)
return tf.cast(
tf.nn.sampled_softmax_loss(
localWt, # Should have shape [num_classes, dim]
localB,
labels,
localInputs,
self.args.softmaxSamples, # The number of classes to randomly sample per batch
self.textData.getVocabularySize()), # The number of classes
self.dtype)
# Creation of the PHM cell
def create_PHM_cell():
encoDecoCell = tf.contrib.PHM.BasicLSTMCell( # Or GRUCell, LSTMCell(args.hiddenSize)
self.args.hiddenSize,
)
if not self.args.test: # TODO: Should use a placeholder instead
encoDecoCell = tf.contrib.PHM.DropoutWrapper(
encoDecoCell,
input_keep_prob=1.0,
output_keep_prob=self.args.dropout
)
return encoDecoCell
encoDecoCell = tf.contrib.PHM.MultiPHMCell(
[create_PHM_cell() for _ in range(self.args.numLayers)],
)
# Network input (placeholders)
with tf.name_scope('placeholder_encoder'):
self.encoderInputs = [tf.placeholder(tf.int32, [None, ]) for _ in range(self.args.maxLengthEnco)] # Batch size * sequence length * input dim
with tf.name_scope('placeholder_decoder'):
self.decoderInputs = [tf.placeholder(tf.int32, [None, ], name='inputs') for _ in range(self.args.maxLengthDeco)] # Same sentence length for input and output (Right ?)
# Sampled softmax only makes sense if we sample less than vocabulary size.
self.decoderTargets = [tf.placeholder(tf.int32, [None, ], name='targets') for _ in range(self.args.maxLengthDeco)]
self.decoderWeights = [tf.placeholder(tf.float32, [None, ], name='weights') for _ in range(self.args.maxLengthDeco)]
# Define the network
# Here we use an embedding model, it takes integer as input and convert them into word vector for
# better word representation
decoderOutputs, states = tf.contrib.legacy_seq2seq.embedding_PHM_seq2seq(
self.encoderInputs, # List<[batch=?, inputDim=1]>, list of size args.maxLength
self.decoderInputs, # For training, we force the correct output (feed_previous=False)
encoDecoCell,
self.textData.getVocabularySize(),
self.textData.getVocabularySize(), # Both encoder and decoder have the same number of class
embedding_size=self.args.embeddingSize, # Dimension of each word
output_projection=outputProjection.getWeights() if outputProjection else None,
feed_previous=bool(self.args.test) # When we test (self.args.test), we use previous output as next input (feed_previous)
)
# TODO: When the LSTM hidden size is too big, we should project the LSTM output into a smaller space (4086 => 2046): Should speed up
# training and reduce memory usage. Other solution, use sampling softmax
# For testing only
if self.args.test:
if not outputProjection:
self.outputs = decoderOutputs
else:
self.outputs = [outputProjection(output) for output in decoderOutputs]
# TODO: Attach a summary to visualize the output
```

```
# For training only
else:
# Finally, we define the loss function
self.lossFct                            =
tf.contrib.legacy_seq2seq.sequence_loss
(
decoderOutputs,
self.decoderTargets,
self.decoderWeights,
self.textData.getVocabularySize(),
softmax_loss_function= sampledSoftmax if
outputProjection else None # If None, use
default SoftMax
)
tf.summary.scalar('loss',  self.lossFct)
# Keep track of the cost
# Initialize the optimizer
opt = tf.train.AdamOptimizer(
learning_rate=self.args.learningRate,
beta1=0.9,
beta2=0.999,
epsilon=1e-08
)
self.optOp = opt.minimize(self.lossFct)
def step(self, batch):
""" Forward/training step operation.
Does not perform run on itself but just
return the operators to do so. Those have
then to be run
Args:
batch  (Batch): Input  data  on  testing
mode, input and target on output mode
Return:
(ops), dict: A tuple of the (training,
loss) operators or (outputs,) in testing
mode with the associated feed dictionary
"""
# Feed the dictionary
feedDict = {}
ops = None
if not self.args.test: # Training
for i in range(self.args.maxLengthEnco):
feedDict[self.encoderInputs[i]]        =
batch.encoderSeqs[i]
for i in range(self.args.maxLengthDeco):
feedDict[self.decoderInputs[i]]        =
batch.decoderSeqs[i]
feedDict[self.decoderTargets[i]]       =
batch.targetSeqs[i]
feedDict[self.decoderWeights[i]]       =
batch.weights[i]
ops = (self.optOp, self.lossFct)
else: # Testing (batchSize == 1)
for i in range(self.args.maxLengthEnco):
feedDict[self.encoderInputs[i]]        =
batch.encoderSeqs[i]
feedDict[self.decoderInputs[0]]        =
[self.textData.goToken]
ops = (self.outputs,)
# Return one pass operator
return ops, feedDict
```

Textdata.py

```
import numpy as np
import nltk # For tokenize
from tqdm import tqdm # Progress bar
import pickle # Saving the data
import math # For float comparison
import os # Checking file existance
import random
import string

import collections
from  chatbot.corpus.cornelldata  import
CornellData
from chatbot.corpus.opensubsdata  import
OpensubsData
from  chatbot.corpus.scotusdata  import
ScotusData
from  chatbot.corpus.ubuntudata  import
UbuntuData
from      chatbot.corpus.lightweightdata
import LightweightData
class Batch:
"""Struct containing batches info
"""
def __init__(self):
self.encoderSeqs = []
self.decoderSeqs = []
self.targetSeqs = []
self.weights = []
class TextData:
"""Dataset class
Warning: No vocabulary limit
"""
availableCorpus                         =
collections.OrderedDict([ # OrderedDict
because the first element is the default
choice
('cornell', CornellData),
('opensubs', OpensubsData),
('scotus', ScotusData),
('ubuntu', UbuntuData),
('lightweight', LightweightData),
])
@staticmethod
def corpusChoices():
"""Return the dataset availables
Return:
list<string>: the supported corpus
"""
return
list(TextData.availableCorpus.keys())
def __init__(self, args):
"""Load all conversations
Args:
args: parameters of the model
```

```
"""
# Model parameters
self.args = args
# Path variables
self.corpusDir                              =
os.path.join(self.args.rootDir,  'data',
self.args.corpus)
basePath = self._constructBasePath()
self.fullSamplesPath = basePath + '.pkl'
# Full sentences length/vocab
self.filteredSamplesPath = basePath + '-
length{}-filter{}-
vocabSize{}.pkl'.format(
self.args.maxLength,
self.args.filterVocab,
self.args.vocabularySize,
) # Sentences/vocab filtered for this
model
self.padToken = -1 # Padding
self.goToken = -1 # Start of sequence
self.eosToken = -1 # End of sequence
self.unknownToken = -1 # Word dropped
from vocabulary
self.trainingSamples = [] # 2d array
containing each question and his answer
[[input,target]]
self.word2id = {}
self.id2word = {} # For a rapid
conversion (Warning: If replace dict by
list, modify the filtering to avoid
linear complexity with del)
self.idCount = {} # Useful to filters the
words (TODO: Could replace dict by list
or use collections.Counter)
self.loadCorpus()
# Plot some stats:
self._printStats()
if self.args.playDataset:
self.playDataset()
def _printStats(self):
print('Loaded   {}:   {}   words,   {}
QA'.format(self.args.corpus,
len(self.word2id),
len(self.trainingSamples)))
def _constructBasePath(self):
"""Return the name of the base prefix of
the current dataset
"""
path  =  os.path.join(self.args.rootDir,
'data' + os.sep + 'samples' + os.sep)
path            +=            'dataset-
{}'.format(self.args.corpus)
if self.args.datasetTag:
path += '-' + self.args.datasetTag
return path
def makeLighter(self, ratioDataset):
"""Only keep a small fraction of the
dataset, given by the ratio
"""
#if not math.isclose(ratioDataset, 1.0):
# self.shuffle() # Really ?
#  print('WARNING:  Ratio  feature  not
implemented !!!')
pass
def shuffle(self):
"""Shuffle the training samples
"""
print('Shuffling the dataset...')
random.shuffle(self.trainingSamples)
def _createBatch(self, samples):
"""Create a single batch from the list
of  sample.  The  batch  size  is
automatically defined by the number of
samples given.
The inputs should already be inverted.
The target should already have <go> and
<eos>
Warning: This function should not make
direct calls to args.batchSize !!!
Args:
samples (list<Obj>): a list of samples,
each sample being on the form [input,
target]
Return:
Batch: a batch object en
"""
batch = Batch()
batchSize = len(samples)
# Create the batch tensor
for i in range(batchSize):
# Unpack the sample
sample = samples[i]
if      not      self.args.test      and
self.args.watsonMode:  #  Watson  mode:
invert question and answer
sample = list(reversed(sample))
if      not      self.args.test      and
self.args.autoEncode: # Autoencode: use
either the question or answer for both
input and output
k = random.randint(0, 1)
sample = (sample[k], sample[k])
# TODO: Why re-processed that at each
epoch ? Could precompute that
# once and reuse those every time. Is not
the bottleneck so won't change
# much ? and if preprocessing, should be
compatible with autoEncode & cie.
batch.encoderSeqs.append(list(reversed(
sample[0]))) # Reverse inputs (and not
outputs), little trick as defined on the
original seq2seq paper
batch.decoderSeqs.append([self.goToken]
+ sample[1] + [self.eosToken]) # Add the
<go> and <eos> tokens
batch.targetSeqs.append(batch.decoderSe
qs[-1][1:])  # Same as decoder, but
shifted to the left (ignore the <go>)
```

```
# Long sentences should have been filtered during the dataset creation
assert len(batch.encoderSeqs[i]) <= self.args.maxLengthEnco
assert len(batch.decoderSeqs[i]) <= self.args.maxLengthDeco
# TODO: Should use tf batch function to automatically add padding and batch samples
# Add padding & define weight
batch.encoderSeqs[i] = [self.padToken] * (self.args.maxLengthEnco - len(batch.encoderSeqs[i])) + batch.encoderSeqs[i] # Left padding for the input
batch.weights.append([1.0] * len(batch.targetSeqs[i]) + [0.0] * (self.args.maxLengthDeco - len(batch.targetSeqs[i])))
batch.decoderSeqs[i] = batch.decoderSeqs[i] + [self.padToken] * (self.args.maxLengthDeco - len(batch.decoderSeqs[i]))
batch.targetSeqs[i] = batch.targetSeqs[i] + [self.padToken] * (self.args.maxLengthDeco - len(batch.targetSeqs[i]))
# Simple hack to reshape the batch
encoderSeqsT = [] # Corrected orientation
for i in range(self.args.maxLengthEnco):
encoderSeqT = []
for j in range(batchSize):
encoderSeqT.append(batch.encoderSeqs[j][i])
encoderSeqsT.append(encoderSeqT)
batch.encoderSeqs = encoderSeqsT
decoderSeqsT = []
targetSeqsT = []
weightsT = []
for i in range(self.args.maxLengthDeco):
decoderSeqT = []
targetSeqT = []
weightT = []
for j in range(batchSize):
decoderSeqT.append(batch.decoderSeqs[j][i])
targetSeqT.append(batch.targetSeqs[j][i])
weightT.append(batch.weights[j][i])
decoderSeqsT.append(decoderSeqT)
targetSeqsT.append(targetSeqT)
weightsT.append(weightT)
batch.decoderSeqs = decoderSeqsT
batch.targetSeqs = targetSeqsT
batch.weights = weightsT
# # Debug
# self.printBatch(batch) # Input inverted, padding should be correct
#
print(self.sequence2str(samples[0][0]))
#
print(self.sequence2str(samples[0][1]))
# Check we did not modified the original sample
return batch
def getBatches(self):
"""Prepare the batches for the current epoch
Return:
list<Batch>: Get a list of the batches for the next epoch
"""
self.shuffle()
batches = []
def genNextSamples():
""" Generator over the mini-batch training samples
"""
for i in range(0, self.getSampleSize(), self.args.batchSize):
yield self.trainingSamples[i:min(i + self.args.batchSize, self.getSampleSize())]
# TODO: Should replace that by generator (better: by tf.queue)
for samples in genNextSamples():
batch = self._createBatch(samples)
batches.append(batch)
return batches
def getSampleSize(self):
"""Return the size of the dataset
Return:
int: Number of training samples
"""
return len(self.trainingSamples)
def getVocabularySize(self):
"""Return the number of words present in the dataset
Return:
int: Number of word on the loader corpus
"""
return len(self.word2id)
def loadCorpus(self):
"""Load/create the conversations data
"""
datasetExist = os.path.isfile(self.filteredSamplesPath)
if not datasetExist: # First time we load the database: creating all files
print('Training samples not found. Creating dataset...')
datasetExist = os.path.isfile(self.fullSamplesPath) # Try to construct the dataset from the preprocessed entry
if not datasetExist:
```

```
print('Constructing full dataset...')
optional = ''
if self.args.corpus == 'lightweight':
if not self.args.datasetTag:
raise ValueError('Use the --datasetTag
to define the lightweight file to use.')
optional = os.sep + self.args.datasetTag
# HACK: Forward the filename
# Corpus creation
corpusData                                =
TextData.availableCorpus[self.args.corp
us](self.corpusDir + optional)
self.createFullCorpus(corpusData.getCon
versations())
self.saveDataset(self.fullSamplesPath)
else:
self.loadDataset(self.fullSamplesPath)
self._printStats()
print('Filtering words (vocabSize = {}
and wordCount > {})...'.format(
self.args.vocabularySize,
self.args.filterVocab
))
self.filterFromFull() # Extract the sub
vocabulary for the given maxLength and
filterVocab
# Saving
print('Saving dataset...')
self.saveDataset(self.filteredSamplesPa
th) # Saving tf samples
else:
self.loadDataset(self.filteredSamplesPa
th)
assert self.padToken == 0
def saveDataset(self, filename):
"""Save samples to file
Args:
filename (str): pickle filename
"""
with open(os.path.join(filename), 'wb')
as handle:
data = { # Warning: If adding something
here, also modifying loadDataset
'word2id': self.word2id,
'id2word': self.id2word,
'idCount': self.idCount,
'trainingSamples': self.trainingSamples
}
pickle.dump(data, handle, -1) # Using
the highest protocol available
def loadDataset(self, filename):
"""Load samples from file
Args:
filename (str): pickle filename
"""
dataset_path = os.path.join(filename)
print('Loading       dataset       from
{}'.format(dataset_path))
with open(dataset_path, 'rb') as handle:
data = pickle.load(handle) # Warning: If
adding something here, also modifying
saveDataset
self.word2id = data['word2id']
self.id2word = data['id2word']
self.idCount = data.get('idCount', None)
self.trainingSamples                      =
data['trainingSamples']
self.padToken = self.word2id['<pad>']
self.goToken = self.word2id['<go>']
self.eosToken = self.word2id['<eos>']
self.unknownToken                         =
self.word2id['<unknown>']   #   Restore
special words
def filterFromFull(self):
""" Load the pre-processed full corpus
and filter the vocabulary / sentences
to match the given model options
"""
def           mergeSentences(sentences,
fromEnd=False):
"""Merge the sentences until the max
sentence length is reached
Also decrement id count for unused
sentences.
Args:
sentences (list<list<int>>): the list of
sentences for the current line
fromEnd (bool): Define the question on
the answer
Return:
list<int>: the list of the word ids of
the sentence
"""
# We add sentence by sentence until we
reach the maximum length
merged = []
# If question: we only keep the last
sentences
# If answer: we only keep the first
sentences
if fromEnd:
sentences = reversed(sentences)
for sentence in sentences:
# If the total length is not too big, we
still can add one more sentence
if  len(merged)  +  len(sentence)  <=
self.args.maxLength:
if fromEnd: # Append the sentence
merged = sentence + merged
else:
merged = merged + sentence
else: # If the sentence is not used,
neither are the words
for w in sentence:
self.idCount[w] -= 1
return merged
newSamples = []
```

```
# 1st step: Iterate over all words and add filters the sentences
# according to the sentence lengths
for inputWords, targetWords in tqdm(self.trainingSamples, desc='Filter sentences:', leave=False):
inputWords = mergeSentences(inputWords, fromEnd=True)
targetWords = mergeSentences(targetWords, fromEnd=False)
newSamples.append([inputWords, targetWords])
words = []
# WARNING: DO NOT FILTER THE UNKNOWN TOKEN !!! Only word which has count==0 ?
# 2nd step: filter the unused words and replace them by the unknown token
# This is also where we update the correnspondance dictionaries
specialTokens = { # TODO: bad HACK to filter the special tokens. Error prone if one day add new special tokens
self.padToken,
self.goToken,
self.eosToken,
self.unknownToken
}
newMapping = {} # Map the full words ids to the new one (TODO: Should be a list)
newId = 0
selectedWordIds = collections \
.Counter(self.idCount) \
.most_common(self.args.vocabularySize or None) # Keep all if vocabularySize == 0
selectedWordIds = {k for k, v in selectedWordIds if v > self.args.filterVocab}
selectedWordIds |= specialTokens
for wordId, count in [(i, self.idCount[i]) for i in range(len(self.idCount))]: # Iterate in order
if wordId in selectedWordIds: # Update the word id
newMapping[wordId] = newId
word = self.id2word[wordId] # The new id has changed, update the dictionaries
del self.id2word[wordId] # Will be recreated if newId == wordId
self.word2id[word] = newId
self.id2word[newId] = word
newId += 1
else: # Cadidate to filtering, map it to unknownToken (Warning: don't filter special token)
newMapping[wordId] = self.unknownToken
del self.word2id[self.id2word[wordId]] # The word isn't used anymore
del self.id2word[wordId]
# Last step: replace old ids by new ones and filters empty sentences
def replace_words(words):
valid = False # Filter empty sequences
for i, w in enumerate(words):
words[i] = newMapping[w]
if words[i] != self.unknownToken: # Also filter if only contains unknown tokens
valid = True
return valid
self.trainingSamples.clear()
for inputWords, targetWords in tqdm(newSamples, desc='Replace ids:', leave=False):
valid = True
valid &= replace_words(inputWords)
valid &= replace_words(targetWords)
valid &= targetWords.count(self.unknownToken) == 0 # Filter target with out-of-vocabulary target words ?
if valid:
self.trainingSamples.append([inputWords, targetWords]) # TODO: Could replace list by tuple
self.idCount.clear() # Not usefull anymore. Free data
def createFullCorpus(self, conversations):
"""Extract all data from the given vocabulary.
Save the data on disk. Note that the entire corpus is pre-processed
without restriction on the sentence length or vocab size.
"""
# Add standard tokens
self.padToken = self.getWordId('<pad>') # Padding (Warning: first things to add > id=0 !!)
self.goToken = self.getWordId('<go>') # Start of sequence
self.eosToken = self.getWordId('<eos>') # End of sequence
self.unknownToken = self.getWordId('<unknown>') # Word dropped from vocabulary
# Preprocessing data
for conversation in tqdm(conversations, desc='Extract conversations'):
self.extractConversation(conversation)
# The dataset will be saved in the same order it has been extracted
def extractConversation(self, conversation):
```

```
"""Extract the sample lines from the conversations
Args:
conversation (Obj): a conversation object containing the lines to extract
"""
if self.args.skipLines: # WARNING: The dataset won't be regenerated if the choice evolve (have to use the datasetTag)
step = 2
else:
step = 1
# Iterate over all the lines of the conversation
for i in tqdm_wrap(
range(0, len(conversation['lines']) - 1, step), # We ignore the last line (no answer for it)
desc='Conversation',
leave=False
):
inputLine = conversation['lines'][i]
targetLine = conversation['lines'][i+1]
inputWords = self.extractText(inputLine['text'])
targetWords = self.extractText(targetLine['text'])
if inputWords and targetWords: # Filter wrong samples (if one of the list is empty)
self.trainingSamples.append([inputWords, targetWords])
def extractText(self, line):
"""Extract the words from a sample lines
Args:
line (str): a line containing the text to extract
Return:
list<list<int>>: the list of sentences of word ids of the sentence
"""
sentences = [] # List[List[str]]
# Extract sentences
sentencesToken = nltk.sent_tokenize(line)
# We add sentence by sentence until we reach the maximum length
for i in range(len(sentencesToken)):
tokens = nltk.word_tokenize(sentencesToken[i])
tempWords = []
for token in tokens:
tempWords.append(self.getWordId(token))
# Create the vocabulary and the training sentences
sentences.append(tempWords)
return sentences
def getWordId(self, word, create=True):
"""Get the id of the word (and add it to the dictionary if not existing). If the word does not exist and
create is set to False, the function will return the unknownToken value
Args:
word (str): word to add
create (Bool): if True and the word does not exist already, the world will be added
Return:
int: the id of the word created
"""
# Should we Keep only words with more than one occurrence ?
word = word.lower() # Ignore case
# At inference, we simply look up for the word
if not create:
wordId = self.word2id.get(word, self.unknownToken)
# Get the id if the word already exist
elif word in self.word2id:
wordId = self.word2id[word]
self.idCount[wordId] += 1
# If not, we create a new entry
else:
wordId = len(self.word2id)
self.word2id[word] = wordId
self.id2word[wordId] = word
self.idCount[wordId] = 1
return wordId
def printBatch(self, batch):
"""Print a complete batch, useful for debugging
Args:
batch (Batch): a batch object
"""
print('----- Print batch -----')
for i in range(len(batch.encoderSeqs[0])):  # Batch size
print('Encoder: {}'.format(self.batchSeq2str(batch.encoderSeqs, seqId=i)))
print('Decoder: {}'.format(self.batchSeq2str(batch.decoderSeqs, seqId=i)))
print('Targets: {}'.format(self.batchSeq2str(batch.targetSeqs, seqId=i)))
print('Weights: {}'.format(' '.join([str(weight) for weight in [batchWeight[i] for batchWeight in batch.weights]])))
def sequence2str(self, sequence, clean=False, reverse=False):
"""Convert a list of integer into a human readable string
```

```
Args:
sequence (list<int>): the sentence to print
clean (Bool): if set, remove the <go>, <pad> and <eos> tokens
reverse (Bool): for the input, option to restore the standard order
Return:
str: the sentence
"""
if not sequence:
return ''
if not clean:
return ' '.join([self.id2word[idx] for idx in sequence])
sentence = []
for wordId in sequence:
if wordId == self.eosToken: # End of generated sentence
break
elif wordId != self.padToken and wordId != self.goToken:
sentence.append(self.id2word[wordId])
if reverse: # Reverse means input so no <eos> (otherwise pb with previous early stop)
sentence.reverse()
return self.detokenize(sentence)
def detokenize(self, tokens):
"""Slightly cleaner version of joining with spaces.
Args:
tokens (list<string>): the sentence to print
Return:
str: the sentence
"""
return ''.join([
' ' + t if not t.startswith('\'') and
t not in string.punctuation
else t
for t in tokens]).strip().capitalize()
def batchSeq2str(self, batchSeq, seqId=0, **kwargs):
"""Convert a list of integer into a human readable string.
The difference between the previous function is that on a batch object, the values have been reorganized as
batch instead of sentence.
Args:
batchSeq (list<list<int>>): the sentence(s) to print
seqId (int): the position of the sequence inside the batch
kwargs: the formatting options( See sequence2str() )
Return:
str: the sentence
"""
sequence = []
for i in range(len(batchSeq)): # Sequence length
sequence.append(batchSeq[i][seqId])
return self.sequence2str(sequence, **kwargs)
def sentence2enco(self, sentence):
"""Encode a sequence and return a batch as an input for the model
Return:
Batch: a batch object containing the sentence, or none if something went wrong
"""
if sentence == '':
return None
# First step: Divide the sentence in token
tokens = nltk.word_tokenize(sentence)
if len(tokens) > self.args.maxLength:
return None
# Second step: Convert the token in word ids
wordIds = []
for token in tokens:
wordIds.append(self.getWordId(token, create=False)) # Create the vocabulary and the training sentences
# Third step: creating the batch (add padding, reverse)
batch = self._createBatch([[wordIds, []]]) # Mono batch, no target output
return batch
def deco2sentence(self, decoderOutputs):
"""Decode the output of the decoder and return a human friendly sentence
decoderOutputs (list<np.array>):
"""
sequence = []
# Choose the words with the highest prediction score
for out in decoderOutputs:
sequence.append(np.argmax(out)) # Adding each predicted word ids
return sequence # We return the raw sentence. Let the caller do some cleaning eventually
def playDataset(self):
"""Print a random dialogue from the dataset
"""
print('Randomly play samples:')
for i in range(self.args.playDataset):
idSample = random.randint(0, len(self.trainingSamples) - 1)
print('Q: {}'.format(self.sequence2str(self.trainingSamples[idSample][0], clean=True)))
```

```
print('A:
{}'.format(self.sequence2str(self.train
ingSamples[idSample][1], clean=True)))
print()
pass
def      tqdm_wrap(iterable,      *args,
**kwargs):
if len(iterable) > 100:
return tqdm(iterable, *args, **kwargs)
return iterable
```