Міністерство освіти і науки України
Державний ВНЗ «Національний гірничий університет»

Факультет інформаційних технологій
(факультет)

Кафедра програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
дипломної роботи

магістра
(назва освітньо-кваліфікаційного рівня)

галузь знань *12 Інформаційні технології*
(шифр і назва галузі знань)

напрям підготовки *122 Комп'ютерні науки*
(код і назва напряму підготовки)

спеціальність *Інформаційні управляючі системи та технології*
(код і назва спеціальності)

освітній рівень *магістр*
(назва освітнього рівня)

кваліфікація *інженер з комп'ютерних систем*
(назва кваліфікації)

на тему: *Обгрунтування синтезу діалогової моделі на основі рекурентної нейронної мережі*

Виконавець:
студент *2* **курсу , групи** *122М-16-1*

(підпис) *Ткач М.Ю.* (прізвище та ініціали)

Керівники	Посада, прізвище, ініціали	Оцінка	Підпис
проекту	*д.т.н., проф. Мещеряков Л.І.*		
розділів:			
Спеціальний	*д.т.н., проф. Мещеряков Л.І.*		
Економічний	*к.е.н., доц. Касьяненко Л.В.*		

Рецензент			

Нормоконтроль	*к.т.н., доц. Коротенко Л.М.*		

Дніпро
2018

Міністерство освіти і науки України
Державний вищий навчальний заклад
«Національний гірничий університет»

ЗАТВЕРДЖЕНО:
завідувач кафедри

програмного забезпечення комп'ютерних систем
(повна назва)

_____ І.М. Удовик
(підпис) (прізвище, ініціали)

« » _____ 20 ____ року

ЗАВДАННЯ
на виконання кваліфікаційної роботи магістра

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

студенту *122М-16-1* *Ткач М.Ю.*
(група) (прізвище та ініціали)

Тема дипломної роботи *Обгрунтування синтезу діалогової моделі на основі рекурентної нейронної мережі*

1 ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Наказ ректора Державного ВНЗ «НГУ» від *26.12.2017 р.* № *2127 -л*

2 МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Об'єкт досліджень – проектування нейронної діалогової моделі на базі рекурентної нейронної мережі.

Предмет досліджень – нейронна діалогова модель.

Мета НДР – підвищення точності навчання моделі та моделювання при вирішенні задач проектування діалогової моделі за рахунок використання нейронних мереж.

Вихідні дані для проведення роботи – теоретичні й експериментальні дослідження, основи навчання та моделювання при вирішенні задач проектування діалогової моделв.

3 ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Актуальність даної теми зумовлена наявністю значних недоліків у традиційному підході до моделювання діалогу.

Наукова новизна результатів, що очікуються, полягає у розробці ефективної структури діалоговох моделі.

Практична цінність результатів полягає у розробці алгоритму, який дозволяє оцінити переваги створення інформаційних технологій, на основі застосування діалогової моделі.

4 ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Результати магістерської роботи повинні відповідати вимогам паспорту наукової спеціальності 05.13.06 – «Інформаційні технології».

Результати досліджень мають бути подані у вигляді, що дозволяє побачити та оцінити безпосереднє використання методики розробки систем штучного інтелекту. Згідно виробничих функцій та професійних задач магістра, повинна бути розроблена ефективна структура нелінійної моделі та алгоритм її адаптації при ідентифікації нелінійних нестаціонарних об'єктів.

5 ЕТАПИ ВИКОНАННЯ РОБІТ

Найменування етапів робіт	**Строки виконання робіт (початок-кінець)**
1	2
Аналіз стану питання	10.09.2017 25.09.2017
Основи нейронних мереж	30.09.2017 18.10.2017
Конфігурація та результати моделювання	25.10.2017 23.11.2017

6 РЕАЛІЗАЦІЯ РЕЗУЛЬТАТІВ ТА ЕФЕКТИВНІСТЬ

Економічний ефект від реалізації результатів роботи очікується позитивним завдяки скорочення часу та затрат на розробку програмного забезпечення, інформаційних систем.

Соціальний ефект від реалізації результатів роботи очікується позитивним завдяки удосконаленню діалогові системи, що дозволяє підвищити точність самонавчання.

7 ДОДАТКОВІ ВИМОГИ

Відповідність оформлення ДСТУ 3008-95. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення.

Завдання видав _____ *Мещеряков Л.І.*
(підпис) (прізвище, ініціали)

Завдання прийняв до виконання _____ *Ткач М.Ю.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *09.09.2017р.*

Термін подання дипломного проекту до ДЕК *25.01.2018*

Реферат

Пояснювальна записка: с. 102, рис. 37, додатків. 3, джерела 50.

Об'єкт дослідження: нейронна діалогова модель.

Мета магістерської роботи: реалізація нейронної діалогової моделі на мові Python, використовуючи рекурентні нейронну мережу і бібліотеку машинного навчання TensorFlow.

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

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

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

Значення роботи та висновки. Удосконалена методика дозволяє проектувати діалогові системи зі значним скороченням як матеріальних витрат, так і тимчасових, що підтверджується розробленим програмним продуктом в даній магістерській роботі.

Прогнози щодо розвитку досліджень. Розробити універсальні програмні модулі, які можуть бути використані для підтримки проектування ділогових систем з різних програмних платформ. Розробити комплекс програмних засобів і призначений для користувача інтерфейс для графічного представлення результатів, ведення ділогу зі штучним інтелектом з використанням рекурентної нейронної мережі.

У розділі «Економіка» проведені розрахунки трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки, а також провести маркетингові дослідження ринку збуту створеного програмного продукту.

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

Реферат

Пояснительная записка: с. 102, рис. 37, приложений. 3, источники 50.

Объект исследования: нейронная диалоговая модель.

Цель магистерской работы: реализация нейронной диалоговой модели на языке Python, используя рекуррентные нейронную сеть и библиотеку машинного обучения TensorFlow.

Методы исследования. При решении поставленной задачи использовались научные достижения в областях искусственного интеллекта и нейронных сетей.

Научная новизна полученных результатов заключается в проведении анализа и выявлении недостатков обучении модели ввести диалог, а также в реализации нейронной диалоговой модели на основе использования рекурентной нейронной сети.

Практическая ценность исследования заключается в создании программных модулей, программного продукта, которые позволяют оценить преимущества машинного обучения на базе нейронных сетей.

Область применения. Разработанная диалоговая система может применяться для решения широкого спектра задач, в частности, для машинного обучения, ведения диалога с искусственным интеллектом.

Значение работы и выводы. Усовершенствованная методика позволяет проектировать диалоговые системы со значительным сокращением как материальных затрат, так и временных, что подтверждается разработанным программным продуктом в данной магистерской работе.

Прогнозы по развитию исследований. Разработать универсальные программные модули, которые могут быть использованы для поддержки проектирования дилоговых систем из программных платформ. Разработать комплекс программных средств и пользовательский интерфейс для графического представления результатов, ведения дилогу с искусственным интеллектом с использованием рекуррентной нейронной сети.

В разделе «Экономика» проведены расчеты трудоемкости разработки программного обеспечения, затрат на создание ПО и длительности его разработки, а также провести маркетинговые исследования рынка сбыта созданного программного продукта.

Список ключевых слов: нейронная сеть, рекурентная нейронная сеть, искусственный интеллект, машинное обучение, диалоговая модель, PYTHON, TENSORFLOW, NUMPY, CUDA, PIP, UBUNTU, PYCHARM.

The abstract

Explanatory note: p. 102, rice. 37, applications. 3, sources 50.

Object of research: neural dialogue model.

The purpose of the degree project: the implementation of the neural dialogue model in the language of Python, using the recurrent neural network and the library of machine learning TensorFlow.

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

The scientific novelty of the results obtained is to carry out the analysis and identify the disadvantages of teaching the model to enter the dialogue, as well as in the implementation of the neural dialogue model based on the use of the recurrent neural network.

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

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

The value of the work and conclusions. The advanced technique allows designing dialog systems with significant reductions both material costs and temporary, which is confirmed by the developed software product in this master's thesis.

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

In section "Economics" calculations of the complexity of software development, expenses for software development and duration of its development, as well as marketing researches of the market of the created software product are carried out.

List of keywords: NERON NETWORK, RECURRENT NEURAL NETWORK, ARTIFICIAL INTELLIGENCE, MACHINE EDUCATION, DIALOGUE MODEL, PYTHON, TENSORFLOW, NUMPY, CUDA, PIP, UBUNTU, PYCHARM.

Зміст

Перелік скорочень

ШІ –	штучний інтелект
ОС –	операционная система
ПО –	программное обеспечение
Seq2seq –	sequence2sequence
TF –	TensorFlow
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 рекуррентной нейронної мережі і бібліотеки машинного навчання TensorFlow.

Об'єкт дослідження – нейронна діалогова модель.

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 рекуррентной нейронної мережі і бібліотеки машинного навчання TensorFlow

Обгрунтованість і достовірність наукових положень

Обгрунтованість і достовірність наукових положень, висновків і рекомендацій магістерської роботи обгрунтована коректністю поставлених проблем і прийнятих припущень при математичному описі процесів, обґрунтованістю вихідних посилок, достатнім обсягом вибірки даних і верифікованими на модельних об'єктах результатами обчислень.

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

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

Зв'язок роботи з державними програмами, планами науково-дослідних робіт.

Результати дипломної роботи можуть бути використані підприємствами, фірмами, розробниками для проектування інформаційних систем, створення програмного продукту, що використовує бази даних.

Особливий внесок магістра складається в:

- виборі методів досліджень і технологій реалізації;
- створення інформаційної системи, що реалізує механізми модельно-орієнтованого підходу;

- розробці теоретичної частини роботи, в якій досліджені і систематизовані знання про існуючі підходи розробки інформаційних систем і модельно-орієнтованого;
- оцінці отриманих результатів.

Апробація результатів магістерської роботи.

Основні положення і результати були докладені та обговорені на студентській науковій конференції.

Структура і обсяг роботи. Робота складається з вступу, трьох розділів і висновків. Містить сторінок друкованого тексту, в тому числі сторінки тексту основної частини з малюнками, списку використаних джерел з найменуваннями на сторінках, додатків на сторінках.

РОЗДІЛ 1.

АНАЛІЗ СТАНУ ПИТАННЯ «ОСОБЛИВОСТІ ВЕДЕННЯ БЕСІДИ ЗІ ШТУЧНИМ ІНТЕЛЕКОМ»

1.1. Визначення штучного інтелекту

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

Відображення в XXI столітті розвитку штучного інтелекту, його вдосконалення все більше знаходить рішення в нейронних мережах. Якщо наблизити поняття ІІ до людського фактору, то можна знайти визначення нейронних мереж. Нейронні мережі будуються за принципом організації та функціонування біологічних нейронних мереж, тобто мереж нервових клітин живого організму. Поняття біологічних нейронних мереж виникло при вивченні процесів, що протікають в мозку і спробі їх моделювання. Так спроба моделювання біологічних нейронних мереж привела до розробки алгоритмів, де згодом отримані моделі на основі відомих і розроблених алгоритмів стали використовувати в задачах ідентифікації об'єктів управління для практичних цілей. Нейронні мережі являють собою систему багатозв'язних між собою процесорів - штучних нейронів, які є досить простими за своєю структурою. Подібні процесори нейронної мережі обробляють сигнали як надходять на них, так і сигнали, надіслані ними на інші процесори нейронної мережі. Будучи з'єднаними в досить велику мережу з керованим взаємодією, такі локально прості процесори разом здатні виконувати досить складні завдання.

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

спеціальних комп'ютерах - нейрокомп'ютер з використанням відповідного програмного забезпечення.

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

Різниця між реальним виходом нейронної мережі та мають місце бажаним відгуком називається помилкою. Величина і знак цієї помилки служать для самоадаптації вагових коефіцієнтів, які на старті навчання задаються випадковими числами. Механізм процесу навчання полягає в цілеспрямованому зміні (самоадаптації) вагових коефіцієнтів (іноді і деяких додаткових параметрів) і називається правилом або алгоритмом навчання. Критерієм навченості є планомірне, асимптотическое зменшення середньоквадратичної помилки, тобто головне, щоб вона зменшувалася з кожним новим повтором нікого прикладу. Навчання нейронних мереж завершується, коли середньоквадратична помилка досягне деякої наперед заданої величини або коли нейронна мережа почне правильно (з необхідною точністю) обробляти дані з окремого тестового набору характерних прикладів.

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

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

1.2. Віртуальний співрозмовник

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

Одним з перших віртуальних співрозмовників була програма Еліза, створена в 1966 році Джозефом Вейзенбаумом. Еліза пародіювала мовна поведінка психотерапевта, реалізуючи техніку активного слухання, перепитуючи користувача і використовуючи фрази типу «Будь ласка, продовжуйте».

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

1.2.1. Призначення віртуальних співрозмовників

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

Тому функціональність більшості сучасних програм обмежується можливістю ведення невигадливій бесіди.

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

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

1.2.2. Принцип дії діалогових агентів

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

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

- реакція на ключові слова: Даний метод був використаний в Елізе. Наприклад, якщо фраза користувача містила слова «батько», «мати», «син» і інші, Еліза могла відповісти: «Розкажіть більше про вашу сім'ю»;

- збіг фрази: мається на увазі схожість фрази користувача з тими, що містяться в базі знань. Може враховуватися також порядок слів;

- збіг контексту: часто в інструкціях до програм-співрозмовникам просять не використовувати фрази, насичені займенниками, типу: «А що це таке?» Для коректної відповіді деякі програми можуть проаналізувати попередні фрази користувача і вибрати найбільш прийнятних відповідей.

Своєрідною міні-проблемою є ідентифікація форм слова і синонімів.

1.3. Особливості та проблеми віртуального спілкування

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

Основними питаннями в створенні систем спілкування є розробка моделі спілкування, моделі учасника спілкування, розвиток засобів, в першу чергу, семантичних і прагматичних, опису навколишнього середовища (моделі мови, моделі користувача, моделі навколишнього середовища, моделі системи

спілкування). Тому для вирішення цих питань необхідно визначення принципів роботи, особливостей імітації мовної поведінки людини в процесі спілкування, розробка моделі спілкування, написання чат-бота. Серед програм-співрозмовників є програми, створені на основі штучного інтелекту.

При розробці таких програм необхідно знати психологію, а також принципи побудови фраз людської мови. Більш того, якщо правильно визначити мовні обмеження і предметну область, то існуючими методами можна отримати системи, придатні для спілкування. З точки зору теорії мови і спілкування необхідна розробка семантичного опису структур текстів і пропозицій. З точки зору моделі навколишнього середовища основним обмеженням є відсутність коштів для подання динамічно мінливого світу. Ця пов'язано з проблемою сприйняття системою тверджень і навчанням 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 використовують для введення-виведення повідомлень і виконання різних функцій. Боти виконують такі основні функції: службові, інформаційно розважальні, функції утиліт.Розглянемо їх докладніше. Службові функції ботів полягають у веденні логів чату обліку прав учасників, забезпеченні заходів безпеки, забезпеченні можливості конференції між більш ніж двома користувачами, коли в протоколі відсутня така функція.Інформаційно-розважальними функціями, які забезпечують бот, можуть бути довідка, словники, віртуальні співрозмовники, ігри.Також в боті використовують утиліти, наприклад, перекладач, калькулятор, коментатор, пошук.Принцип роботи чат-бота полягає в реалізації етапів: бот приймає вхідні повідомлення, аналізує їх і відсилає результат виконання і / або виконує команду.

Висновок

Отже, спілкування в чат-ботах здійснюється шляхом введення повідомлень і виведення відповіді (думки) співрозмовника. Тут можливі два види ведення розмови: звичайна бесіда або обговорення важливого питання. Але, на відміну від розмови людей, програма не володіє гнучким інтелектом, тому більшість віртуальних співрозмовників запрограмовані на ведення нескладної бесіди.

Такі програми відносять до класу програм з природним мовним інтерфейсом. Обробка природної мови людини, особливо розмовного стилю, є проблемою, що стосується штучного інтелекту. Проблема створення програм співрозмовників на базі штучного інтелекту, які можуть моделювати інтелектуальну діяльність людини, на сьогоднішній день залишається відкритим.

На жаль, сучасні віртуальні співрозмовники лише частково вирішують питання імітації розмови людини. Основу їх функціонування складає база знань. У найпростішому випадку вона містить набори можливих питань користувача і відповідних відповідей на них. Найбільш поширені методи вибору відповідей в даному випадку такі: реакція на ключові слова (наприклад, якщо фраза користувача містила слова «яблуко», «зливу», «груша», програма може відповісти «Ви любите фрукти?"); співпадання користувача з тієї, яка є в базі знань; також програма може враховувати порядок слів (наприклад, якщо питання користувача: «Які фрукти містять більше вітаміну С?», програма може відповісти «Цитрусові.»).Програми співрозмовники не можуть використовувати фрази, насичені займенниками, наприклад, «Наскільки він Ваш?».У таких випадках програми аналізують попередні фрази користувача і вибирають найбільш прийнятний відповідь.Також проблематичним може бути підбір слів-синонімів.

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

РОЗДІЛ 2.

ВИКОРИСТАННЯ РЕКУРЕНТНИХ НЕЙРОННИХ МЕРЕЖ ПРИ МОДЕЛЮВАННІ РОЗМОВИ

2.1. Штучні неронні мережі

Інтелектуальні системи на основі штучних нейронних мереж дозволяють з успіхом вирішувати проблеми розпізнавання образів, виконання прогнозів, оптимізації, асоціативної пам'яті і управління. Традиційні підходи до вирішення цих проблем не завжди дають необхідну гнучкість і багато додатків виігивают від використання нейромереж.

Штучні нейромережі є електронними моделями нейронної структури мозку, який, головним чином, вчиться на досвіді. Природною аналог доводить, що безліч проблем, що не піддаються вирішенню традиційними комп'ютерами, можуть бути ефективно вирішені за допомогою нейромереж.

Тривалий період еволюції додав мозку людини багато якостей, відсутніх в сучасних комп'ютерах з архітектурою фон Неймана. До них відносять:

- розподілене представлення інформації і паралельні обчислення
- здатність до навчання і узагальнення
- адаптивність
- толерантність до помилок
- низьке енергоспоживання

Прилади, побудовані на принципах біологічних нейронів, мають перераховані характеристики, які можна вважати суттєвим досягненням в індустрії обробки даних.

Досягнення в галузі нейрофізіології дають початкове розуміння механізму природного мислення, де зберігання інформації відбувається у вигляді складних образів.Процес зберігання інформації як образів, використання образів і рішення поставленої проблеми визначають нову область в обробці даних, яка, не використовуючи традиційного програмування, забезпечує створення паралельних мереж і їх навчання. У лексиконі розробників та користувачів нейромереж присутні слова, відмінні від традиційної обробки даних, зокрема, "вести себе", "реагувати", "самоорганізовувати", "навчати", "узагальнювати" та "забувати".

2.1.1. Біологічний нейрон

Нейрон (нервова клітина) складається з тіла клітини - соми (soma), і двох типів зовнішніх деревоподібних відгалужень: аксона (axon) і дендритів (dendrites). Тіло клітини містить ядро (nucleus), де знаходиться інформація про властивості нейрона, і плазму, яка виробляє необхідні для нейрона матеріали. Нейрон отримує сигнали (імпульси) від інших нейронів через дендрити (приймача) і передає сигнали, згенеровані тілом клітки, вздовж аксона (передавач), що наприкінці розгалужується на волокна (strands). На закінченнях волокон знаходяться синапси (synapses).

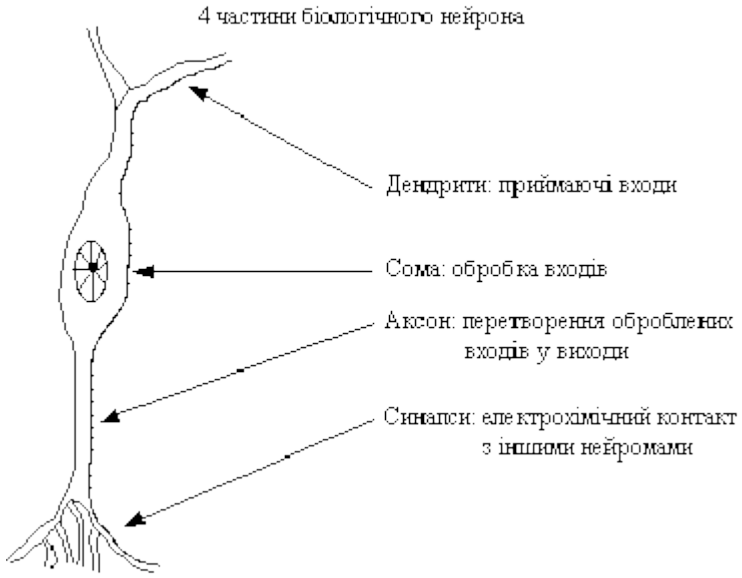

Рис. 2.1. Схема біологічного нейрона.

Синапс є функціональним вузлом між двома нейронами (волокно аксона одного нейрона і дендрит іншого). Коли імпульс досягає синаптичного закінчення, виробляються хімічні речовини, названі нейротрансмітерами. Нейротрансмітерів проходять через синаптичну щілину, і в залежності від типу синапсу, збуджуючи або гальмуючи здатність нейрона-приймача генерувати електричні імпульси. Результативність синапсу налаштовується проходять через нього сигналами, тому синапси навчаються в залежності від активності процесів, в яких вони беруть участь. Нейрони взаємодіють за допомогою короткої серії імпульсів.Повідомлення передається за допомогою частотно-імпульсної модуляції.

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.

2.1.2. Штучний нейрон

Базовий модуль нейронних мереж - штучний нейрон моделює основні функції природного нейрона (рис. 2.2).

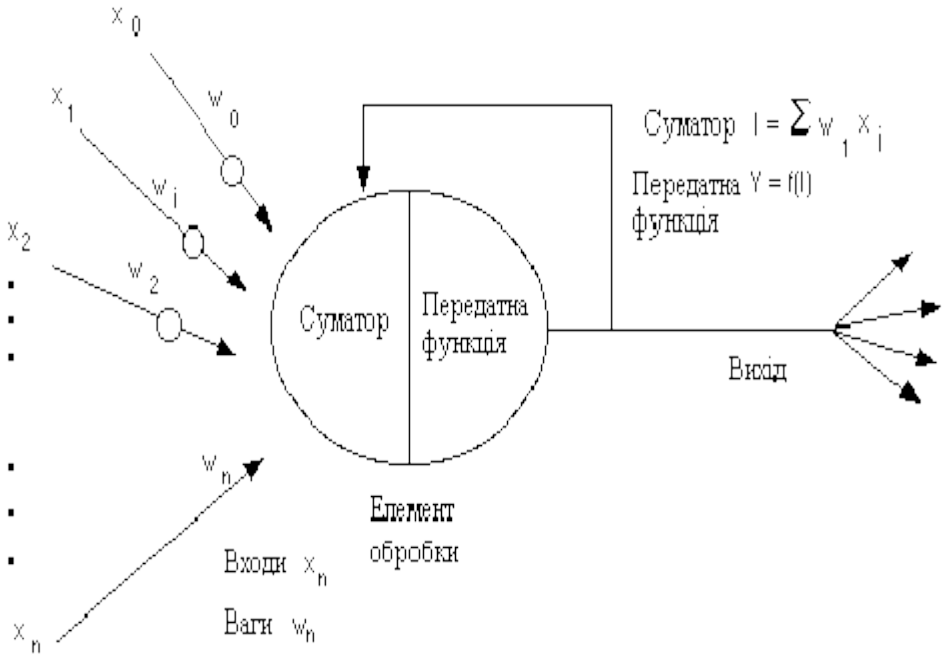

Рис. 2. 2. Базовий штучний нейрон

Вхідні сигнали xn зважені ваговими коефіцієнтами з'єднання wn складаються, проходять через передатну функцію, генерують результат і виводяться.

В існуючих нині пакетах програм штучні нейрони називаються "елементами обробки" і мають більше можливостей, ніж простий штучний нейрон, описаний вище. На рис. 2.3 зображена детальна схема спрощеного штучного нейрона.

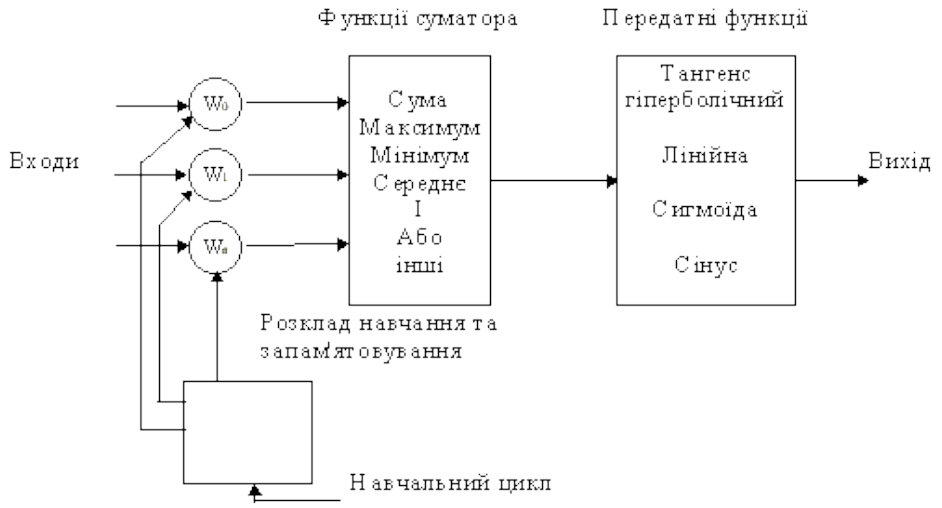

Рис. 2.3. Модель нейрона

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.п., 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 ускладнюється 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 нейросетях як передавальних функцій можуть бути використані сигмоїда, синус, гіперболічний тангенс і ін. Приклад того, як працює передавальна функція показаний на рис. 2.4..

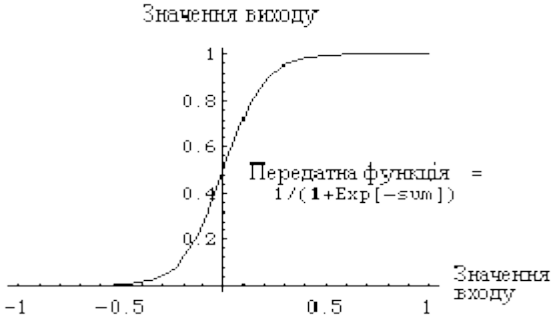

Рис. 2.4. Сігмоіднаяя передавальна функція

Всі штучні нейромережі конструюються з базового блоку - штучного нейрона. Існуючі різноманітності і відмінності, є підставою для мистецтва талановитих розробників при реалізації ефективних нейромереж.

2.1.3. Штучні нейронні мережі

Інша частина створення і використання нейромереж стосується великої кількості зв'язків, що пов'язують окремі нейрони. Групування в мозку людини відбувається так, що інформація обробляється динамічним, інтерактивним і систем, що самоорганізуються шляхом.Біологічні нейронні мережі створені в тривимірному просторі з мікроскопічних компонентів і здатні до різноманітних з'єднань, а для створеної людиною мережі існують фізичні обмеження.

Існуючі на даний час, нейромережі є групуванням штучних нейронів, у вигляді з'єднаних між собою шарів.

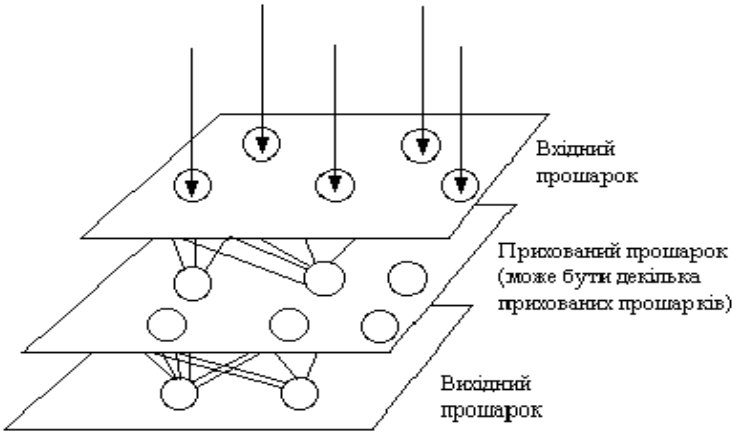

Рис. 2.5. Діаграма простий нейронної мережі

На рис. 2.5 показана типова структура штучних нейромереж. Хоча існують мережі, які містять лише один шар, або навіть один елемент, більшість реалізацій використовують мережі, що містять як мінімум три типи шарів - вхідний, прихований і вихідний. Шар вхідних нейронів отримує дані або з вхідних файлів, або безпосередньо з електронних датчиків. Вихідний шар пересилає інформацію безпосередньо в зовнішнє середовище, до вторинного комп'ютерного процесу, або до іншого пристрою. Між цими двома шарами може бути кілька прихованих шарів, що містять багато різноманітно пов'язаних

нейронів. Входи і виходи кожного з прихованих нейронів з'єднані з іншими нейронами.

Напрямок зв'язку від одного нейрона до іншого є важливим аспектом нейромереж. У більшості мереж кожен нейрон прихованого шару отримує сигнали від всіх нейронів попереднього шару і зазвичай від нейронів вхідного шару. Після виконання операцій над сигналами, нейрон передає свій вихід всім нейронам наступних шарів, забезпечуючи передачу сигналу вперед (feedforward) на вихід.

При зворотного зв'язку, вихід нейронів шару прямує до нейронам попереднього шару (рис. 2.6).

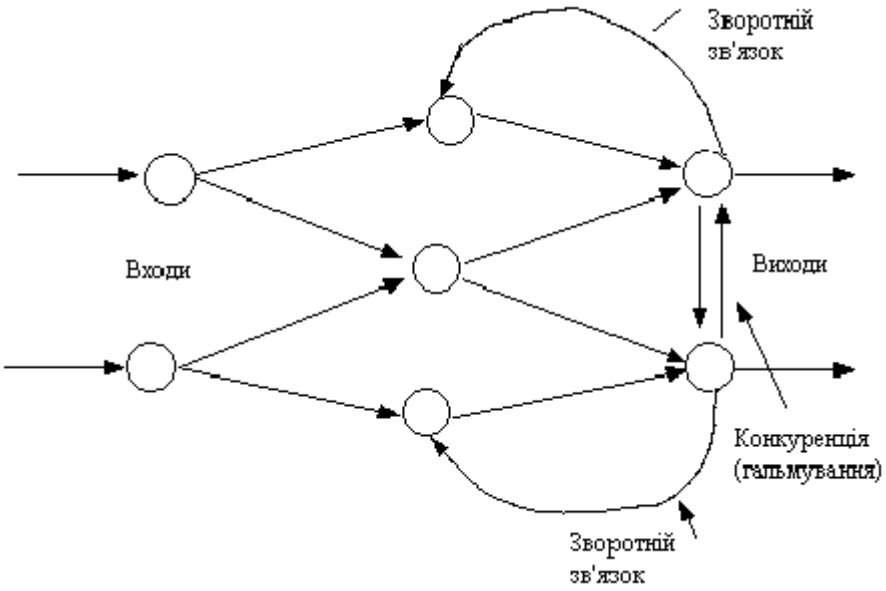

Рис. 2.6. Напрямок зв'язків між нейронами

Вид з'єднання між нейронами має великий вплив на роботу мережі. Більшість пакетів програмних реалізацій нейронних мереж дозволяють користувачеві додавати, віднімати і керувати з'єднаннями як завгодно. Постійно коректіруемие параметри зв'язку можна зробити як збудливими так і гальмують.

2.1.4. Навчання штучної нейронної мережі

Фундаментальною властивістю мозку є здатність до навчання. Процес навчання може розглядатися як визначення архітектури мережі і налаштування ваг зв'язків для ефективного виконання спеціального завдання. Нейросеть налаштовує ваги зв'язків під наявне навчальну множину.

Для процесу навчання необхідно мати модель зовнішнього середовища, в якому функціонує нейронна мережа - потрібну для вирішення завдання інформацію. По-друге, необхідно визначити, як налаштовуються ваги зв'язків мережі. Алгоритм навчання означає процедуру, в якій використовуються правила навчання для настроювання ваг.

Існують три види навчання: "з учителем", "без вчителя" (самонавчання) і змішане. У першому випадку нейросеть має в своєму розпорядженні правильні відповіді (виходи мережі) на кожен вхідний приклад. Ваги налаштовуються так, щоб мережа виробляла відповіді близькі до відомих правильних відповідей. Навчання без вчителя не вимагає знання правильних відповідей на кожен приклад навчальної вибірки. У цьому випадку використовується внутрішня структура даних і кореляція між зразками в навчальній множині для розподілу зразків за категоріями. При змішаному навчанні частина ваг визначається за допомогою навчання з учителем, в той час як інша визначається за допомогою самонавчання.

2.2. Нейронна розмовна модель

2.2.1. Моделювання розмови

Досягнення в наскрізному (end-to-end) навчанні нейронних мереж привели до помітного прогресу в багатьох областях, таких як розпізнавання мови, комп'ютерний зір і обробка мови. Недавні досягнення припускають, що нейронні мережі можна використовувати не тільки для задач класифікації, їх також можна використовувати для зіставлення одних складних структур з іншими складними структурами.Прикладом цього є завдання зіставлення послідовності з іншою послідовністю, яка має пряме застосування в розумінні

природної мови. Основною перевагою цієї структури є те, що вона вимагає невеликої функціональної інженерії та специфіки галузі при зіставленні. Ця перевага, дозволяє дослідникам працювати над завданнями, для яких знання області можуть бути недоступні.

2.2.2. Модель

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

Конкретно, припустимо, що ми спостерігаємо розмову: перша людина вимовляє «ABC», а друга людина відповідає «WXYZ. Ми можемо використовувати рекуррентную нейронну мережу і тренувати її для відображення «ABC» на «WXYZ», як показано на малюнку 1 нижче.

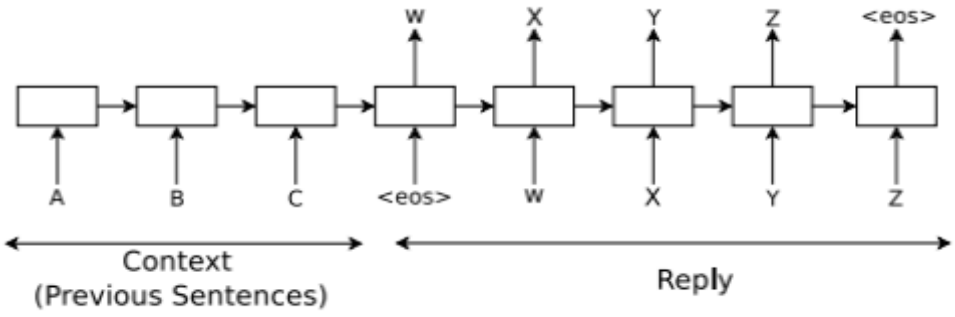

Рис 2.7. Використання seq2seq для моделювання діалогу

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

2.3. Рекурентні нейромережі і чим вони відрізняються від звичайних

Давайте спочатку згадаємо, що таке «звичайні» нейромережі, і тоді відразу стане зрозуміло, чим вони відрізняються від реккурентное. Уявімо собі найпростішу нейросеть - перцептрон. Він являє собою один шар нейронів, кожен з яких приймає шматочок вхідних даних (один або кілька бітів, дійсних чисел, пікселів і т.п.), модифікує його з урахуванням власної ваги і передає далі. В одношаровому перцептроном видача всіх нейронів об'єднується тим чи іншим чином, і нейросеть дає відповідь, але можливості такої архітектури сильно обмежені. Якщо ви хочете отримати більш просунутий функціонал, можна піти декількома шляхами, наприклад, збільшити кількість шарів і додати операцію згортки, яка б «розшаровується» входять дані на шматочки різних масштабів. В цьому випадку у вас вийдуть згорткові нейромережі для глибинного навчання, які досягли успіху в обробці зображень і розпізнаванні

котиків. Однак що у примітивного перцептрону, що у сверточное нейромережі є загальне обмеження: і вхідні і вихідні дані мають фіксований, заздалегідь визначений розмір, наприклад, картинка 100 × 100 пікселів або послідовність з 256 біт. Нейросеть з математичної точки зору поводиться як звичайна функція, хоч і дуже складно влаштована: у неї є заздалегідь позначене число аргументів, а також позначений формат, в якому вона видає відповідь. Простий приклад - функція x2, вона приймає один аргумент і видає одне значення.

Перераховані вище особливості технічно нескладне значних труднощів, якщо мова йде про тих же картинках або заздалегідь визначених послідовності символів. Але що, якщо ви хочете використовувати нейросеть для обробки тексту або музики? У загальному випадку - будь-який умовно нескінченної послідовності, в якій важливо не тільки зміст, а й порядок, в якому слід інформація.Ось для цих завдань і були придумані рекурентні нейромережі. Їх протилежності, які ми називали «звичайними», мають більш суворе назва - нейромережі прямого поширення (feed-forward neural networks), так як в них інформація передається тільки вперед по мережі, від шару до шару. У рекурентних нейромереж нейрони обмінюються інформацією між собою: наприклад, до того ж до нового шматочку входять даних нейрон також отримує деяку інформацію про попередньому стані мережі. Таким чином в мережі реалізується «пам'ять», що принципово змінює характер її роботи і дозволяє аналізувати будь-які послідовності даних, в яких важливо, в якому порядку йдуть значення - від з вукозапісей до котирувань акцій.

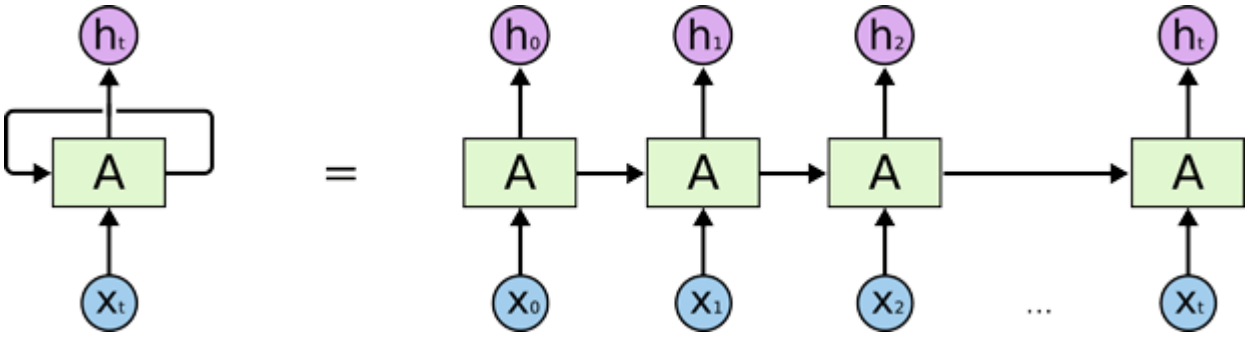

Рис. 2.8. Схема однойслойной рекуррентной нейронної мережі: на кожному циклі роботи внутрішній шар нейронів отримує набір вхідних даних X і

інформацію про попередньому стані внутрішнього шару A, на підставі чого генерує відповідь h.

Наявність пам'яті у зворотних нейромереж дозволяє дещо розширити нашу аналогію з x2. Якщо нейромережі прямого поширення ми назвали «простий» функцією, то рекурентні нейромережі можна майже з чистою совістю назвати програмою. Справді, пам'ять рекурентних нейромереж (хоча і не повноцінна, але про це пізніше) робить їх Тьюринг-повними: при правильному завданні ваг нейросеть може успішно емулювати роботу комп'ютерних програм.

2.3.1. Для яких завдань використовується РНС і в чому перевага перед звичайним перцептроном

Ймовірно, першою РНС була мережа Хопфілда (вперше згадана в 1974 році, остаточно оформилася в 1982-му), яка реалізовувала на практиці осередок асоціативної пам'яті. Від сучасних РНС вона відрізняється тим, що працює з послідовностями фіксованого розміру. У найпростішому випадку мережа Хопфілда має один шар внутрішніх нейронів, пов'язаних між собою, а кожна зв'язок характеризується певною вагою, що задає її значимість. З такою мережею асоціюється якийсь еквівалент фізичної «енергії», який залежить від всіх ваг в системі. Мережа можна навчити за допомогою градієнтного спуску по енергії, коли мінімум відповідає стану, в якому мережу «запам'ятала» певний шаблон, наприклад 10101. Тепер, якщо їй на вхід подати спотворений, зашумленний або неповний шаблон, скажімо, 10000, вона «згадає» і відновить його аналогічно тому, як працює асоціативна пам'ять у людини. Ця аналогія досить віддалених, тому не варто сприймати її надто серйозно. Проте, мережі Хопфілда успішно справлялися зі своїм завданням і обходили за можливостями існуючі тоді перцептрони. Цікаво, що оригінальна публікація Джона Хопфилда в Proceedings of the National Academy of Sciences вийшла в розділі «Біофізика».

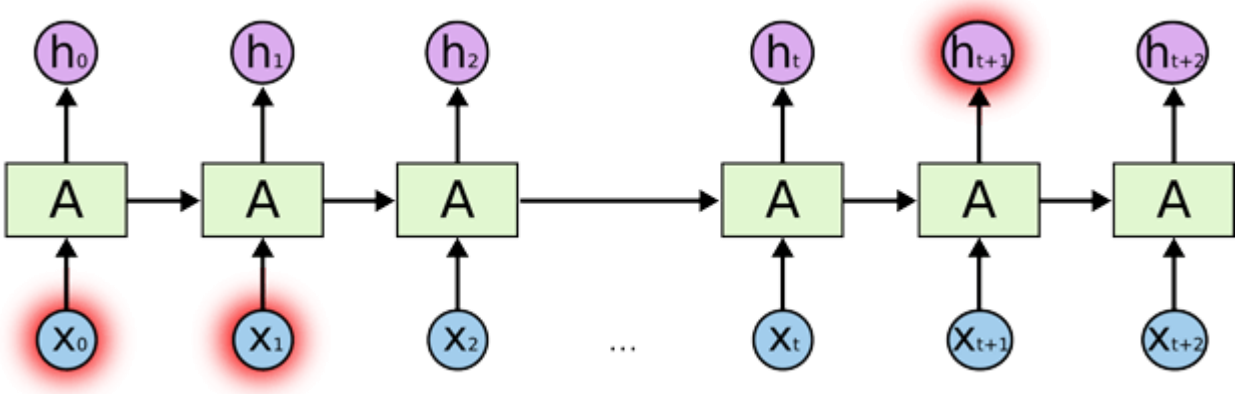

Рис. 2.9. Проблема довгостроковій пам'яті в простих РНС: чим більше циклів пройшло з моменту отримання тієї чи іншої інформації, тим більша ймовірність, що значимість цих даних не буде грати великої ролі на новому циклі роботи.

Наступним кроком в еволюції РНС була «проста рекуррентная мережу» Джеффа Елмана, описана в 1990 році. У ній автор детально торкнувся питання про те, як можна (і чи можна взагалі) навчити нейромережа розпізнавати тимчасові послідовності. Наприклад, якщо є вхідні дані 1100 і 0110, чи можна їх вважати одним і тим же набором, зсунутим в часі? Звичайно, можна, але як навчити цьому нейросеть? Звичайний перцептрон легко запам'ятає цю закономірність для будь-яких прикладів, які йому запропонують, але кожен раз це буде завданням порівняння двох різних сигналів, а не завданням про еволюцію або зсуві одного і того ж сигналу. Рішення Елмана, засноване на попередніх напрацювань в цій галузі, грунтувалося на тому, що в просту нейросеть додавався ще один - «контекстний» - шар, в який просто копіювалося стан внутрішнього шару нейронів на кожному циклі роботи мережі. При цьому зв'язок між контекстним і внутрішнім шарами можна було навчати. Така архітектура дозволяла порівняно легко відтворювати тимчасові ряди, а також обробляти послідовності довільної довжини, що різко відрізняло просту РНС Елмана від попередніх концепцій. Більш того, ця мережа змогла розпізнати і навіть класифікувати іменники і дієслова в реченні, грунтуючись тільки на

порядку слів, що було справжнім проривом для свого часу і викликало величезний інтерес як лінгвістів, так і фахівців з дослідження свідомості.

За простій РНС Елмана пішли всі нові розробки, а в 1997 році Хохрейтер і Шмідхубер опублікували статтю «Long Short-term memory» («довгострокова короткострокова пам'ять», також існує безліч інших варіацій перекладу), що заклав основу для більшості сучасних РНС. У своїй роботі автори описували модифікацію, вирішувати проблему довгостроковій пам'яті простих РНС: їх нейрони добре «пам'ятають» недавно отриману інформацію, але не мають можливості надовго зберегти в пам'яті щось, що обробили багато циклів назад, якою б важливою та інформація не була. У LSTM-мережах внутрішні нейрони «обладнані» складною системою так званих воріт (gates), а також концепцією клітинного стану (cell state), яка і являє собою якийсь вид довгострокової пам'яті. Ворота ж визначають, яка інформація потрапить в клітинне стан, яка зітреться з нього, і яка вплине на результат, який видасть РНС на даному етапі. Детально розбирати LSTM ми не будемо, однак відзначимо, що саме ці варіації РНС широко використовується зараз, наприклад, для машинного перекладу Google.

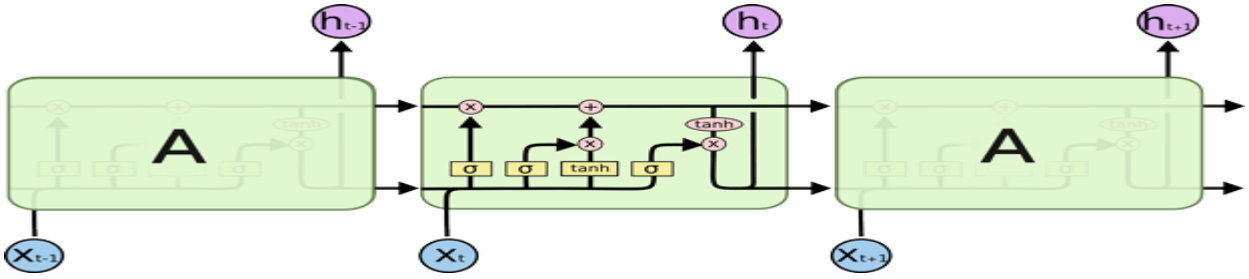

Рис. 2.10. Принцип роботи РНС типу LSTM: нейрони внутрішніх шарів можуть зчитувати і змінювати стан осередку (cell state), яке поєднує в собі функції короткостроковій і довгостроковій пам'яті.

2.3.2. Області застосування РНС

Одна з головних областей застосування РНС на сьогоднішній день - робота з мовними моделями, зокрема - аналіз контексту і загальної зв'язку слів

в тексті. Для РНС структура мови - це довгострокова інформація, яку треба запам'ятати. До неї відносяться граматика, а також стилістичні особливості того корпусу текстів, на яких проводиться навчання. Фактично РНС запам'ятовує, в якому порядку зазвичай йдуть слова, і може дописати пропозицію, отримавши деяку приманку. Якщо ця приманка випадкова, може вийти зовсім безглуздий текст, стилістично нагадує шаблон, на якому вчилася РНС. Якщо ж вихідний текст був осмисленим, РНС допоможе його стилізувати, проте в останньому випадку однієї РНС буде мало, так як результат повинен являти собою «суміш» випадкового, але стилізованого тексту від РНС і осмисленої, але «неокрашенной» вихідної частини. Це завдання вже настільки нагадує популярні нині програми для обробки фотографій в стилі Моне і Ван Гога, що мимоволі напрошується аналогія.

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

Нарешті, давайте розберемося, чому музику РНС худо-бідно пишуть, а з повноцінними текстами Толстого і Достоєвського виникають проблеми? Справа в тому, що в інструментальній музиці, як би по-варварськи це не звучало, немає сенсу в тому ж значенні, в якому він є в більшості текстів. Тобто музика може подобатися або не подобатися, але якщо в ній немає слів - вона не несе інформаційного навантаження (звичайно, якщо це не секретний код). Саме з доданням своїм творам сенсу і спостерігаються проблеми у РНС: вони можуть чудово вивчити граматику мови і запам'ятати, як повинен виглядати текст в

певному стилі, але створити і донести якусь ідею або інформацію РНС (поки) не можуть.

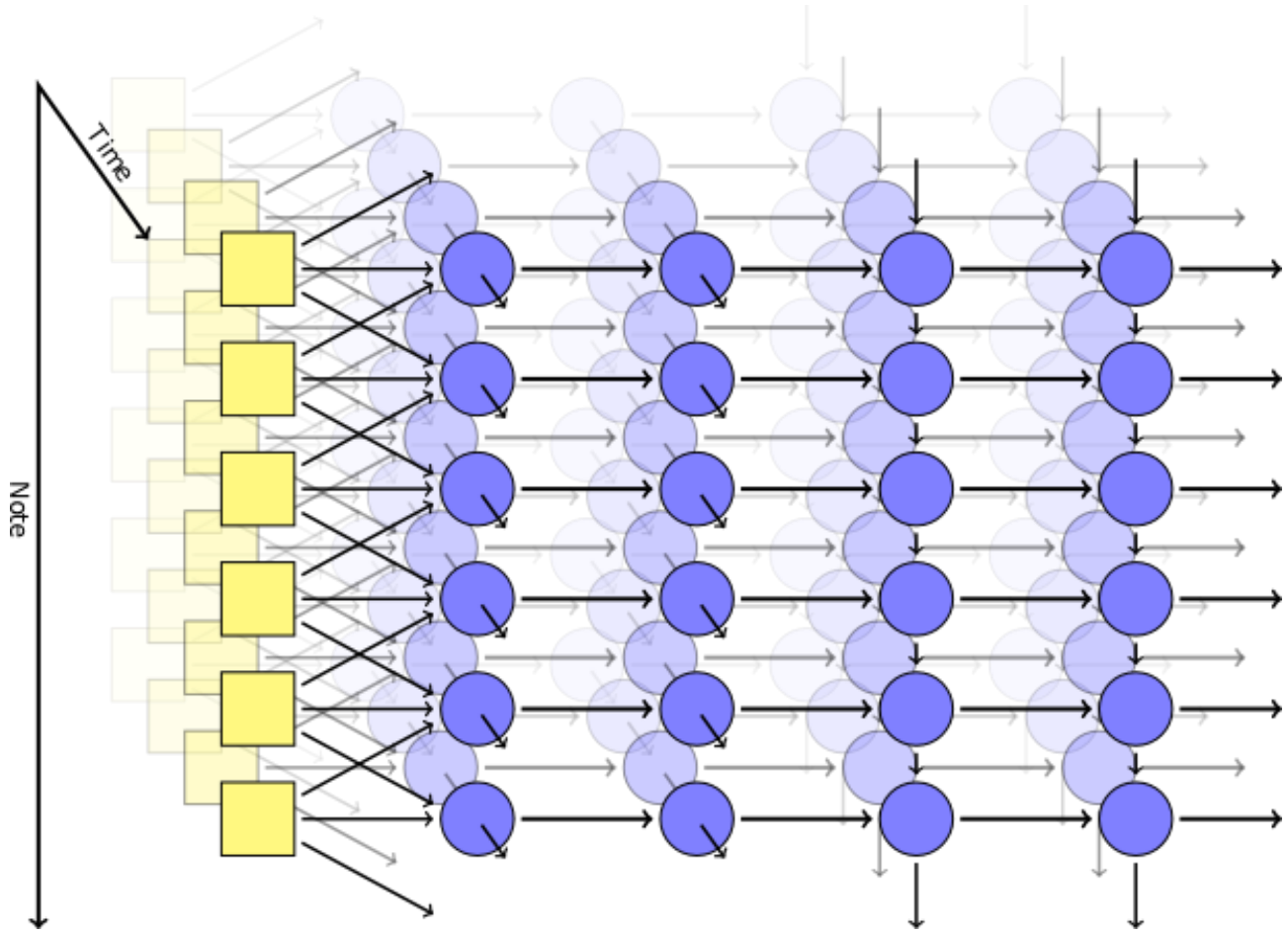

Рис. 2.11. Схема тривимірної рекуррентной нейромережі для написання музичних фрагментів: на відміну від найпростішої архітектури, в даній системі фактично об'єднані дві РНС, окремо описують послідовність в часі і поєднання нот в кожен момент.

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

2.3.3. Застосування РНС

Зрозуміло, РНС, крім розважальних, повинні переслідувати і більш прагматичні цілі. З їх дизайну автоматично випливає, що головні області їх застосування повинні бути вимогливі до контексту і / або тимчасової залежності в даних, що по суті одне і те ж. Тому РНС використовуються, наприклад, для аналізу зображень. Здавалося б, ця область зазвичай сприймається в контексті згортальних нейромереж, однак і для РНС тут знаходяться завдання: їх архітектура дозволяє швидше розпізнавати деталі, грунтуючись на контексті і оточенні. Аналогічним чином РНС працюють в сферах аналізу і генерації текстів. З більш незвичайних завдань можна згадати спроби використовувати ранні РНС для класифікації вуглецевих спектрів ядерного магнітного резонансу різних похідних бензолу, а з сучасних - аналіз появи негативних відгуків про товари.

2.3.4. Застосування РНС в машинному перекладі

На поточний момент в Google для машинного перекладу використовуються РНС типу LSTM, що дозволило домогтися найбільшої точності в порівнянні з існуючими аналогами, проте, за словами самих авторів, машинного перекладу ще дуже далеко до рівня людини. Складнощі, з якими стикаються нейромережі в задачах перекладу, обумовлені відразу декількома факторами: по-перше, в будь-якому завданні існує неминучий розмін між якістю і швидкістю. На даний момент чоловік дуже сильно випереджає штучний інтелект за цим показником. Оскільки машинний переклад найчастіше використовується в онлайн-сервісах, розробники змушені жертвувати точністю на догоду швидкодії. У недавній публікації Google на цю тему розробники детально описують багато рішень, які дозволили оптимізувати поточну версію Google Translate, однак проблема досі залишається. Наприклад, рідкісні слова, або сленг, або навмисне спотворення слова (наприклад, для більш яскравого заголовка) може збити з пантелику навіть перекладача-людини, якій доведеться

витратити час, щоб підібрати найбільш адекватний аналог в іншій мові.Машину ж така ситуація поставить в глухий кут, і перекладач буде змушений «викинути» складне слово і залишити його без перекладу. У підсумку проблема машинного перекладу не настільки обумовлена архітектурою (РНС успішно справляються з рутинними завданнями в цій галузі), наскільки складністю і різноманіттям мови. Радує те, що ця проблема має більш технічний характер, ніж написання осмислених текстів, де, ймовірно, потрібно кардинально новий підхід.

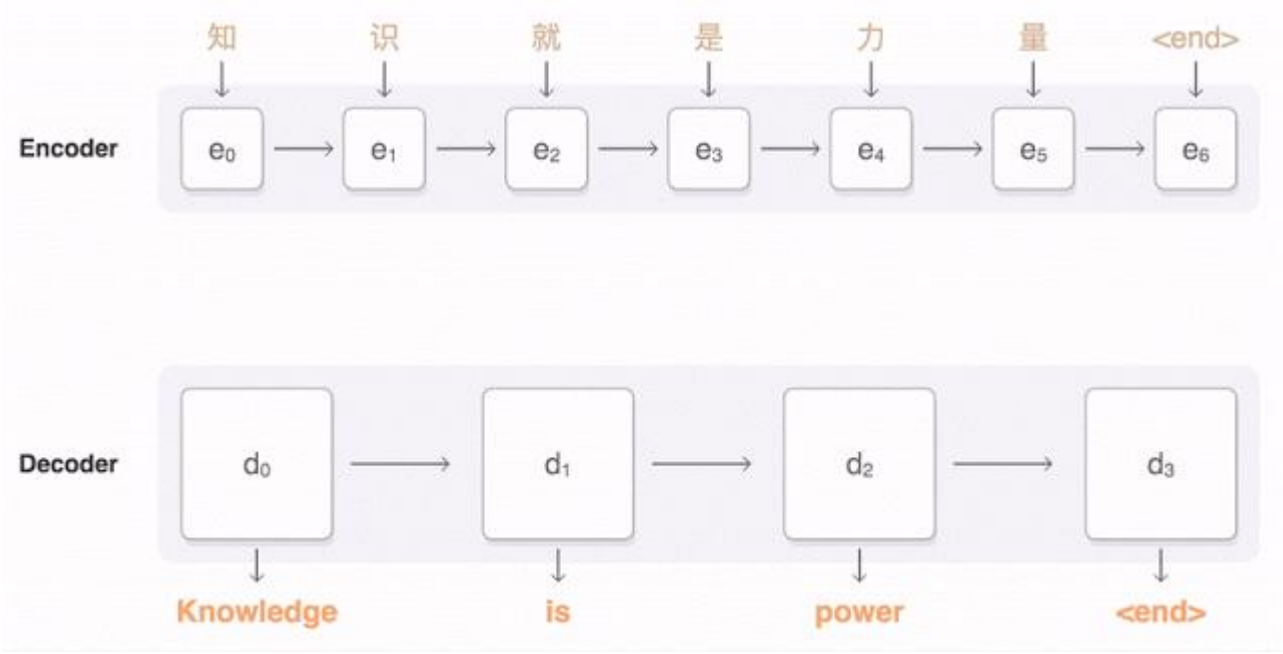

Рис. 2.12. Принцип роботи машинного перекладача Google Translate, заснованого на комбінації кілька рекурентних нейромереж.

2.4. Навчання послідовність-в-послідовність з нейронними мережами

Глибокі нейронні мережі (DNN) - це потужні моделі, які домоглися відмінної продуктивності при складних завданнях навчання. Хоча DNN працюють добре, коли доступні великі марковані навчальні набори, вони не можуть використовуватися для зіставлення послідовностей. У цій роботі представляється загальний наскрізний підхід до послідовного навчання, який

робить мінімальні припущення про структуру послідовності. Цей метод використовує багатошарову довгу короткострокову пам'ять (LSTM) для зіставлення вхідної послідовності з вектором фіксованої розмірності, а потім інший глибокої LSTM для декодування цільової послідовності з вектора. Нарешті, ми виявили, що зміна порядку слів у всіх вихідних пропозиціях (але не на цільові пропозиції) значно поліпшило продуктивність LSTM, оскільки це призвело до появи багатьох короткострокових залежностей між вихідним і цільовим пропозицією, що спростило завдання оптимізації.

2.4.1. Технічні складності

Розмовні агенти повинні задовольняти двом наборам 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

Висновок

Точна і ефективна обробка природної мови має важливе значення для ефективного діалогового агента. Щоб відповісти відповідним чином на висловлювання користувача, 1 говорить агент повинен (1) інтерпретувати висловлювання, (2) визначати дії, які повинні бути зроблені у відповідь на висловлювання, і (3) виконувати дії, які можуть включати відповідь з текстом, уявлення веб- сторінок або іншої інформації і виконання системних дій, таких як запис інформації в базу даних.Наприклад, якщо висловлювання користувача було: (1) я хотів би купити його зараз, агент повинен спочатку визначити буквальне значення висловлювання: користувач хоче щось купити, можливо, щось, згадане раніше в розмові. Крім того, агент повинен вивести цілі, які

користувач намагався виконати, вимовивши висловлювання з цим значенням. Хоча висловлювання користувача в формі твердження, ймовірно, було призначене для вираження запиту на завершення покупки. Як тільки агент інтерпретує заяву, він повинен визначити, як діяти. Відповідні дії залежать від поточної мети агента (наприклад, продажу продуктів або обробки скарг), історії діалогу (попередні заяви, зроблені агентом і користувачем), та інформації в базах даних, доступних агенту, таких як дані про конкретних клієнтів або продуктів. Наприклад, якщо агент має на меті продавати продукти, в попередньому обговоренні було визначено конкретний предмет споживання для продажу на веб-сайті агента, а в каталозі товарів показаний товар, який повинен бути на складі, сприятливу дію може полягати в тому, щоб представити форму замовлення і попросіть користувача виконати його. Якщо замість цього в попередньому обговоренні не було чітко визначено елемент, відповідним дією могло б бути викликано опис конкретного елемента у користувача. Аналогічним чином, якщо елемент був недоступний, відповідні дії можуть полягати в тому, щоб запропонувати користувачеві інший вибір. Нарешті, агент повинен відповісти відповідними діями. Відповідні дії можуть включати в себе складання заяви, подання інформації в інших формах, таких як фотографії продуктів, і прийняття інших дій, таких як ведення журналу інформації в базі даних. Наприклад, якщо відповідне дія мала надати користувачеві форму замовлення і попросити користувача виконати його, агент повинен буде отримати або створити заяву, наприклад «Відмінно!Будь ласка, заповніть форму нижче, щоб завершити покупку, «створіть або завантажте відповідну веб-сторінку, відобразите текст і веб-сторінку в браузері користувача і зареєструйте інформацію. На рис. 1.1 показаний потік даних в системі діалогового агента.

Три основних компоненти в обробці кожного висловлювання показані на малюнку 1.2. Перший компонент цієї архітектури, інтерпретатор, виконує чотири типи аналізу виразу користувача: синтаксичний, дискурсивний, семантичний і прагматичний. Синтаксичний аналіз полягає у визначенні

граматичні відносини між словами в заяві користувача.Наприклад, у реченні (2) я хотів би, щоб швидкий синтаксичний аналіз комп'ютера приводив до синтаксичному аналізу пропозиції, що показує, що «хотів» - це головний дієслово, «я» є предметом, а «швидкий комп'ютер» - це об'єкт. Хоча багато діалогові агенти (в тому числі самі ранні) покладаються на зіставлення шаблонів без будь-якого синтаксичного аналізу [Weizenbaum, 1966], цей підхід не може масштабироваться. Оскільки кількість виразів, які агент повинен розрізняти між зростанням, кількість шаблонів, необхідних для розрізнення між операторами, швидко зростає за кількістю і складності. Аналіз дискурсу полягає у визначенні відносин між кількома реченнями. Важливою складовою аналізу дискурсу є еталонне дозвіл, завдання визначення об'єкта, що позначається посилальним виразом, наприклад «воно», «я б хотів його купити зараз». Пов'язана з цим проблема - це інтерпретація трьох крапок, тобто, матеріал опущений з інструкції, але неявний в діалоговому контексті. Наприклад, «Бездротовий зв'язок» означає «Я хотів би використовувати бездротову мережу» у відповідь на запитання: «Чи цікавитеся ви стандартної або бездротовою мережею?», Але один і той же вислів означає «Я хочу бездротової PDA» у відповідь на запитання: « який КПК вам потрібен? » Семантичний аналіз полягає у визначенні значення пропозиції.Як правило, це складається з подання заяви в канонічному формалізмі, який відображає затвердження з аналогічним значенням в єдине уявлення і полегшує висновки, які можна зробити з вистави.

РОЗДІЛ 3.

ВИКОРИСТАННЯ НЕЙРОННОЇ ДІАЛОГОВОЇ МОДЕЛІ ПРИ ПРОЕКТУВАННІ ДІАЛОГОВОЇ СИСТЕМИ

3.1. Розробка діалогового агента за допомогою рекурентних нейронних мереж

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

• Python 2.7.12

• Tensorflow 0.12.0

• Cuda Toolkit 8.0

• cuDNN v5.1

3.2. Ubuntu

Ubuntu - операційна система, заснована на Debian GNU / Linux. Основним розробником і спонсором є компанія Canonical. В даний час проект активно розвивається і підтримується вільним спільнотою.

Установка дистрибутива Linux Ubuntu:

1. Для початку необхідно завантажити образ iso з останньою версією Ubuntu на на офіційному сайті Ubuntu.com;

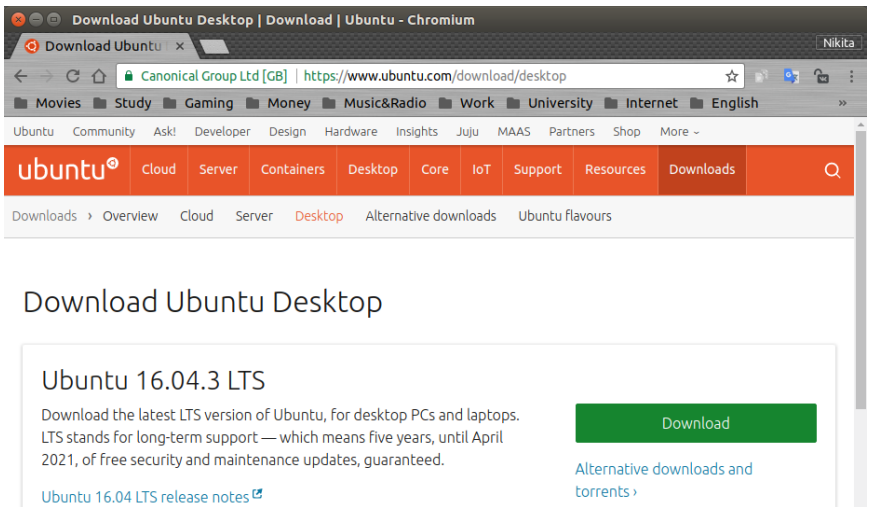

Рис. 3.1. Офіційний сайт Ubuntu містить дистрибутив

2. Далі необхідно створити інсталяційний накопичувач, для цієї мети досить використовувати безкоштовну програму WinSetupFromUSB. Завантажити її можна тут http://www.winsetupfromusb.com/downloads/ .

Після завантаження необхідно запустити програму (приклад наведено для останньої версії 1.0, що вийшла 17 жовтня 2013 і доступною по вищезазначеним посиланням) і виконати наступні нескладні дії:

- Вибрати необхідний USB накопичувач (врахуйте, що всі інші дані з нього будуть вилучені).
- Відзначити пункт Auto format it with FBinst.
- Відзначити пункт Linux ISO / Other Grub4dos compatible ISO і вказати шлях до образу диска Ubuntu.
- З'явиться діалогове вікно з питанням про те, як назвати цей пункт в меню завантаження. Напи сати що-небудь, допустимо, Ubuntu 13.04.
- Натиснути кнопку «Go», підтвердити, що обізнані про те, що всі дані з USB накопичувача будуть видалені і дочекайтеся завершення процесу створення завантажувальної флешки.

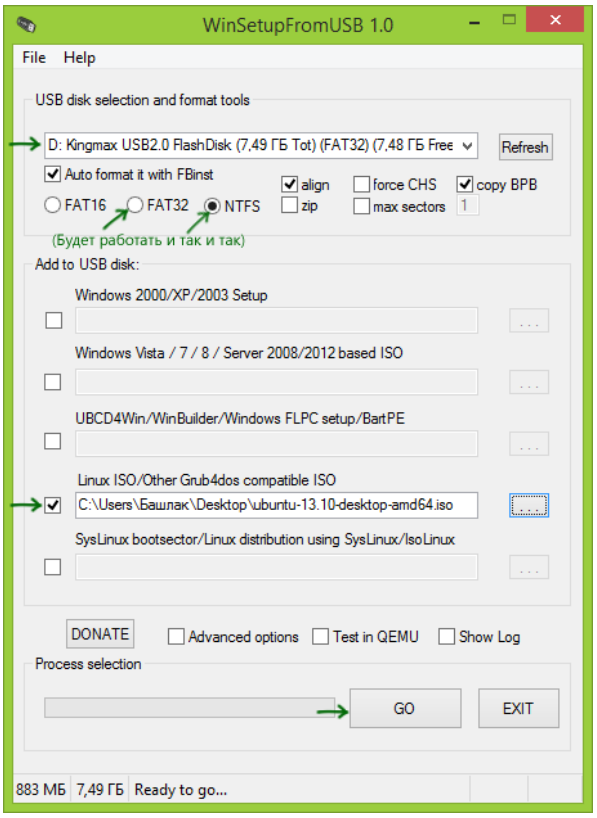

Рис. 3.2. Інтерфейс програми створення інсталяційного накопичувача

3. Наступний крок - зайти в BIOS комп'ютера і встановити там завантаження саме з щойно створеного дистрибутива.

Відразу після завантаження з флешки, ви побачите пропозицію вибрати мову і:

- Запустити Ubuntu без установки на комп'ютер;
- Встановити Ubuntu.

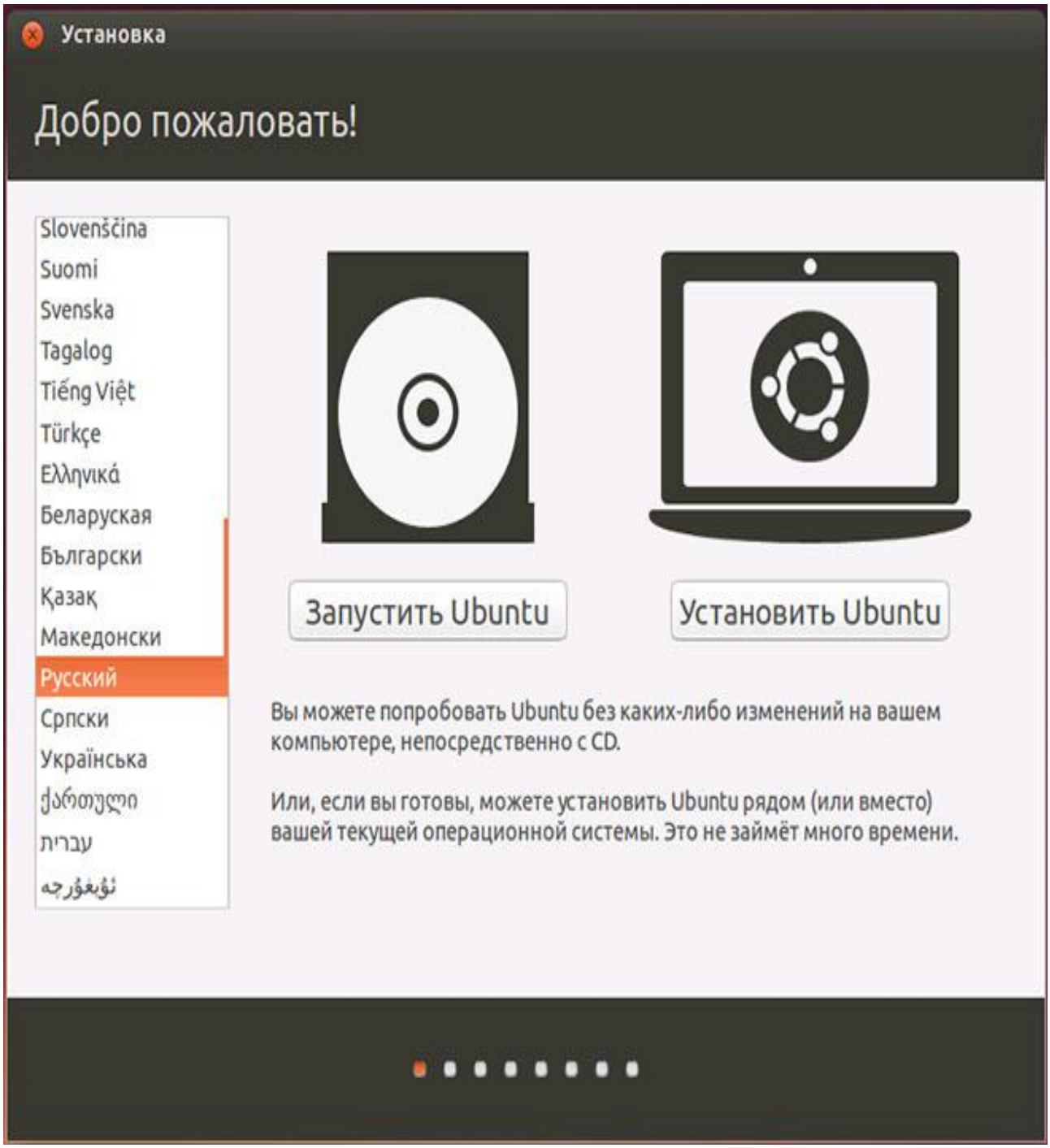

Рис. 3.3. Вікно «Ласкаво просимо»

Вибираємо «Встановити Ubuntu».

Вибираємо другий варіант, не забуваючи попередньо вибрати російську мову (або будь-якої іншої, якщо він для вас зручніше).

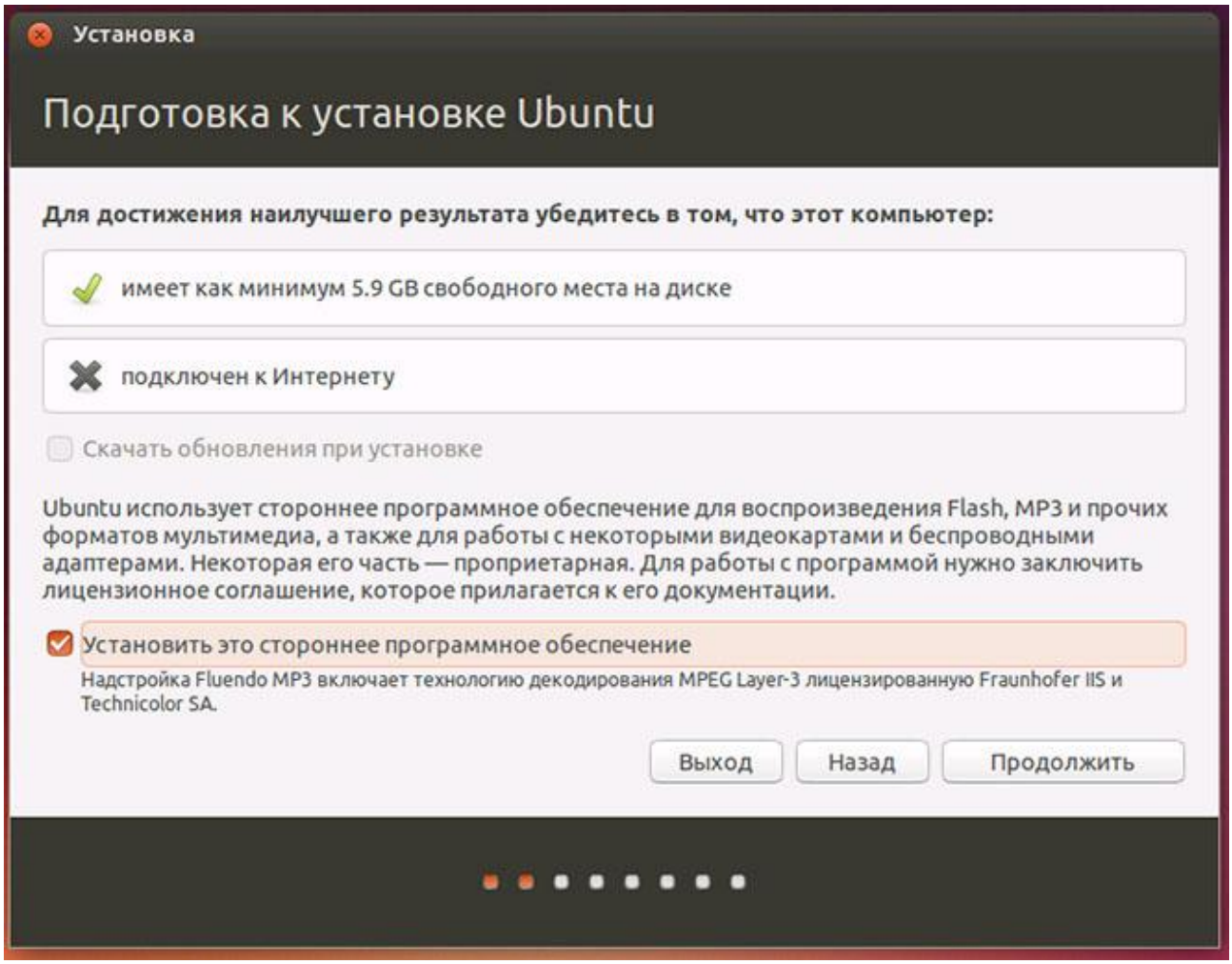

Рис. 3.4. Вікно «Підготовка до встановлення Ubuntu»

Наступне вікно буде називатися «Підготовка до встановлення Ubuntu». У ньому вам буде запропоновано переконатися, що комп'ютер має досить вільного місця на жорсткому диску і, крім цього, підключений до Інтернету. У багатьох випадках, якщо ви не використовуєте вдома Wi-Fi роутер і користуєтеся послугами провайдера з підключенням L2TP, PPTP або PPPoE, інтернет на цьому етапі буде відключений. Нічого жахливого. Він потрібен для того, щоб встановити всі оновлення і доповнення Ubuntu з Інтернету вже на початковому етапі. Але це можна зробити і пізніше. Також внизу ви побачите пункт «Встановити це стороннє програмне забезпечення». Він має відношення

до кодекам для відтворення MP3 і його краще відзначити. Причина, по якій цей пункт винесено окремо, в тому, що ліцензія даного кодека не цілком «Вільна», а в Ubuntu використовується тільки вільне програмне забезпечення.

4. На наступному кроці потрібно вибрати варіант установки Ubuntu:

- Поруч з Windows (в цьому випадку, при включенні комп'ютера буде показуватися меню, в якому можна буде вибрати, у чому ви збираєтеся працювати - Windows або Linux).
- Замінити наявну ОС на Ubuntu.
- Інший варіант (є самостійною розмітку жорсткого диска, для досвідчених користувачів).

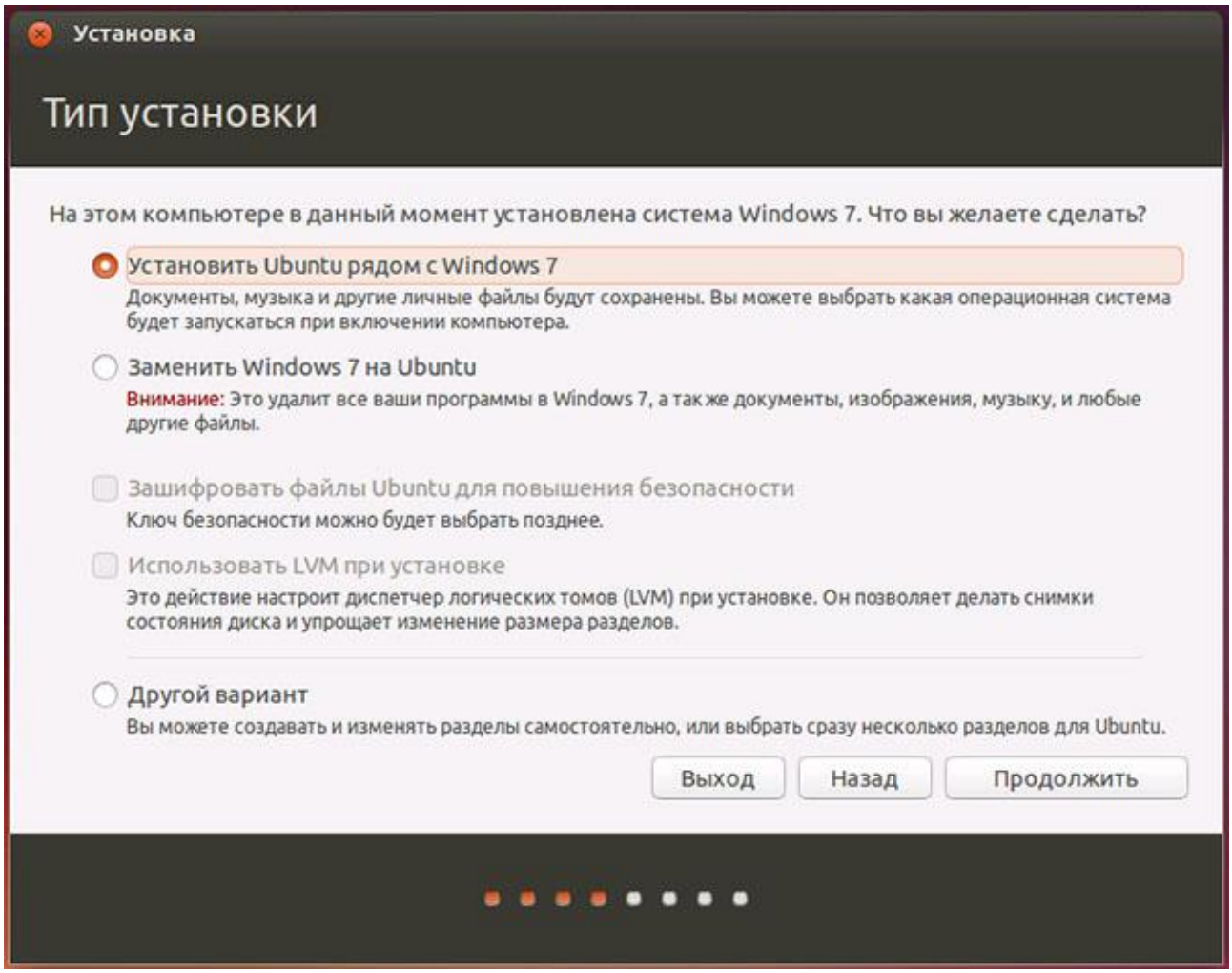

Рис. 3.5. Вікно «Установка»

Для цілей цієї інструкції я вибираю найбільш часто використовуваний варіант - установка другої операційної системи Ubuntu, залишаючи Windows 7.

У наступному вікні будуть відображені розділи вашого жорсткого диска. Пересуваючи роздільник між ними, ви можете вказати, скільки місця ви виділяєте для розділу з Ubuntu. Також є можливість самостійно виконати розбивку диска за допомогою розширеного редактора розділів. Однак, якщо ви користувач, не рекомендую звертатися до нього (сказав парі друзів, що там нічого складного, вони в підсумку без Windows залишилися, хоча мета інша була).

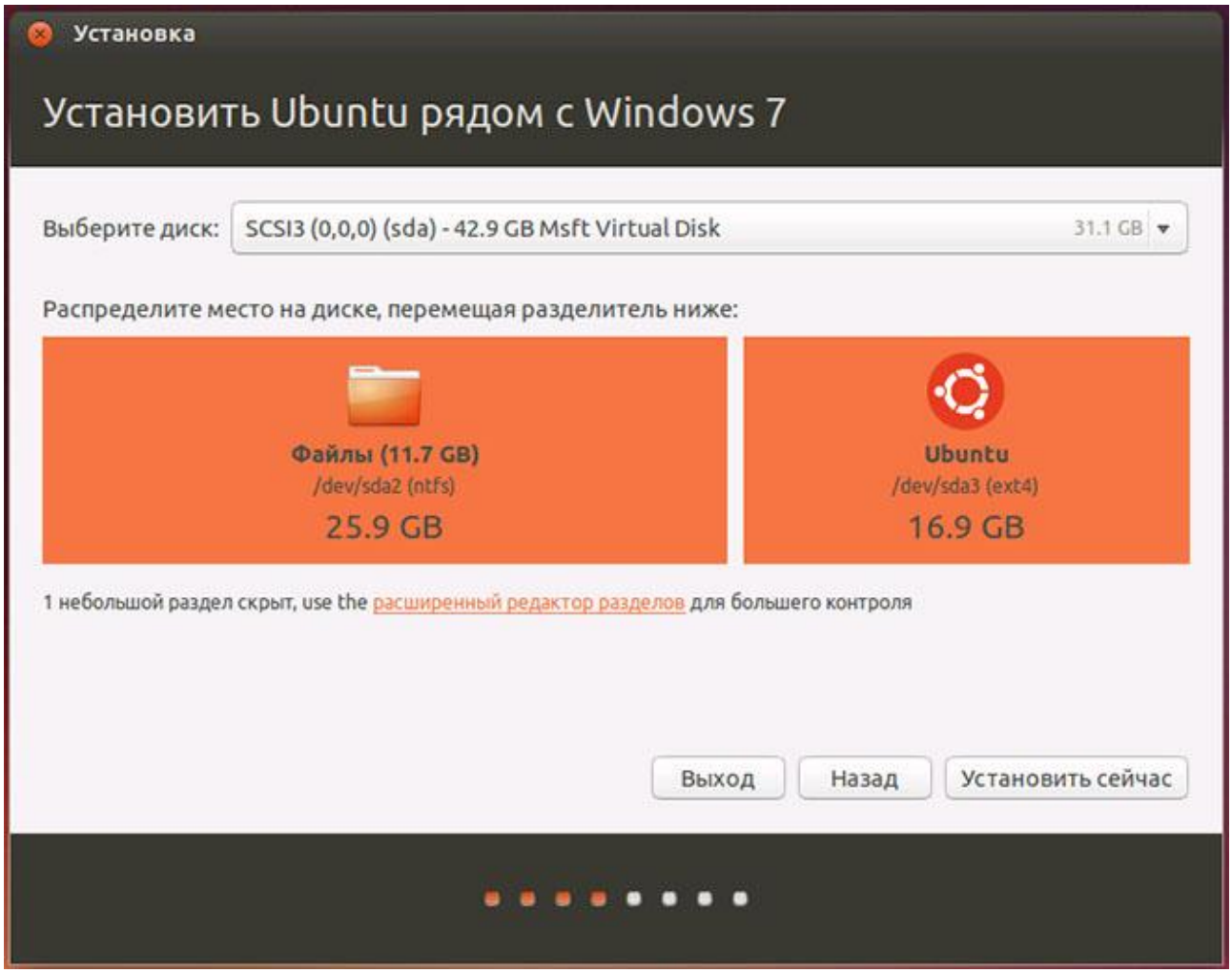

Рис. 3.6. Вікно «Установка»

Коли ви натиснете «Встановити зараз», вам буде продемонстровано попередження про те, що зараз будуть створені нові розділи дисків, а також змінені розміри старих і це може зайняти тривалий час (Залежить від ступеня зайнятості диска, а також його фрагментації). Натисніть «Продовжити».

Через деякий (різний, для різних комп'ютерів, але зазвичай не довго) вам буде запропоновано вибрати регіональні стандарти для Ubuntu - часовий пояс і розкладку клавіатури.

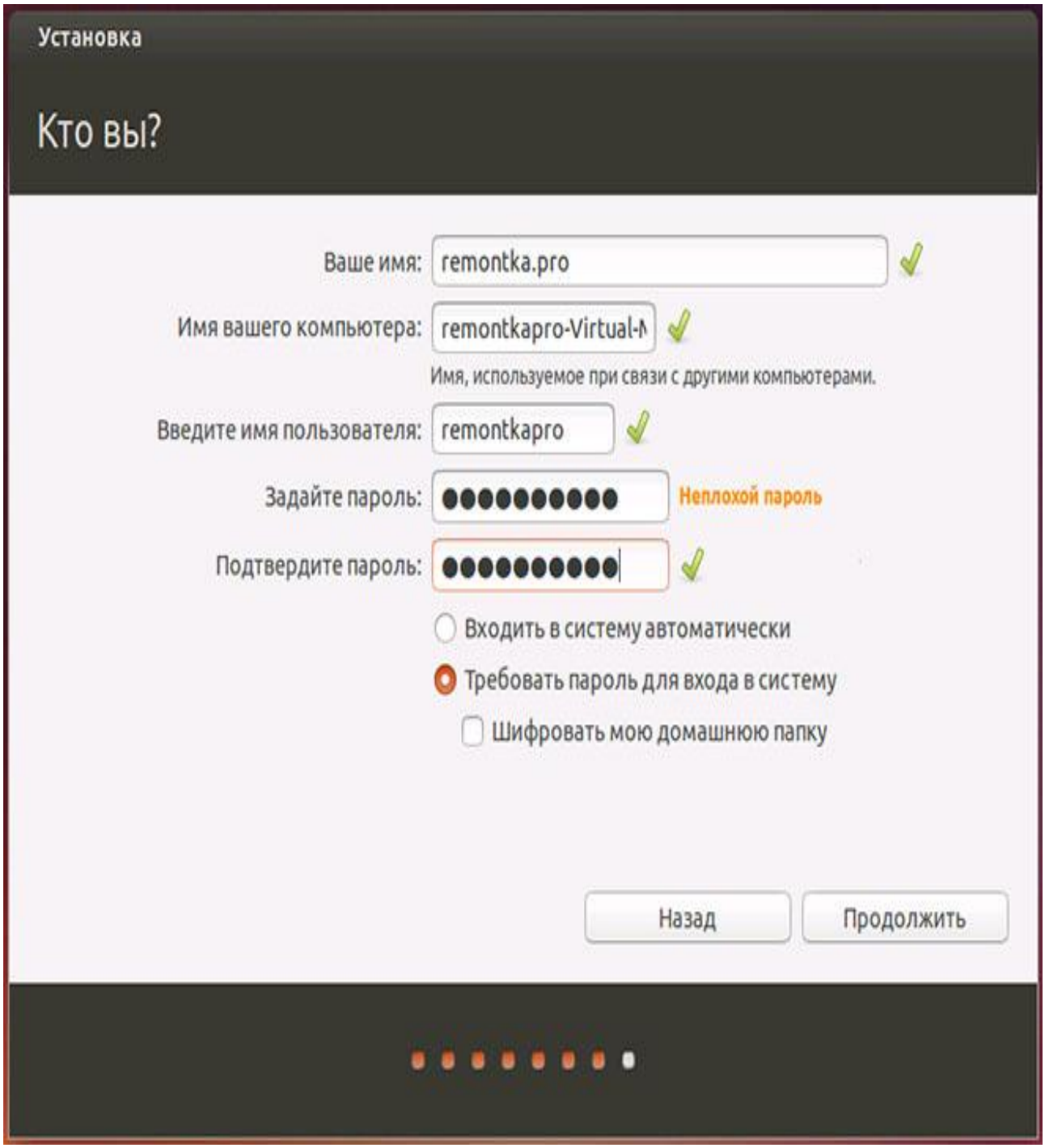

Рис. 3.7. Вікно «Установка»

5. Наступний етап - створення користувача і пароля Ubuntu. Тут нічого складного. Після заповнення, натискаємо «Продовжити» і починається сама установка Ubuntu на комп'ютер. Незабаром ви побачите

повідомлення про те, що установка завершена і пропозиція перезавантажити комп'ютер.

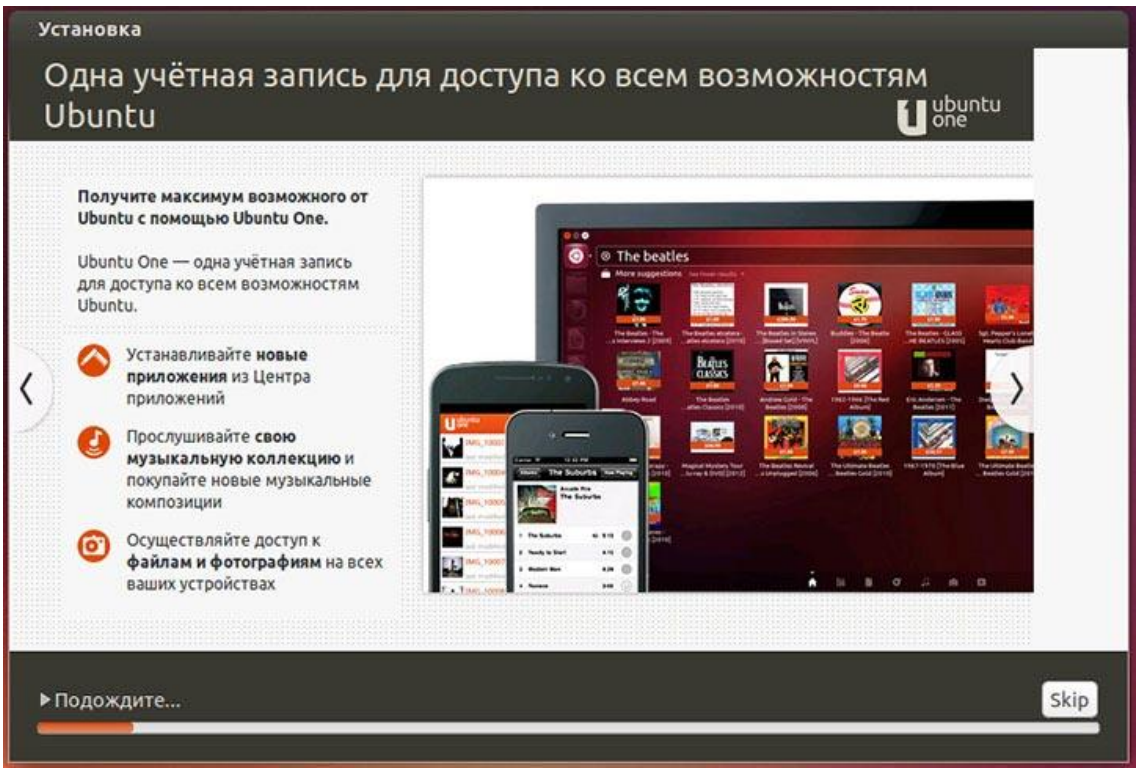

Рис. 3.8. Вікно «Установка»

6. На цьому все. Тепер, після того, як комп'ютер був перезавантажений, ви побачите меню вибору завантаження Ubuntu (в різних варіантах) або Windows, а потім, після введення пароля користувача - сам інтерфейс операційної системи.

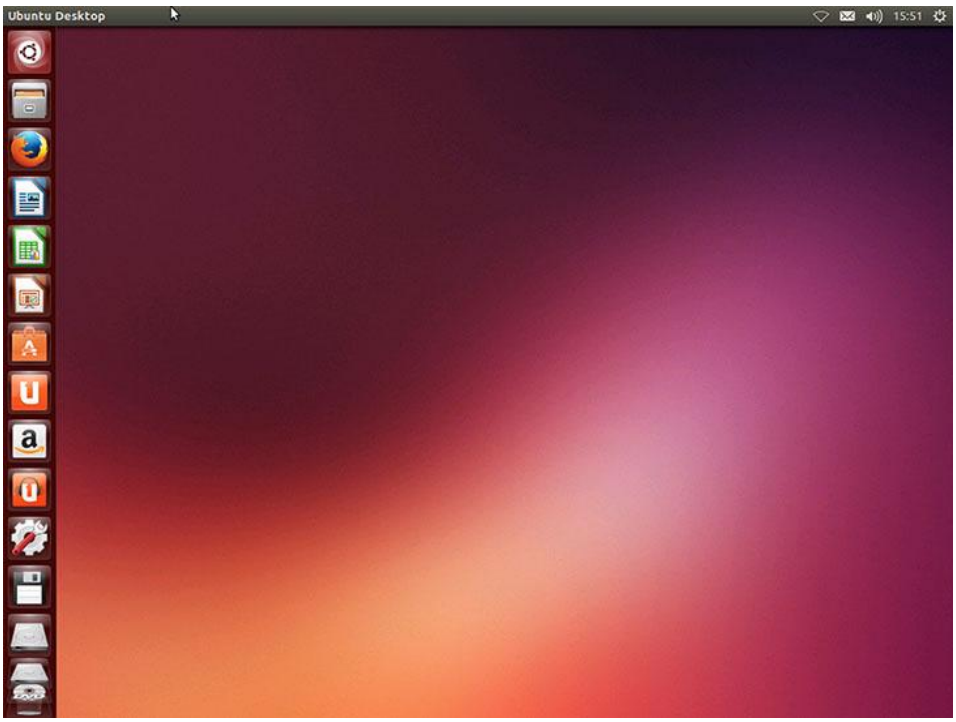

Рис. 3.9. Інтерфейс ОС Ubuntu

3.3. Python

Python - це багатофункціональний мова програмування для розробки різних програмних проектів. Python вийшов у світ в 1991 і названий на честь британської комік-групи Monty Python: так розробники хотіли підкреслити, що ця мова програмування настільки простий у використанні, що це навіть смішно. Простота установки, щодо зрозумілий синтаксис, негайне повідомлення про помилки - завдяки таким своїм якостям Python є відмінним рішенням як для новачків, так і для досвідчених розробників. Python 3 - остання версія мови - вже вважається майбутнім Python. Вимоги:

- Сервер Ubuntu 16.04 або Debian 8 (або будь-яка інша версія Debian Linux).
- Користувач з доступом до sudo.

3.3.1. Інсталяція Python 3

У системах Ubuntu 16.04 і Debian 8 Python 3 і Python 2 встановлені за замовчуванням. Щоб оновити пакети системи, необхідно ввести:

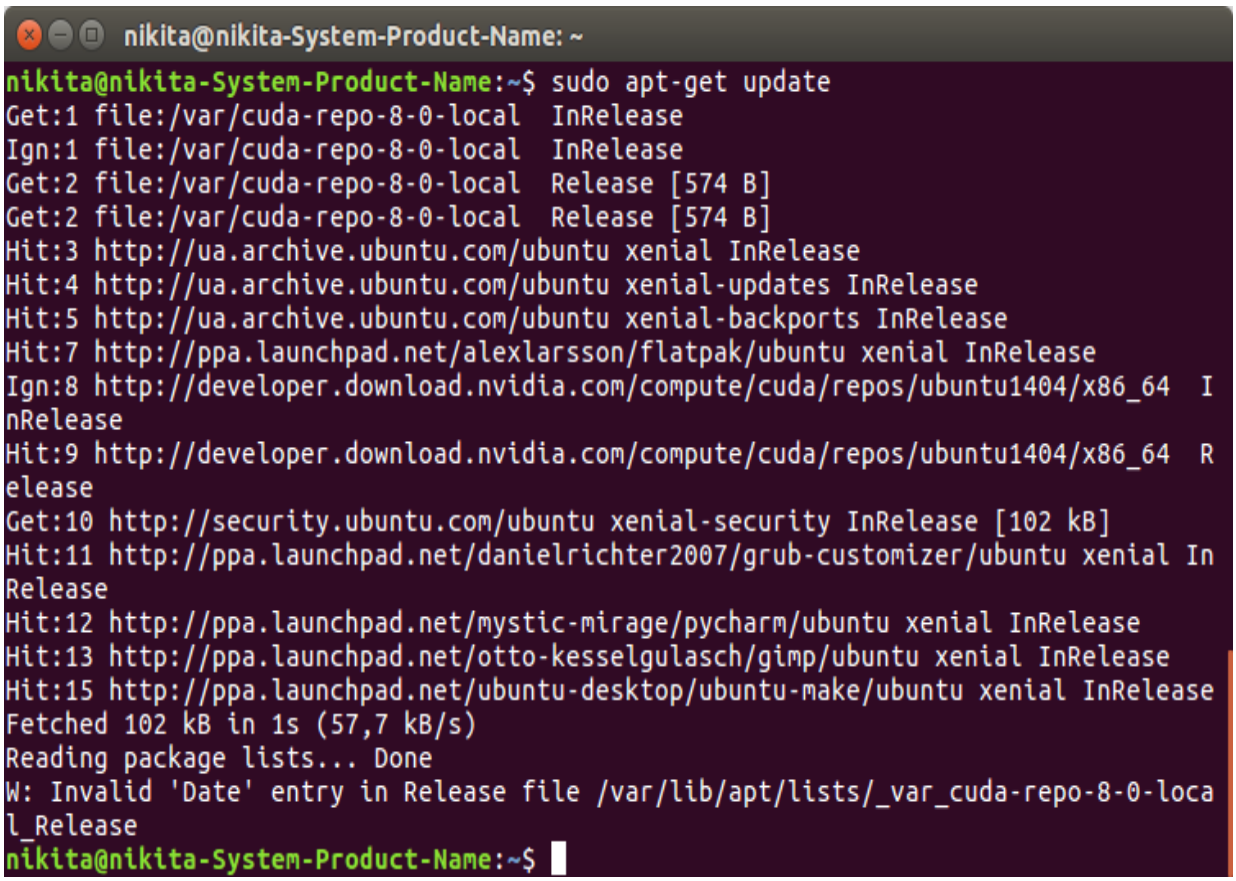

Рис. 3.10. Оновлення індексів пакетів

Щоб дізнатися, яка версія Python 3 встановлена в системі, необхідно ввести:

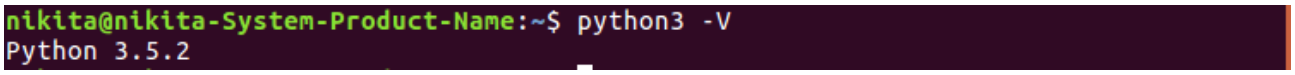

Рис. 3.11. Перевірка установки Python

У терміналі можна буде побачити номер поточної версії. Далі необхідно встановити пакетний менеджер Python, pip:

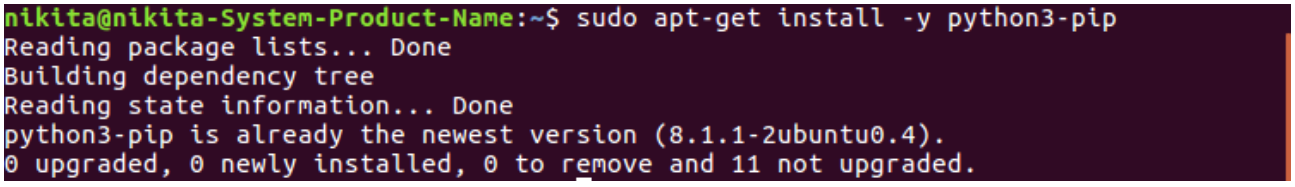

Рис. 3.12. Інсталяція пакетного менеджера PIP

Менеджер pip дозволяє встановлювати і управляти пакетами Python. Для установки пакета використовується такий синтаксис: sudo pip3 install

package_name. Замість package_name необхідно вказати ім'я пакету або бібліотеки.Наприклад, щоб встановити бібліотеку NumPy, можна ввести:

```
nikita@nikita-System-Product-Name:~$ sudo pip3 install numpy
The directory '/home/nikita/.cache/pip/http' or its parent directory is not owned
by the current user and the cache has been disabled. Please check the permissions
and owner of that directory. If executing pip with sudo, you may want sudo's -H fl
ag.
The directory '/home/nikita/.cache/pip' or its parent directory is not owned by th
e current user and caching wheels has been disabled. check the permissions and own
er of that directory. If executing pip with sudo, you may want sudo's -H flag.
Requirement already satisfied (use --upgrade to upgrade): numpy in ./.local/lib/py
thon3.5/site-packages
```

Рис. 3.13. Інсталяція пакета Numpy

3.4. Tensorflow

TensorFlow це програмне забезпечення машинного навчання з відкритим вихідним кодом, побудоване Google для навчання нейронних мереж. Нейронні мережі TensorFlow виражені у вигляді зберігають стан графів потоків даних. Кожен вузол в графі представляє операції, що виконуються за допомогою нейронних мереж на багатовимірних масивах. Ці багатовимірні масиви зазвичай відомі як «тензори», звідси і назва TensorFlow.

TensorFlow є глибоким вивченням системного програмного забезпечення. TensorFlow добре працює для пошуку інформації, як показаний в Google в тому, як побудований рейтинг пошук в їх машинному навчанні системи штучного інтелекту, RankBrain. TensorFlow може виконувати розпізнавання образів, як показано в Google, Inception, а також звукового розпізнавання людської мови.Це також корисно в рішенні інших проблем, не характерних для машинного навчання, таких як диференціальні рівняння в приватних.

Архітектура TensorFlow дозволяє розгортатися на декількох процесорах або графічних процесорів в межах робочого столу, сервера або мобільного пристрою. Є також розширення для інтеграції з CUDA, паралельні обчислення платформою від Nvidia. Це дає користувачам, які розгортають прямий доступ до GPU до безлічі віртуальних команд і інших елементів графічного процесора, які необхідні для паралельних обчислювальних задач.

Для нормальної роботи бібліотеки досить встановити версію CPU support only, але якщо потрібно прискорити навчання, то можна встановити версію

GPU, але ця версія вимагає наявності відеокарти від Nvidia.В даному випадку ми розглянемо обидва варіанти.

TensorFlow можна встановити декількома способами. Кожен метод має різні варіанти використання і середовище розробки:

- Python і Virtualenv: при такому підході, можна встановити TensorFlow і всі пакети, необхідні для використання TensorFlow у віртуальному середовищі Python;

- Native pip: глобальна установка TensorFlow. Цей метод рекомендується користувачам, які хочуть отримати загальнодоступну установку TensorFlow в багато користувачів;

В даному випадку достатньо встановити TensorFlow за допомогою пакетного менеджера pip.Після завершення установки, необхідно перевірити установку, запустивши коротку програму TensorFlow.

Перед тим, як почати установку необхідно встановити наступне компоненти:

- Один сервера Ubuntu 16.04, принаймні, 1 Гб оперативної пам'яті, слідуючи керівництву по початковому налаштуванні сервера Ubuntu 16.04, включаючи некорневого sudo користувача і брандмауер.Вам знадобиться принаймні, 1 Гб оперативної пам'яті, щоб успішно виконати останній приклад в цьому підручнику.

- Python 3.3 або вище і встановлений virtualenv. Дотримуйтесь гіду як встановити Python 3 на Ubuntu 16.04 для настройки Python і virtualenv.

- Встановлений Git, виконаний по керівництву: як встановити Git на Ubuntu 16.04. Ви будете використовувати його, щоб завантажити сховище прикладів.

Інсталяція TensorFlow:

1. Необхідно виконати наступну команду для встановлення та оновлення до необхідної версії TensorFlow:

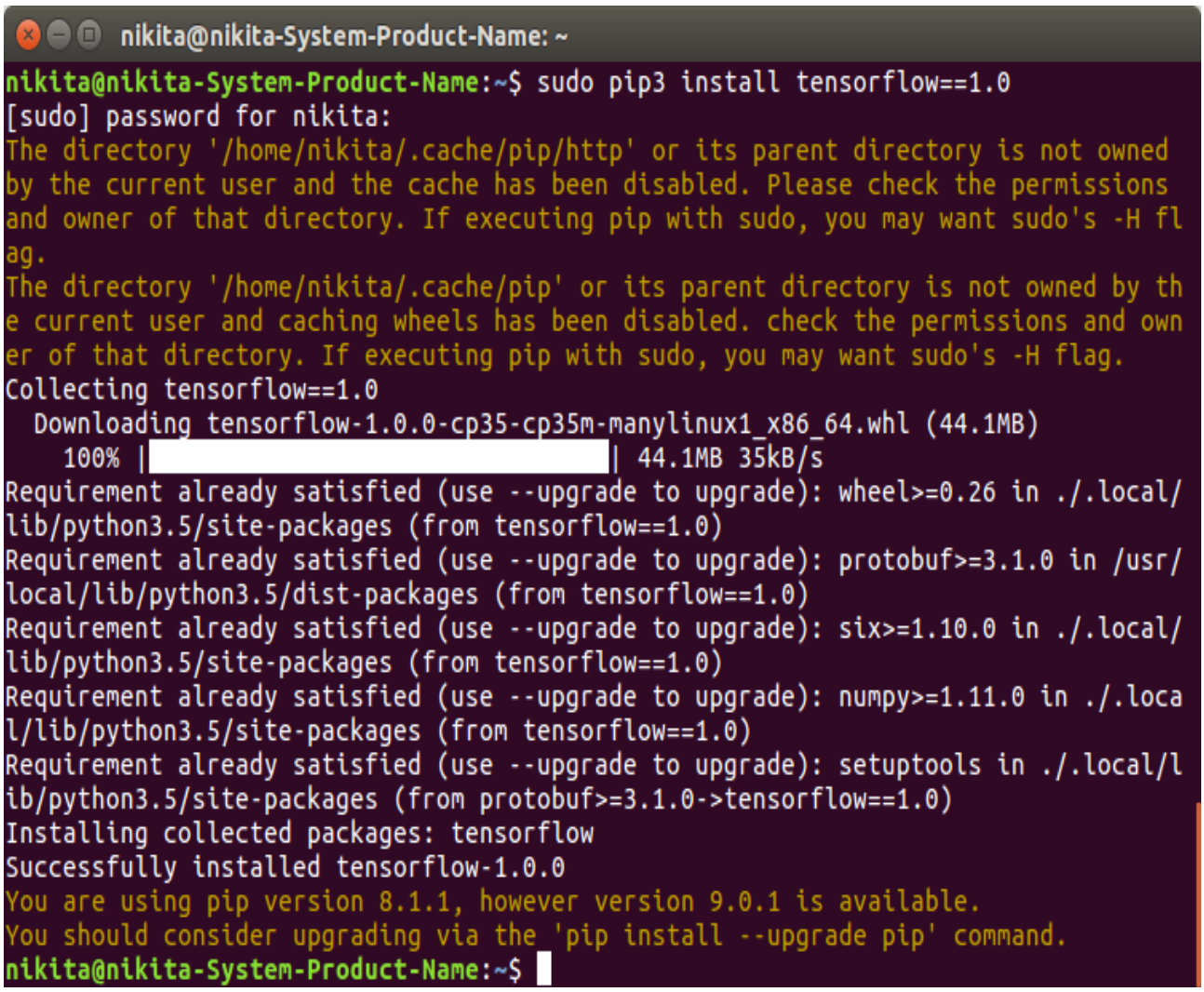

Рис. 3.14. Вікно інсталяції TensorFlow-CPU

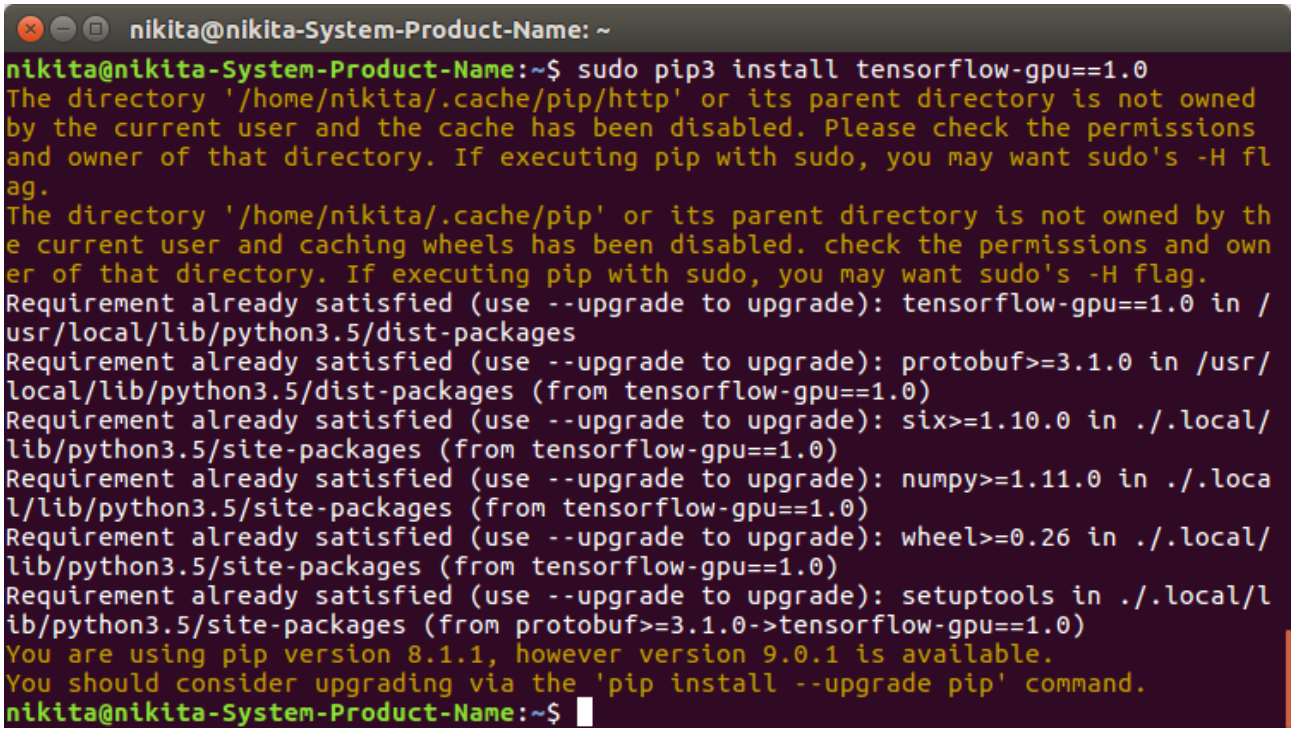

Рис. 3.15. Вікно інсталяції TensorFlow-GPU

Тепер, коли TensorFlow встановлений, необхідно упевниться, що TensorFlow працює.

2. Перевірка інсталяції.

Щоб перевірити інсталяцію TensorFlow, необхідно запустити просту програму в TensorFlow. Для цього можна використовувати приклад «Привіт, світ!». Замість того, щоб створювати файл Python, досить створити програму, використовуючи інтерактивну консоль Python.

Для того, щоб написати програму, необхідно запустити інтерпретатор Python:

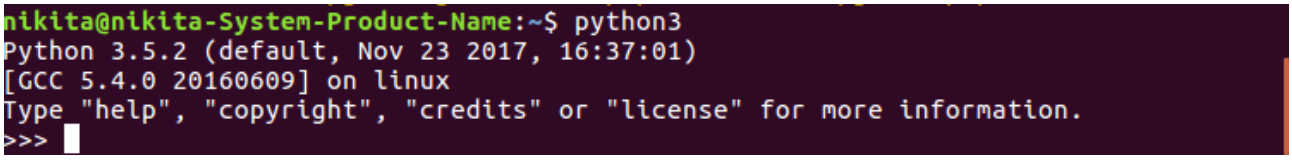

Рис. 3.16. Запуск інтерпретатора Python

Після запуску інтерпретатора на можна побачити наступне повідомлення в терміналі ">>>"

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:

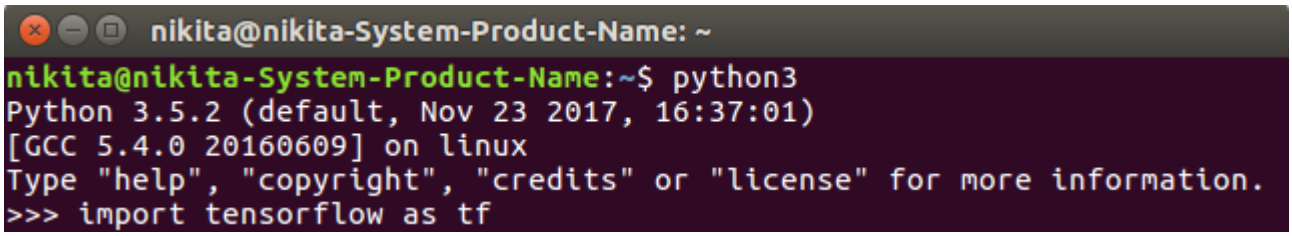

Рис. 3.17. Імпорт пакета TensorFlow

Потім необхідно додати наступний рядок коду, щоб вивести повідомлення «Привіт, світ!»:

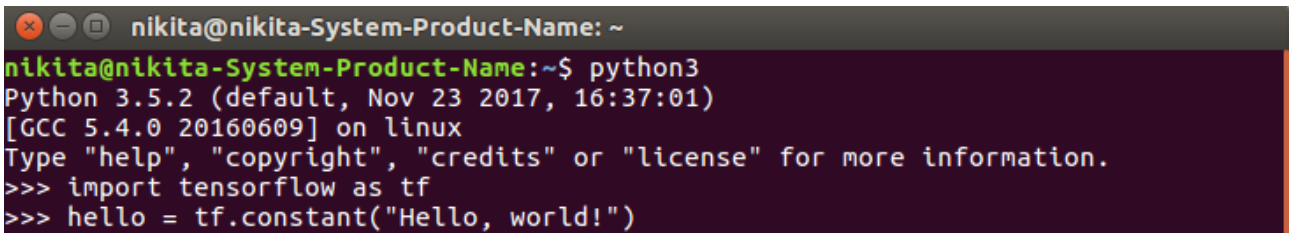

Рис. 3.18. Привласнення змінної hello

Потім необхідно створити новий сеанс TensorFlow і привласнити його змінної:

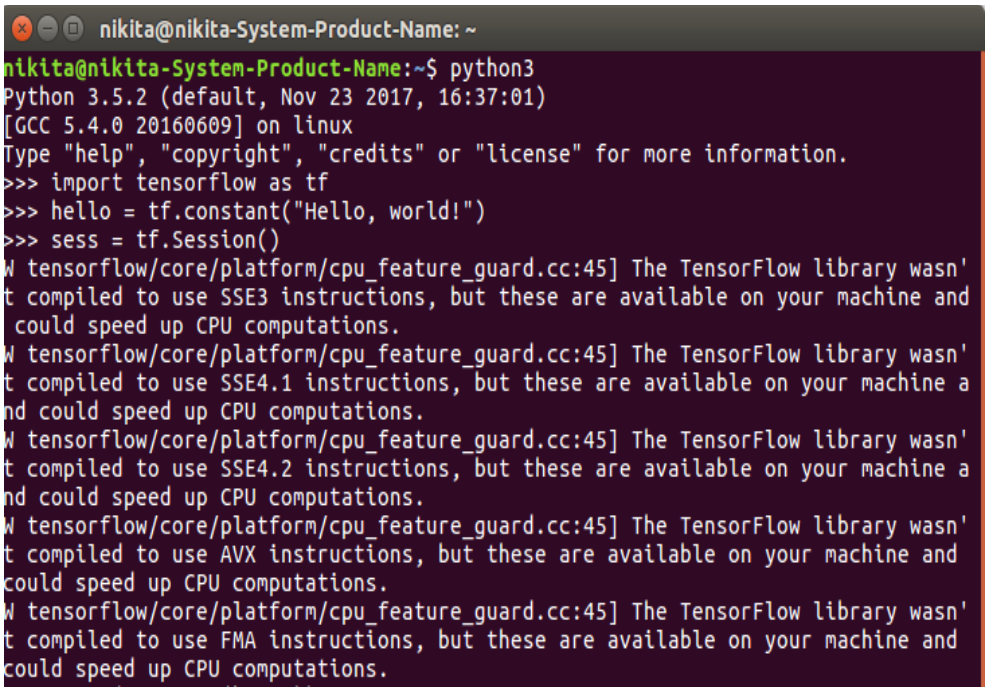

Рис. 3.19. Створення нового сеансу TenorFlow

Нарешті, щоб вивести результат запуску сеансу hello в TensorFlow, необхідно ввести такі рядки коду:

Рис. 3.20. Вивід повідомлення «Привіт, світ!»

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

3.5. Cuda

CUDA (англ. Compute Unified Device Architecture) - програмно-апаратна архітектура паралельних обчислень, яка дозволяє істотно збільшити обчислювальну продуктивність завдяки використанню графічних процесорів фірми Nvidia.

Навчання нейронної мережі можна значно прискорити використовуючи GPU. Tensorflow може навчати нейронні мережі на GPU компанії NVIDIA. Для цього потрібно встановити NVIDIA CUDA.

1. Інсталяція NVIDIA

Якщо у вас на комп'ютері була встановлена попередня версію CUDA або драйвери NVIDIA, то їх необхідно видалити. Встановлювати драйвер NVIDIA GPU найзручніше зі сховищ Proprietary GPU Drivers.

```
nikita@nikita-System-Product-Name:~$ sudo add-apt-repository ppa:graphics-driver
s/ppa
 Fresh drivers from upstream, currently shipping Nvidia.

## Current Status

Current official release: `nvidia-387` (387.34)
Current long-lived branch release: `nvidia-384` (384.98)

For G8x, G9x and GT2xx GPUs use `nvidia-340` (340.104)
For NV4x and G7x GPUs use `nvidia-304` (304.137)

Support timeframes for Unix legacy GPU releases:
https://nvidia.custhelp.com/app/answers/detail/a_id/3142
```

Рис. 3.21. Додання репозиторія

```
nikita@nikita-System-Product-Name: ~
nikita@nikita-System-Product-Name:~$ sudo apt-get install nvidia-375
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  nvidia-375
0 upgraded, 1 newly installed, 0 to remove and 17 not upgraded.
Need to get 5 270 B of archives.
After this operation, 17,4 kB of additional disk space will be used.
Get:1 http://ppa.launchpad.net/graphics-drivers/ppa/ubuntu xenial/main amd64 nvi
dia-375 amd64 384.98-0ubuntu0~gpu16.04.1 [5 270 B]
Fetched 5 270 B in 0s (44,6 kB/s)
Selecting previously unselected package nvidia-375.
(Reading database ... 231962 files and directories currently installed.)
Preparing to unpack .../nvidia-375_384.98-0ubuntu0~gpu16.04.1_amd64.deb ...
Unpacking nvidia-375 (384.98-0ubuntu0~gpu16.04.1) ...
Setting up nvidia-375 (384.98-0ubuntu0~gpu16.04.1) ...
nikita@nikita-System-Product-Name:~$
```

Рис. 3.22. Інсталяція драйверу NVIDIA

2. Інсталяція CUDA 8

У репозиторії Ubuntu зараз є тільки CUDA 7.5. Так як ми хочемо використовувати CUDA 8, то доведеться встановлювати вручну. Завантажте файл, що запускається (runfile) CUDA 8 з сайту NVIDIA.

Для інсталяції необхідно запустити файл. Зверніть увагу, що не потрібно встановлювати драйвери GPU, які входять до складу цього файлу.

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

CUDA 8 буде встановлена в каталог /usr/local/cuda-8.0. Додатково буде створено символічне посилання / usr / local / cuda. Рекомендується використовувати саме цю посилання, щоб в майбутньому можна було швидко змінити використовувану версію CUDA.

Рис. 3.23. Розташування NVIDIA CUDA

3. Налаштування параметрів оточення

Після інсталяції необхідно прописати шляхи до виконуваних файлів і бібліотек CUDA. Для цього необхідно створити файл /etc/profile.d/cuda.sh і записати туди наступне:

export PATH = $ PATH: / usr / local / cuda / bin
export CUDADIR = / usr / local / cuda
export GLPATH = / usr / lib

Для бібліотек створити файл /etc/ld.so.conf.d/cuda.conf з одним рядком:

/ Usr / local / cuda / lib64

Потім для налаштування бібліотек потрібно викликати команду:

sudo ldconfig

На цьому інсталяція завершена. Необхідно перезавантажити комп'ютер, щоб можна було використовувати CUDA.

3.6. PyCharm

PyCharm - це одна з кращих середовищ розробки для написання програм на мові Python. Програма поширюється як у професійній версії, яка коштує грошей, так і у версії для спільноти. Професійна версія має більше можливостей, в цій статті буде розглянута установка PyCharm Ubuntu 16.04, як професійної, так і безкоштовної версії.

Особливості PyCharm. Середовище розробки програмного забезпечення на мові Python має такі особливості:

- підсвічування синтаксису;
- автовідступ і форматування коду;
- автозавершення коду;
- підсвічування блокових і строкових коментарів;
- виділення помилок в реальному часі;
- згортання фрагментів коду;
- легка навігація по коду і пошук;
- аналіз коду;
- настроюються вставки коду;
- рефракторінг Python;
- документація.

Можна використовувати неофіційний PPA щоб встановити PyCharm Ubuntu 16.04. Цей репозиторій підтримує різні версії Ubuntu, в тому числі більш ранні, починаючи від Ubuntu 14.04. Для додавання репозиторію необхідно виконати:

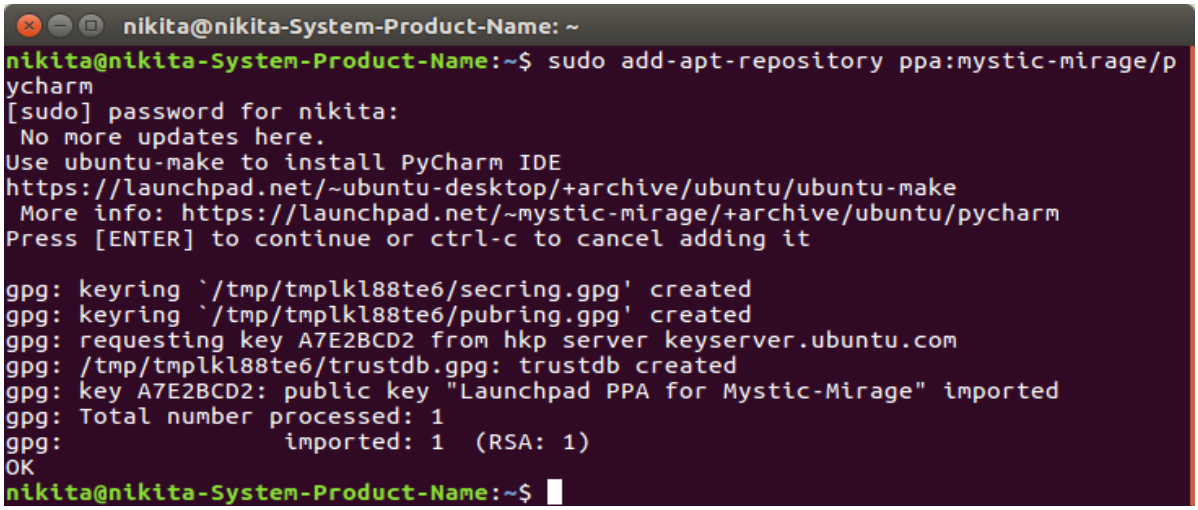

Рис. 3.24. Додавання репозитрію де знаходится PyCharm

Для встановлення безкоштовної версії використовуйте таку команду:

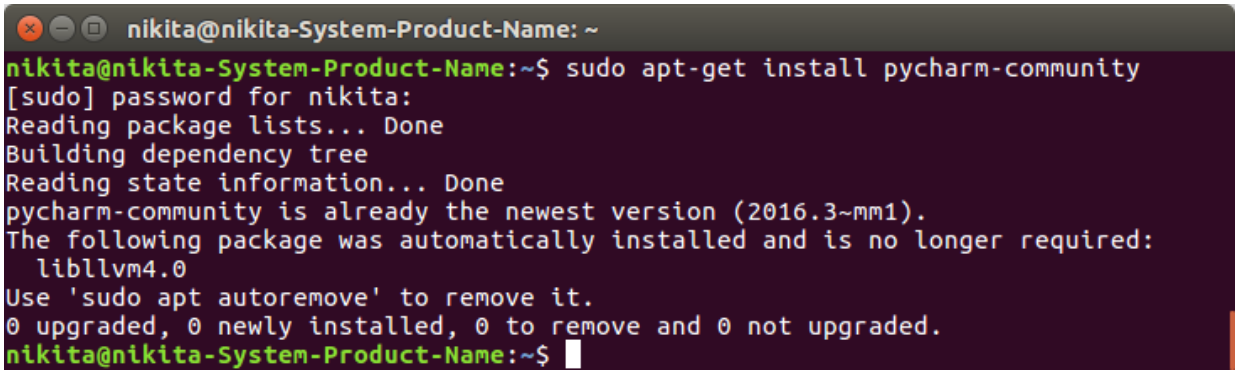

Рис. 3.25. Інсталяція PyCharm

Після її виконання буде завантажено близько 120 Мегабайт даних. Після інсталяції програма буде знаходоится в головному меню Unity. Ось так буде виглядати головне вікно:

Рис. 3.26. Головне вікно PyCharm

3.7. Розробка нейронної діалогові моделі

Для того щоб запустити навчання моделі треба прописати наступні параметри у скрипті main.py:

--corpus opensubs – означає вибір корпусу даних;

--modelTag OpenSubtitles – вибір моделі яку треба навчити/протестувати;

Щоби прописати параметри треба перейти у вкладку Run, потім Edit Configurations:

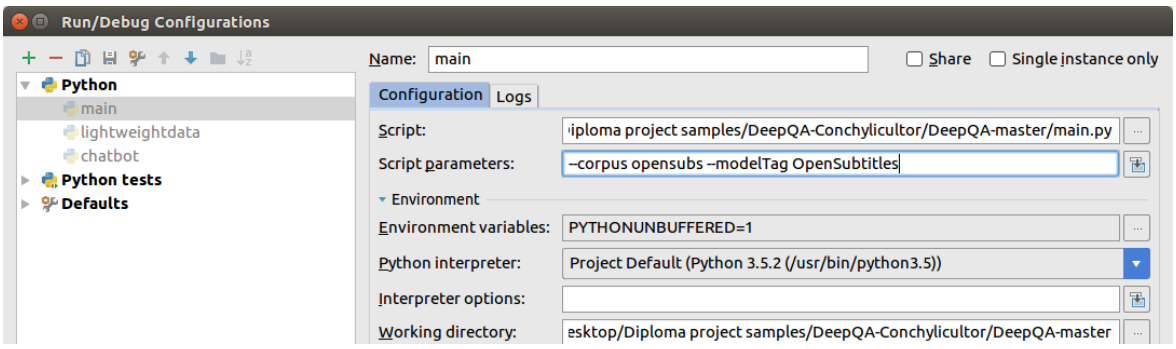

Рис. 3.27. Параметри запуску навчання скрипту main.py

Потім треба підтвердити зміни та запустити скрипт main.py наступньою комбінацією клавіш: Alt+Shift+F10 у середовищі PyCharm. Після цього пійде процес навчання моделі:

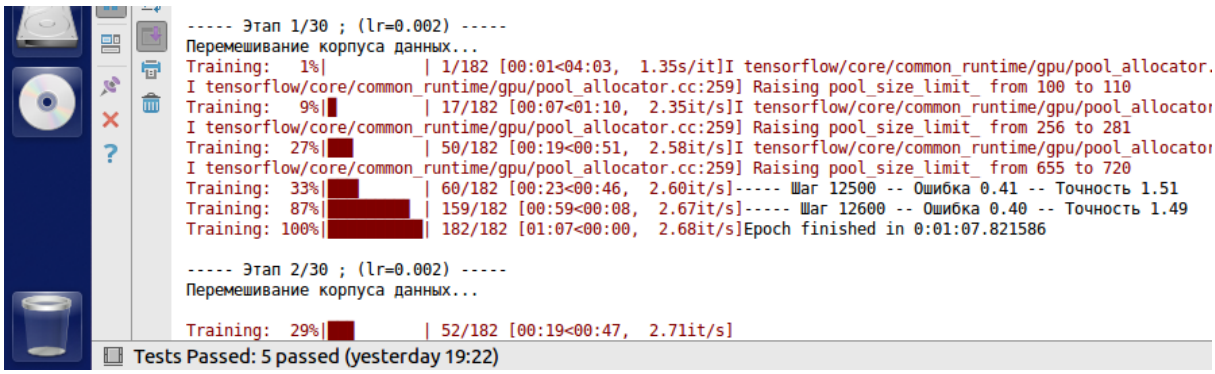

Рис. 3.28. Початок навчання моделі

Після того як модель почала вчитися треба почекати пару годин, щоб побачити результат.

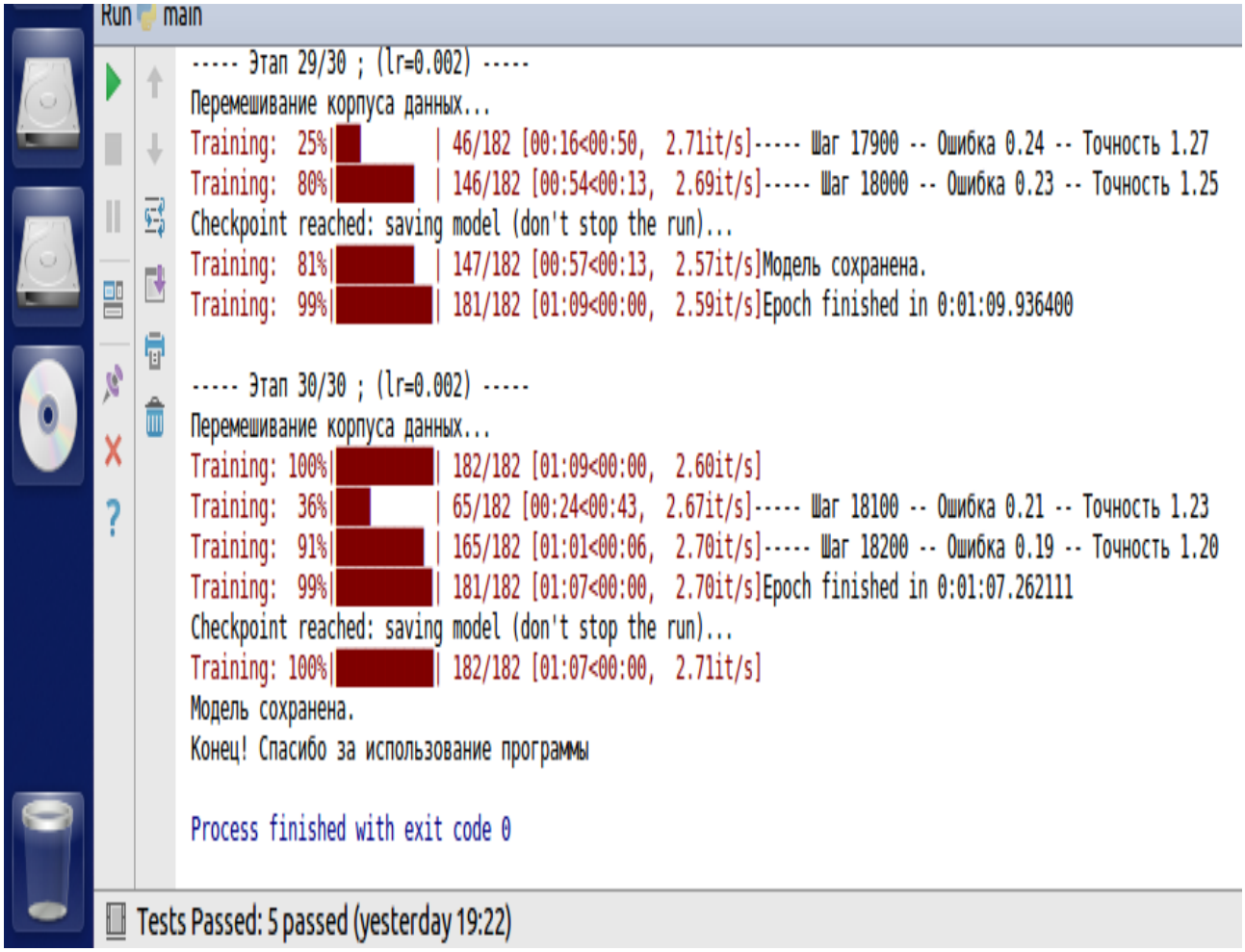

Рис. 3.29. Кінец навчання моделі.

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

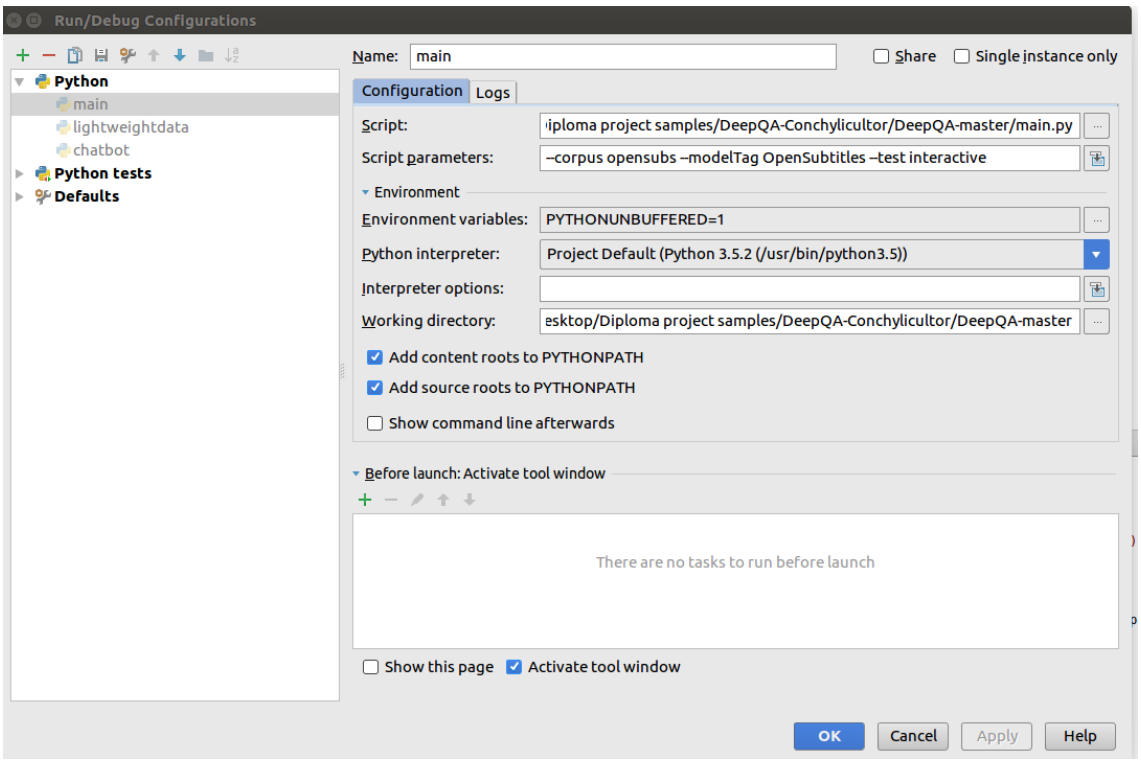

Рис. 3.30. Параметри скрипту main.py

Результати роботи програми:

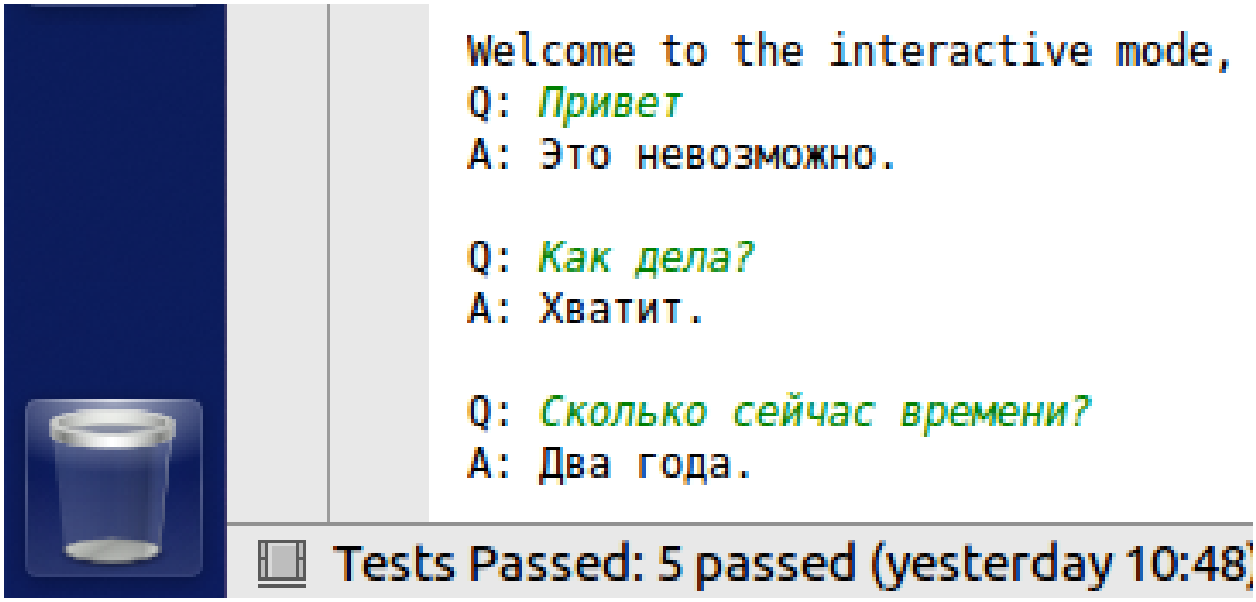

Рис. 3.31. Результат роботи програми

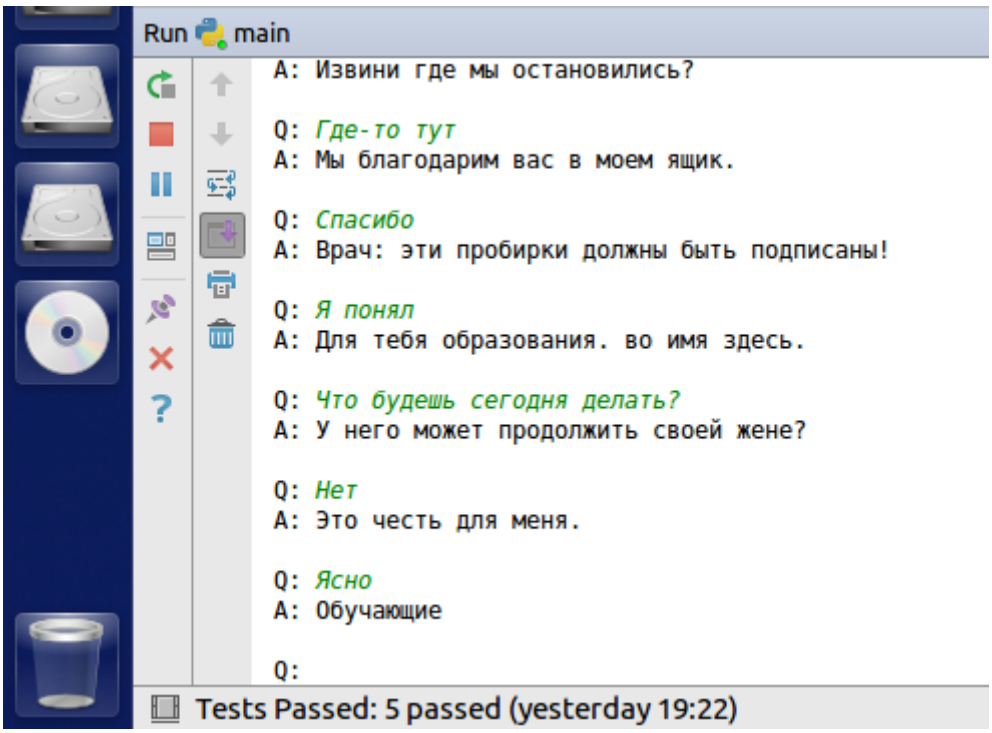

Рис. 3.32. Результат роботи програми

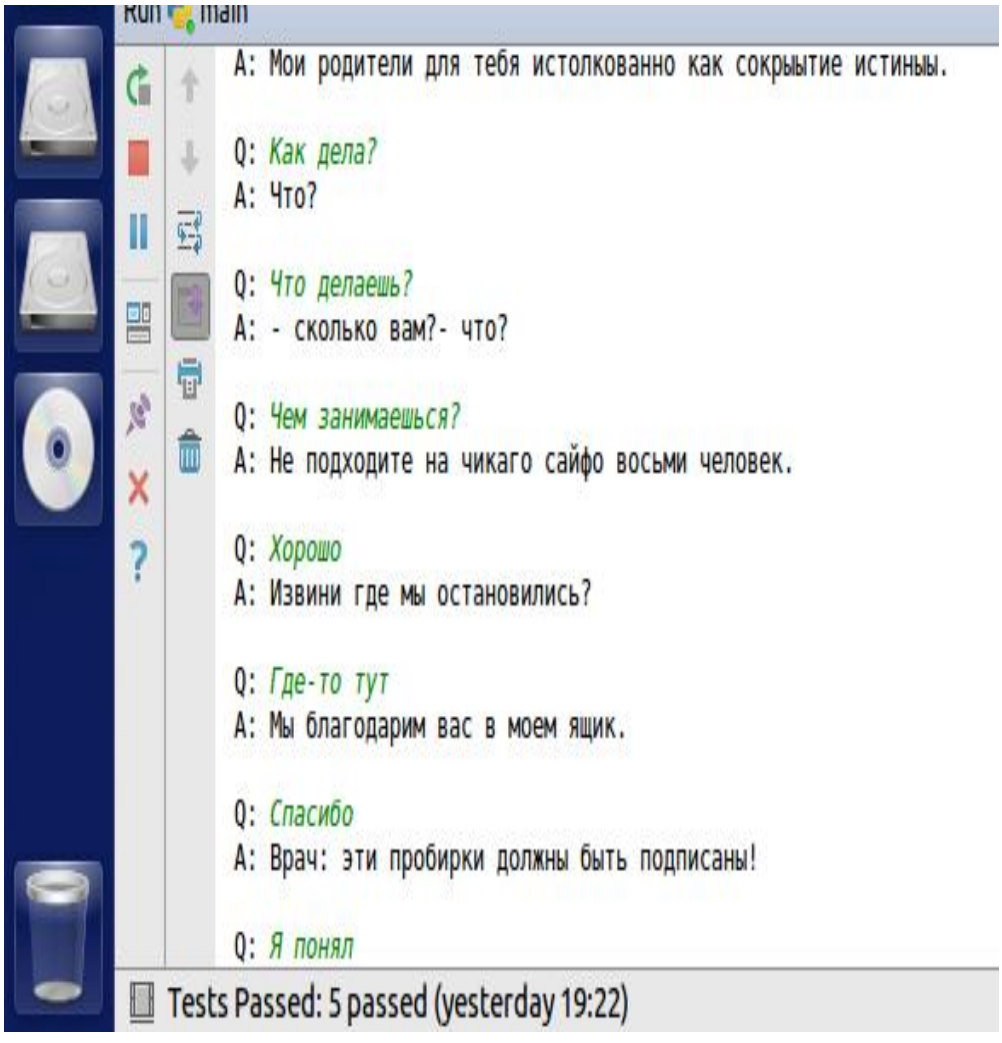

Рис. 3.33. Результат роботи програми

Висновки

Таким чином, в статті ми провели аналіз особливостей імітації мовної поведінки людини в процесі спілкування, визначили основні функції та принципи роботи чат-бота з метою розробки моделі спілкування і проілюстрували приклад написання чат-бота.

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

РОЗДІЛ 4.

ЕКОНОМІЧНА ЧАСТИНА

В дипломному проекті було розглянуто діалогову модель використовуючи рекурентну нейронну мережу для ведення діалогу зі штучним інтелектом . Як результат була розроблена, реалізована та відлагоджена діалогова модель яка може вести осмислену бесіду та самонавчатися.

1) Передбачуване число операторів – 940.

2) Коефіцієнт складності програми – 1,3.

3) Коефіцієнт корекції програми в ході її розробки – 0,1.

4) Коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,2.

5) Коефіцієнт кваліфікації програміста – 0,8.

6) Середня годинна заробітна плата програміста – 35 грн/год.

7) Вартість машино-годин ЕОМ – 10 грн/год.

4.1. Визначення трудомісткості розробки програмного забезпечення

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д, \text{людино-годин}$$

де t_o – витрати праці на підготовку й опис поставленої задачі (приймається 50);

t_u – витрати праці на дослідження алгоритму рішення задачі;

t_a – витрати праці на розробку блок-схеми алгоритму;

t_n – витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ – витрати праці на налагодження програми на ЕОМ;

$t_д$ – витрати праці на підготовку документації.

t = 50 + 34 + 96 + 96 + 513 + 228 = 1017 людино-годин

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів (підпрограм):

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

де q – передбачуване число операторів;

C – коефіцієнт складності програми (в інтервалі від 1.25 до 2.0);

p – коефіцієнт корекції програми в ході її розробки (в інтервалі від 0.05 до 1).

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

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75...85) \cdot k}, \text{людино-годин},$$

де B – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі (в інтервалі від 1.2 до 1.5);

k – коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності.

Таблиця 4.1

Значення коефіцієнта кваліфікації програміста

Стаж програміста	Значення коефіцієнта k
до 2-х років	0,8
від 2 до 3 років	1,0
від 3 до 5 років	1,1 … 1,2
від 5 до 10 років	1,2 … 1,3
понад 10 років	1,3 … 1,5

$$t_u = \frac{1344 \cdot 1,2}{80 \cdot 0,8} = 25 \text{ людино} - \text{годин}$$

Витрати праці на розробку алгоритму рішення задачі:

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

$$t_a = \frac{1344}{23 \cdot 0,8} = 73 \text{ людино} - \text{годин}$$

Витрати на складання програми по готовій блок-схемі:

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

$$t_n = \frac{1344}{22 \cdot 0{,}8} = 76 \text{ людино} - \text{годин}$$

Витрати праці на налагодження програми на ЕОМ:

– за умови автономного налагодження одного завдання:

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

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

– за умови комплексного налагодження завдання:

$$t^k_{отл} = 1{,}5 \cdot t_{отл}, \text{людино-годин}$$

$$t^{\text{к}}{}_{\text{отл}} = 1{,}5 * 420 = 630 \text{ людино} - \text{годин}$$

Витрати праці на підготовку документації:

$$t_д = t_{др} + t_{до}, \text{людино-годин}$$

$$t_д = 130 + 98 = 228 \text{ людино-годин},$$

де $t_{др}$ – трудомісткість підготовки матеріалів і рукопису.

$$t_{др} = \frac{Q}{(15...20)\cdot k}, \text{людино-годин}$$

$$t_{\text{др}} = \frac{1344}{16 \cdot 0{,}8} = 105 \text{ людино} - \text{годин}$$

$t_{до}$ – трудомісткість редагування, печатки й оформлення документації

$$t_{до} = 0{,}75 \cdot t_{др}, \text{людино-годин}$$

$$t_{\text{до}} = 0{,}75 \cdot 105 = 79 \text{ людино} - \text{годин}$$

4.2. Витрати на створення програмного забезпечення

Витрати на створення ПЗ $К_{по}$ включать витрати на заробітну плату

виконавця програми $З_{ЗП}$ і витрат машинного часу, необхідного на налагодження програми на ЕОМ.

$$К_{ПО} = З_{ЗП} + З_{МВ}, \text{грн}$$

$$\text{К}_{\text{по}} = 33120 + 5040 = 38160 \text{ грн.}$$

Заробітна плата виконавців визначається за формулою:

$$З_{ЗП} = t \cdot C_{пр}, \text{грн.}$$

де t – загальна трудомісткість, людино-годин;

$C_{пр}$ – середня годинна заробітна плата програміста, грн/годин

$$\text{З}_{\text{зп}} = 828 \cdot 40 = 33120 \text{ грн.}$$

Вартість машинного часу, необхідного для налагодження програми на ЕОМ:

$$З_{МВ} = t_{отл} \cdot C_{мч}, \text{грн.},$$

де $t_{отл}$ – трудомісткість налагодження програми на ЕОМ, год.

$C_{мч}$ – вартість машино-годин ЕОМ, грн/год.

$$\text{З}_{\text{мв}} = 630 \cdot 8 = 5040 \text{ грн.}$$

Визначені в такий спосіб витрати на створення програмного забезпечення є частиною одноразових капітальних витрат на створення АСУП.

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p}, \text{міс.},$$

де B_k – число виконавців;

F_p – місячний фонд робочого часу (при 40-годинному робочому тижні F_p=176 годин).

$$\text{Т} = \frac{828}{1 \cdot 176} = 4{,}7 \text{ мес.}$$

4.3. Маркетингові дослідження ринку збуту розробленого програмного продукту

Моделювання розмови важливе завдання в розумінні природної мови і машинного інтелекту. 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.

У дипломній роботі розглядається діалогова модель і рекурентна нейронна мережа яка може використовуватися для ведення осмисленого діалогу зі штучним інтелектом та самонавчатися.

Актуальність даної роботи диктується умовами постійного розвитку штучного інтелекту та машинного навчання у всесвіті. Віртуальний співрозмовник - це комп'ютерна програма, яка створена для імітації мовної поведінки людини при спілкуванні з одним або декількома співрозмовниками. По відношенню до віртуальних співрозмовникам вживається також назва програма-співрозмовник.

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

Основними конкурентами даного програмного забезпечення будуть віртуальні співрозмовники, які можуть самонавчатися. Самими популярнішими з них є Google assistant та Yandex Алиса.

Вивчивши ринок на плагіати, можу сказати, що існують приклади використання різних типів рекурентних нейронних мереж для ведення

осмисленого діалогу, але розроблених і удосконалених продуктів які використовують за основу рекурентних нейронних мереж немає.

4.4. Оцінка економічної ефективності впровадження програмного забезпечення

Економічна оцінка ефективності пропонованого впровадження оцінена за системою показників, які використовуються у міжнародній і вітчизняній практиці.

При оцінці економічної ефективності використані наступні показники:

– чиста поточна вартість (NPU);
– строк окупності капітальних вкладень;
– індекс прибутковості;
– коефіцієнт ефективності інвестицій.

При ухваленні рішення стосовно доцільності впровадження проекту необхідно враховувати значення всіх показників. Розрахунок показаний у таблиці 4.2.

Таблиця 4.2

Розрахунок чистих грошових надходжень від розробки ПЗ.

Показники, грн	По місяцям							Всього за 6 місяців	Середнє за 6 місяців
		1	2	3	4	5	6		
1. Інвестиції ПЗ	0725	-						41295	6883
2. Витрати до впровадження ПЗ		27000	885	896	905	376	888	36950	6158
- на придбання обладнання		23900						23900	3983
- на придбання витратних матеріалів		1200				00		1700	283
- на маркетинг		1800	800	800	800	800	800	10800	1800
- на електроенергію		100	5	6	05	6	8	550	92
3. Витрати після впровадження ПЗ		1680	690	680	679	680	675	10084	1680
- на оренду		500	00	00	00	00	00	3000	500

приміщення									
- на електроенергію		80	0	0	9	0	5	484	81
- на рекламу		1000	000	000	000	000	000	6000	1000
- на Інтернет		100	00	00	00	00	00	600	100
4. Економія		25320	95	16	26	96	13	26866	4478
5. Амортизація		6882,5	882,5	882,5	882,5	882,5	882,5	41295	6882,5
6. Чисті грошові надходження		18437.5	077,5	098,5	108,5	578,5	095,5	68161	11360
7. Коефіцієнт дисконтування		0,870	,756	,658	,572	,497	,432	-	-
8. Дисконтовані грошові находження		28016	351	671	066	767	065	48936	8156

Коефіцієнти економічної ефективності

Чиста поточна вартість доходів:

$$NPU = 48936 – 40725 = 8211 \text{ грн.} > 0$$

Строк окупності:

$$T = 40725/8156 = 5 \text{ місяців}$$

Індекс прибутковості:

$$ИД = 48936/40725 = 1,2$$

Показник економічної ефективності (NPU – чиста поточна вартість доходів за реалізації впровадження складе 8211 грн., тобто відповідає умовам ефективності, тому що NPU > 0. Середній строк окупності капітальних

вкладень складе 5 місяців. Індекс прибутковості за 6 місяців складе 1,2, тобто ИД > 1, проект варто прийняти. Таким чином, показник ефективності свідчить про те, що дане впровадження є економічно вигідним.

4.5 Висновки

В дипломному проекті виконано «Синтез діалогові моделі на основі рекурентної нейроннї мережі».

При розробці програмного забезпечення синтезу діалогові моделі витрати на його створення склали 40725 грн., очікуваний період створення – 6 місяців та визначено трудомісткість розробленої інформаційної системи (1017 люд-год). Показник економічної ефективності (NPU – чиста поточна вартість доходів за термін реалізації впровадження) становить 8211 грн, тобто відповідає умовам ефективності, т. к. NPU>0. Середній термін окупності капітальних вкладень становить 5 місяців.

Дружність інтерфейсу розробленого ПЗ забезпечує природність, легкість та сприйняття інформації.

Список використаних джерел

1. Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473, 2014
2. Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. A neural probabilistic language model. The Journal of Machine Learning Research, 3:1137–1155, 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. 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.
13. Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. In NIPS, 2014.
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, 2009. ISBN 978 90 272 4825 1.
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, 2014a.

17. Vinyals, O., Toshev, A., Bengio, S., and Erhan, D. Show and tell: A neural image caption generator. arXiv preprint arXiv:1411.4555, 2014b.
18. Will, T. Creating a Dynamic Speech Dialogue. VDM Verlag Dr, 2007.
19. Тема 5. Искусственные нейронные сети. [Електрон. ресурс]. Спосіб доступу URL: http://www.victoria.lviv.ua/html/oio/html/theme5_rus.htm
20. Гусев С.С. Искусственный интеллект как отражение действительности в XXI веке. Интерактивная наука. – Чебоксары: Общество с ограниченной ответственностью "Центр научного сотрудничества "Интерактив плюс", 2016. – 139 с.
21. Гусев С.С. Взаимосвязь человеческого фактора и искусственного интеллекта // Искусственный интеллект: философия, методология, инновации: Мате- риалы III Всероссийской конференции студентов, аспирантов и молодых уче- ных, г. Москва, МИРЭА, 11–13 ноября 2009 г. / Под ред. Д.И. Дубровского и Е.А. Никитиной. – М.: Связь-Принт, 2009. – С. 279–281.
22. Гусев С.С. Нейронные сети, как основное направление развития искусственного интеллекта // Современная российская наука глазами молодых иссле- дователей: Материалы III Международной научно-практической конференции молодых ученых и специалистов (Красноярск, 28 февраля 2013) / Научн. ред. Я.А. Максимов. – Красноярск: Изд. Научно-инновационный центр, 2013. – С. 147–153.
23. Гусев С.С. Современное мировоззрение на проблему искусственного интеллекта // Научное творчество XXI века: Сб. статей / Научн. ред. Я.А. Макси- мов. – Т. 2. – Красноярск: Изд. Научно-инновационный центр, 2012. – С. 73–77.
24. Гусев С.С. К вопросу о «рождении» и истории развития искусственного интеллекта // Искусственный интеллект: философия, методология, инновации: Материалы IV Всероссийской конференции студентов, аспирантов и молодых ученых, г. Москва, МИРЭА, 10–12 ноября 2010 г.

/ Под ред. Д.И. Дубровского и Е.А. Никитиной. – Ч. 2. – М.: Связь-Принт, 2010. – С. 161–164.

25. Азбука ИИ: «Рекуррентные нейросети». [Електрон. ресурс]. Спосіб доступу URL: https://nplus1.ru/material/2016/11/04/recurrent-networks
26. Азбука ИИ: «Машинное обучение». [Електрон. ресурс]. Спосіб доступу URL: https://nplus1.ru/material/2016/09/06/mistakesflow
27. Установка TensorFlow в Ubuntu 16.04. [Електрон. ресурс]. Спосіб доступу URL: https://www.8host.com/blog/ustanovka-tensorflow-v-ubuntu-16-04/
28. Установка Python 3 в Ubuntu. [Електрон. ресурс]. Спосіб доступу URL: https://losst.ru/ustanovka-python-3-ubuntu
29. Как установить CUDA 8 на Ubuntu 16.04. [Електрон. ресурс]. Спосіб доступу URL: https://www.asozykin.ru/deep_learning/2017/02/26/how-to-install-cuda-8-on-ubuntu-16-04
30. Установка Ubuntu с флэшки. [Електрон. ресурс]. Спосіб доступу URL: http://remontka.pro/ustanovka-ubuntu-s-fleshki/
31. A. Graves. Generating sequences with recurrent neural networks. In Arxiv preprint arXiv:1308.0850, 2013.
32. Kyunghyun Cho, Bart van Merrienboer, C¸ aglar Gul¨ c¸ehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. Proc. of EMNLP.
33. Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In Proc. of ICML, pages 160–167. ACM.
34. Tomas Mikolov and Geoffrey Zweig. 2012. Context Dependent Recurrent Neural Network Language Model.
35. Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. Proc. of ICML, pages 1310–1318.

36. Amanda Stent and Srinivas Bangalore. 2014. Natural Language Generation in Interactive Systems. Cambridge University Press.
37. Steve Young. 2002. Talking to machines (statistically speaking). In Proc. of INTERSPEECH.
38. Michael Auli, Michel Galley, Chris Quirk, and Geoffrey Zweig. 2013. Joint language and translation modeling with recurrent neural networks. In EMNLP, pages 1044–1054.
39. Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.
40. Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078.
41. Hao Wang, Zhengdong Lu, Hang Li, and Enhong Chen. 2013. A dataset for research on short-text conversations. In EMNLP, pages 935–945.
42. Goodman Joshua T. (2001). A bit of progress in language modeling, extended version. Technical report MSR-TR-2001-72.
43. Holger Schwenk and Jean-Luc Gauvain. Training Neural Network Language Models On Very Large Corpora. in Proc. Joint Conference HLT/EMNLP, October 2005.
44. Joseph Weizenbaum. ELIZA - a computer program for the study of natural language communication between man and machine. Communications of the Association for Computing Machinery, 9(1): 36–45, 1966.
45. Sanda Harabagiu, Marius Pasca, and Steven Maiorano. Experiments with open-domain textual question answering. In Proceedings of COLING-2000, Saarbruk¨ en Germany, August 2000.
46. W. Xu and A. Rudnicky. Can artificial neural network learn language models. In International Conference on Statistical Language Processing, pages M1–13, Beijing, China, 2000.

47. Cho, K., van Merrienboer, B., Bahdanau, D., and Bengio, Y. (2014b). On the properties of neural ¨ machine translation: Encoder–Decoder approaches. In Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation. to appear.
48. Pascanu, R., Gulcehre, C., Cho, K., and Bengio, Y. (2014). How to construct deep recurrent neural networks. In Proceedings of the Second International Conference on Learning Representations (ICLR 2014).
49. Tomas Mikolov, Stefan Kombrink, Lukas Burget, Jan Cernocky, and Sanjeev Khudanpur. 2011. Extensions of recurrent neural network language model. In ICASSP, pages 5528–5531. IEEE.
50. Mahoney, M. Text Compression as a Test for Artificial Intelligence. In AAAI/IAAI, 486-502, 1999

Приложение А

Текст программы
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._getSummaryName())
    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(self.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_initializer())

    # 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.ratioDataset)  # 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
        # Sampled softmax only makes sense if we sample less than vocabulary size.
```

```
        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 rnn cell
        def create_rnn_cell():
            encoDecoCell = tf.contrib.rnn.BasicLSTMCell(  # Or GRUCell, LSTMCell(args.hiddenSize)
                self.args.hiddenSize,
            )
            if not self.args.test:  # TODO: Should use a placeholder instead
                encoDecoCell = tf.contrib.rnn.DropoutWrapper(
                    encoDecoCell,
                    input_keep_prob=1.0,
                    output_keep_prob=self.args.dropout
                )
            return encoDecoCell
        encoDecoCell = tf.contrib.rnn.MultiRNNCell(
            [create_rnn_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 ?)
```

```
        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_rnn_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.decoderSeqs[-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.corpus](self.corpusDir + optional)
                self.createFullCorpus(corpusData.getConversations())
                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.filteredSamplesPath)  # Saving tf samples
```

```
        else:
            self.loadDataset(self.filteredSamplesPath)

        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.trainingSamples[idSample][1], clean=True)))
            print()
        pass
```

```
def tqdm_wrap(iterable, *args, **kwargs):
  if len(iterable) > 100:
      return tqdm(iterable, *args, **kwargs)
   return iterable
```

ДОДАТОК Б

ВІДГУК

на дипломну роботу магістра на тему:
«Обгрунтування синтезу діалогової моделі на основі рекурентної нейронної мережі»
студента групи 122М-16-1 Ткача Микити Юрійовича

1. Метою дипломної роботи магістра є реалізація діалогової моделі за допомогою впровадження бібліотеки машинного навчання TensorFlow.
2. Актуальність даної теми обумовлена наявністю значних недоліків в традиційному підході до проектування інформаційних систем: витрат часу і коштів.
3. Тема дипломної роботи безпосередньо пов'язана з об'єктом діяльності магістра спеціальності 8.05010301 «Програмне забезпечення систем» напряму 6.050103 «Програмна інженерія» - створення, дослідження і реалізація моделей і програмних засобів.

Наукова новизна результатів, які очікуються, полягає в проведенні аналізу та виявлення недоліків підходу до розробки діалогових систем, а також в розробці удосконаленої методики проектування діалогових агентів на основі використання рекурентних нейронних мереж.
Оригінальність технічних рішень при розробці програмного засобу полягає в використанні зв'язки таких інструментальних засобів як PyCharm, Nvidia CUDA, Python.
Практична цінність дослідження полягає в розробці програмних модулів, програмного продукту, які дозволяють оцінити переваги проектування діалоговіх систем.
Оформлення дипломної роботи магістра виконано на сучасному рівні і відповідає вимогам, що пред'являються до робіт даної кваліфікації. Ступінь самостійності виконання досить висока.

Дипломна робота магістра в цілому заслуговує оцінки «відмінно», а сам автор - присвоєння кваліфікації «професіонал з розробки та тестування програмного забезпечення, дослідник».

Руководитель дипломной
работы магистра, к.т.н. Мещеряков Л.И.,
доц. кафедры ПЗКС М.Ю. Ткач

Приложение Б

Рецензия

на дипломный проект бакалавра на тему:
«Создание мобильного приложения диагностирования состояния человека»

студента группы Книт-12-1 Ткач Никита Юрьевича

Как известно, к одной из достаточно сложных задач, связанных с анализом и проектированием диалоговых систем, относится технология структурирования и формализации таких данных. Человек все больше времени проводит за общением в мессенджерах. Поэтому выбранная тема актуальна в связи с необходимостью сбора и анализа информации .

В рецензируемой работе разработана диалоговая модель на основе рекурентной нейронной сети..

Используемые технологии разработки систем обработки информации напрямую связаны с объектом деятельности магистра направления 6.050101 «Компьютерные науки».

Оригинальность технических решений заключается в разработке диалогового агента на основе рекурентных нейронных сетей; применении комплекта средств разработки PyCharm который позволяет задействовать в работе подсветку синтаксиса.

Студент Н.Ю. Ткач достаточно хорошо разобрался в специфике применения разнообразных информационных технологий.

Учитывая вышеизложенное, можно сделать вывод, что данный проект вполне отвечает требованиям, предъявляемым к квалификационным работам уровня магистра.

Степень проработки компонентов данного проекта, с рядом замечаний, позволяет оценить работу и рекомендовать присвоить студенту Н.Ю. Ткач квалификацию «специалист по информационным технологиям».

Рецензент