HOCHSCHULE SRH HEIDELBERG

SRH University Heidelberg

Master's Thesis

Design and Implementation of a Smart Chatbot Using Deep Learning

Author:	Dsilva, Rianna Maria Roshni
Matriculation Number:	11010632
Address:	18/1 Geshwister Scholl Strasse 69214 Heidelberg Germany
Email Address:	dsilva.rianna@gmail.com
Supervisor:	Dr. Achim Gottscheber (SRH) Sofia Trojanowska (Singularity Design)
Begin:	03. December 2018
End:	28. June 2019

Declaration of Authorship

I hereby declare that the thesis submitted is my own unaided work. All direct or indirect sources used are acknowledged as references.

I am aware that the thesis in digital form can be examined for the use of unauthorized aid and to determine whether the thesis as a whole or parts incorporated in it may be deemed as plagiarism. For the comparison of my work with existing sources, I agree that it shall be entered in a database where it shall also remain after examination, to enable comparison with future theses submitted. Further rights of reproduction and usage, however, are not granted here.

This thesis was not previously presented to another examination board and has not been published.

First and Last Name	Place, Date and Signature

Abstract

A chatbot or conversational agent is a piece of software that can communicate with human beings using natural language. One of the challenges faced by Artificial Intelligence is modeling a conversation between a robot and a human - clearly, efficiently and within an accurate context. Many such chatbots have been developed in recent years which comprehend speech and respond in real time. Startups find it cost-prohibitive to allocate resources for solely for the purpose of customer support, feedback and various other repetitive tasks. This is where Conversational Agents come into play, as they can run 24/7 and with minimum overhead cost for maintenance. The goal of this thesis is to design and develop one such conversational agent based on artificial neural networks. The architecture of the Conversational Agent is of hybrid nature, consisting of primarily a deep learning framework but incorporating rules for efficiency, thus combining the best of both frameworks. In the scope of this thesis, a chatbot which can be deployed on a website or used solely as a stand-alone application as per customer requirements can hold open-ended conversations. It can be further improvised and integrated with system functionalities to perform tasks such as to book flights or set reminders.

Kurzfassung

Ein Chatbot oder Conversational Agent ist eine Software, die mit Menschen kommunizieren kann, indem sie eine natürliche Sprache verwendet. Eine der Herausforderungen, die künstlicher Intelligenz begegnen, ist eine eindeutige, effiziente und innerhalb eines bestimmten Kontextes stattfindende Konversation zwischen einem Roboter und einem Menschen aufzubauen. In den letzten Jahren wurden viele solcher Chatbots entwickelt, die Sprache verstehen und in Echtzeit darauf antworten können. Für Startups ist es rentabel, Mittel zum Zwecke des Kundensupports, für Feedback und für verschiedene andere repetitive Aufgaben einem Chatbot zuzuweisen. Hier kommen verschiedene Chatbots ins Spiel, da diese rund um die Uhr und mit minimalen Wartungskosten betrieben werden können. Das Ziel dieser Abschlussarbeit ist es, einen solchen Chatbot auf Grundlage von künstlichen, neuronalen Netzwerken und Deep Learning zu erstellen. Der Aufbau des Chatbots ist hybrider Natur, primär bestehend aus einem Deep Learning Framework schließt er Regeln für Effizienz mit ein. Dementsprechend kombiniert er das Beste von beiden Frameworks. Im Rahmen dieser Abschlussarbeit soll ein Chatbot, der auf einer Website eingesetzt oder ausschließlich als eigenständige Anwendung nach den Anforderungen der Anwender oder Kunden benutzt werden kann, unlimitierte Konversationen führen. Er kann des Weiteren mit Systemfunktionen ausgestattet werden, um bestimmte Aufgaben auszuführen wie einen Flug zu buchen oder Erinnerungen einzustellen.

Acknowledgement

Firstly, I would like to thank my thesis advisor **Sofia Trojanowska**, Founder and Digital Consultant, Singularity Design, Hannover. Right from my start as an Intern at Singularity Design and all through my Master thesis, she has been a very encouraging mentor and guide both academically and in terms of career-guidance. I am gratefully indebted to her for her very valuable feedback on this thesis.

I would also like to thank **Dr. Prof. Achim Gottscheber**, Dean of Information Technology, SRH Hochschule Heidelberg. The door to Prof. Achim Gottscheber's office was always open whenever I had a question about my research and writing.

I would also like to thank the other external experts who were involved in providing inputs and improvisations for this research project: **Professor Ramakrishna Sunku**, Electronics and Communication Department, Dayananda Sagar College of Engineering, India.

Finally, I must express my profound gratitude to **my family** for their unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without the combined efforts and support of all mentioned above. Thank you.

Notations and Abbreviations

ANN:	Artificial Neural Networks
RNN:	Recurrent Neural Networks
BRNN:	Bidirectional Recurrent Neural Networks
NMT:	Neural Machine Translation
NLP:	Natural Language Processing
SDLC:	Software Development Lifecycle
LSTM:	Long Short-Term Memory
AI:	Artificial Intelligence
ML:	Machine Learning
DL:	Deep Learning
IQ:	Intelligence Quotient
EQ:	Emotional Quotient
CNN:	Convolutional Neural Network
GPU:	Graphical User Interface
RBML:	Rule-based Machine Learning
SML:	Statistical Machine Learning
JSON:	JavaScript Object Notation
AWS:	Amazon Web Services
NLTK:	Natural Language Toolkit
OS:	Operating System

Contents

Part II: Practical

List of Figures

List of Tables

PART I

THEORY

Chapter 1

Introduction to Conversational Agents

A chatbot (also known as the conversational agent or virtual agent) is a computer program that carries out the conversation via auditory or textual methods or even with gestures. Such programs are often designed to conscientiously simulate how a human would behave as a conversational partner.

Conversation in natural language enables people to exchange thoughts and information efficiently. The human-machine conversation aims to facilitate communication between computers and users through natural language processing (NLP). The pre-existing technique of programming using languages such as Java, C, and C++ used to direct the computer to perform certain tasks in a deterministic manner is difficult to grasp and proves to be tedious for NLP. Due to their dependencies on compile, build, deploy, and run in the Software Development Life Cycle (SDLC) process, they cannot adequately adapt and simulate intelligent conversation in real time.

A conversational agent or chatbot is a piece of software program which interprets and responds to human user requests in natural language. As shown in Figure 1.1, this process is divided into two distinct tasks. The proficiency in identifying the user's intent and extracting data and relevant entities contained in the user's request is the first and foremost condition at the core of a chatbot. This information is then used in the next step to provide the most appropriate response in such a way that the user perceives the conversation as similar to one with another human being.

Conversational agents are typically used in dialog systems for various practical purposes and applications including information acquisition or customer service. Some chatbots use sophisticated natural language processing techniques, but many more uncomplicated programs search for keywords in the input statement, then pull a reply with the greatest number of matching keywords, or the most similar wording pattern from the linked database.

The applications of conversational agents span through a wide variety of domains - healthcare, education, customer support, entertainment, marketing, human resources, and many others. They are also used as personal assistants like Cortana, Siri, Google Assistant, Alexa - which tries to help users in performing various actions such as booking a flight or a movie ticket, scheduling a meeting, online shopping or retrieving weather information.

They have different mediums of communication, such as graphics, gestures, but the most commonly used ones are text and speech. Smartphones available in the current market are designed with pre-existing conversational agents, which help in various tasks. For example, for a visually-impaired user, it is no longer necessary to type or select certain buttons to perform a particular task. The user can simply speak to the conversational agent, asking it to 'Set an alarm' or 'Call a contact' and the conversational agent responds with information or performs the required tasks. These functionalities have simplified and uplifted the dependence on smartphones and technologies to make life much easier.

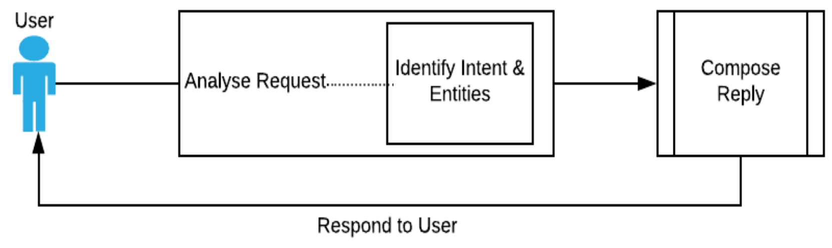
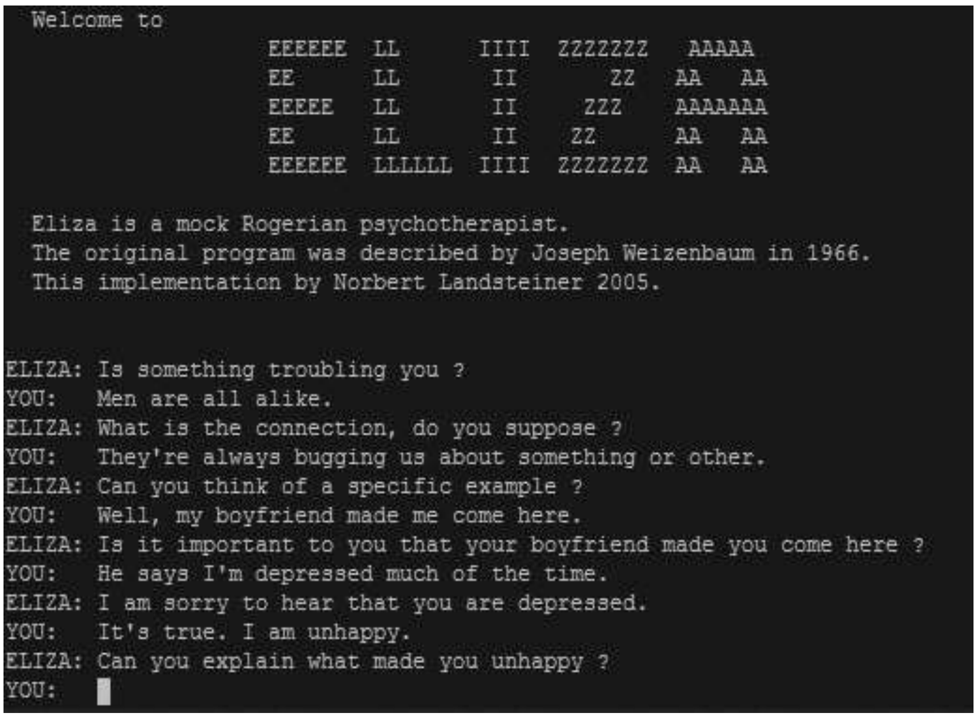


Figure 1. 1: Simplified Model of Chatbot

Chatbots are a class of conversational agents that predominantly use text as the medium of interaction. ELIZA – the first ever chatbot, was created in the 1960s, which was a program developed to interact with its users in the same manner as a psychotherapist. ELIZA was created at the MIT Artificial Intelligence Laboratory by Joseph Weizenbaum, a German-American computer scientist.

ELIZA influenced a whole series of chatbots after its success, such as ALICE, SmarterChild and Albert One. Albert One and ELIZA were based on pattern matching techniques similar to ELIZA. Later in the year 2000, SmarterChild was developed with an added ability to process natural language by parsing a user input into meaningful words. Countless chatbots have been developed with different design techniques and different architectural models, and they are becoming increasingly effective with improved user interfaces, natural language processing, machine learning, and more recently deep learning techniques.

Chatbots have recently received a lot of attention in the industry with the birth of conversational platforms like messaging applications, object recognition, image recognition, and text to speech recognition. The industry is shifting its focus from human-led customer-care interactions towards chatbot-based customer care interfaces which use virtual assistants to answer any queries from the user.



```
 Welcome to
                EEEEEE  LL      IIII  ZZZZZZZ   AAAAA
                EE      LL       II       ZZ   AA   AA
                EEEEE   LL       II     ZZZ    AAAAAAA
                EE      LL       II    ZZ      AA   AA
                EEEEEE  LLLLLL  IIII  ZZZZZZZ  AA   AA

 Eliza is a mock Rogerian psychotherapist.
 The original program was described by Joseph Weizenbaum in 1966.
 This implementation by Norbert Landsteiner 2005.


ELIZA: Is something troubling you ?
YOU:   Men are all alike.
ELIZA: What is the connection, do you suppose ?
YOU:   They're always bugging us about something or other.
ELIZA: Can you think of a specific example ?
YOU:   Well, my boyfriend made me come here.
ELIZA: Is it important to you that your boyfriend made you come here ?
YOU:   He says I'm depressed much of the time.
ELIZA: I am sorry to hear that you are depressed.
YOU:   It's true. I am unhappy.
ELIZA: Can you explain what made you unhappy ?
YOU:   
```

Source: https://en.wikipedia.org/wiki/ELIZA

Figure 1. 2: ELIZA used as an application for psychotherapy

Just as in real life, efficient dialogue management is vital in holding a good conversation. A chatbot-initiated dialogue is comparatively easy to handle and function because this method has a direct sequence of statements to ask a user, and the chatbot knows what to expect as an answer for a particular statement. The sequence of statements is designed based on pre-defined scenarios and contexts. Chatbot drives the user through this loop-like sequence until it has all the information it needs to supply to the user with the relevant information for their respective queries.

The complexity escalates when the chatbot and the user, both can initiate the conversation. It becomes hard for a chatbot to understand the context of the user statement. Hindrances arise in understanding whether the incoming statement is in a sequence with the previous statement or an answer to the chatbot’s follow-up question or a new sentence altogether with a different request. It becomes hard for a chatbot to maintain the state of a user request and the answer, which is addressed using Long-Short term memory (LSTM) models.

A typical LSTM module consists of a cell, an input gate, an output gate and a forget gate. (LSTM'S have been discussed in the following chapters in co-relation to Recurrent and Bidirectional Recurrent Neural Networks.)

Developing a chatbot that can hold open-ended conversations with a human becomes extremely challenging. A chatbot requires the capability of solving multiple problems and various challenges simultaneously in real time. In spite of extensive research in artificial intelligence, chatbots are still restricted in understanding users, consistently following a conversation, and serving in a wider domain.

In scientific literature, chatbots are more formally referred to as conversational agents. In the context of this document, the terms chatbot and conversational agent are used interchangeably.

1.1. Problem Statement

For a chatbot, the very first step after receiving a user request is to understand the request accurately. The chatbot needs to understand the exact intent of a user request in order to respond to the user with the required information. If the chatbot fails to understand the context of the user's request, it will not be able to deliver a suitable or appropriate reply. It may lead to providing an incorrect response or no response at all, which may deem insufficient or vague to a requestor of such information.

Consider an example of a user who requests for certain information. One user types in ''What is weather today in Heidelberg?'' and another user types in ''How is it in Heidelberg today?''. While both of the questions request for the same output, there are different ways in which the information can be requested. Uncertainty and ambiguity in natural language make the responses from the chatbot inefficient and sometimes even inaccurate.

Using Deep Learning architectural model, it is not required to manually create training data or to train the chatbot model to respond appropriately as it is an intelligent machine that learns from the linked corpus dataset. Currently, there are a lot of chatbot models that are developed either on Deep Learning structure or the Rule-based structure. Both these architectures contain pros and cons, which is why the goal of this thesis is to combine the advantages of both these models in order to design and develop a hybrid chatbot model.

1.2. Research Objectives

In order to improve their workflow and minimize costs, Singularity Design is looking for ways to automate part of their interactions with customers on their homepage. In particular, the company would like to deploy a dialogue system solution (chatbot) that would be integrated seamlessly in the existing customer support team. The objective of this thesis is to design one such multipurpose system.

Two main objectives are introduced, aimed to tackle a couple of the challenges mentioned previously in this section.

Objective 1: Research the state-of-the-art models in terms of design and architecture for chatbots and develop a new hybrid architectural model.

In chapter 2, we discuss background and history, the evolution of Artificial Intelligence and its sub-topics ranging from Machine learning to Deep learning. Here we aim to carry out a literature survey about chatbot methodologies, it's challenges and how to involve human computation in chatbots. Chapter 3 deals with concepts and parameters such as Neural Machine Translation (NMT), Recurrent Neural Networks (RNN), Bidirectional Recurrent Neural Networks (BRNN). These concepts are essential in understanding the design and evolution of basic to more advanced chatbots. It also contains examples of successful implementations of Conversational Agents as robots that perform certain tasks and are used for various applications. Chapter 4 includes the architecture and system design explained with the class diagram of the hybrid model discussed in this thesis report.

Objective 2: Device strategy to obtain very large amounts (available free and online) of raw data in order to train the model.

In chapter 5, we commit to designing a methodology to collect and structure high-quality training data with which, the automatic natural language understanding model could be trained at the design phase. We will look at already existing repositories and how the data can be tweaked and customized such that the chatbot responds to a certain query in a specified manner.

1.3. Contributions

The thesis work provides four distinct contributions as follows:

1. Literature survey of chatbots and all necessary parameters in the design of an efficient 21st-century application based chatbot.

2. An approach to collect high quality and diverse training data for a chatbot with the help of human computation and the design of a database using SQL.

3. An approach to determine and measure the quality of chat responses.

4. An approach to detect emojis or emoticons in a user's text and set up a mechanism to score certain responses from the chatbot.

1.4. Thesis Outline

The thesis report is organized in the following three parts:

Part 1 focusses on the theoretical aspects (chapters 1 to 4), describing the background related work in Chapter 2. In Chapter 3, we briefly discuss Conversational Agents with their classifications and all necessary concepts and parameters required in the design and development of a chatbot using Agile Software Development Lifecycle technique. Chapter 4 deals with the architecture of the chatbot, class diagram and system design.

In **Part 2**, the practical aspects related to software implementation are discussed in chapters 5 and 6 starting from an approach for high-quality data collection, its experimental setup and using SQL and TensorFlow in designing the model. Moreover, the results are discussed in Chapter 7. Chapter 8 deals with limitations related to chatbots and how to improve the retention rate of chatbot and then we conclude the thesis by describing the future scope in Chapter 9.

Part 3 of the report includes the Appendices involving the code in Python, a Briefing about the company and Bibliography.

Summary of Chapter: In this chapter, we start off with the overview of a simplified chatbot model and discuss a brief history of chatbots and the very first chatbot model that was developed. In the further sections, focus is given to Problem statement, the research objectives and Individual thesis contributions discussed in Section 1.3 and end with a brief outline of the way this thesis report has been structured.

Chapter 2

Background and Literature Survey

A comprehensive definition of Artificial Intelligence is a computer program which can carry out an integrated simulation of the human brain. Machine Intelligence also called Artificial Intelligence (AI), is intelligence demonstrated by machines, in contrast to the natural or habitual intelligence displayed by humans and other animals or species. These intelligent machines are now commonly referred to as ‘‘Robots’’. In the field of computer science, Artificial Intelligence or more commonly referred to as AI, is defined as the study of "intelligent agents" - any device that perceives its surroundings and takes actions that amplifies its chance of successfully achieving its goals. Colloquially, the phrase "artificial intelligence" is applied when a machine imitates cognitive functions that humans associate with other human minds, such as learning, problem-solving and cross-questioning.

Artificial intelligence can be compartmentalized into three different categories:

- **Analytical AI:** Analytical AI has characteristics consistent with cognitive intelligence, generating a cognitive representation of the world and using learning based on past experiences to impact future decisions.
- **Human-inspired AI:** Human-inspired AI has elements from emotional as well as cognitive intelligence, understanding human emotions, in addition to cognitive elements, and applying them in their decision making.
- **Humanized AI**: Humanized AI depicts characteristics of all types of competencies (i.e. cognitive, emotional and social intelligence) and is capable of self-consciousness.

The goal of Artificial Intelligence is to improve computer functions which are related to human knowledge, for example - learning, problem-solving and reasoning. Artificial intelligence, Internet of Things and increasingly complex algorithms currently influence our lives and our civilization more than ever. The areas of AI application are diverse and the possibilities extensive: in particular, because of the improvements in computer hardware, certain AI algorithms already surpass the capacities of human experts today.

2.1. History of Artificial Intelligence

The domain of artificial intelligence is relatively new. In the early 1950s, scientists and researchers began to consider the possibility of machines processing intellectual capabilities similar to those of human beings. The field of AI research was born at Dartmouth College in 1956 at a workshop. Allen Newell (CMU), Herbert Simon (CMU), John McCarthy (MIT), Marvin Minsky (MIT) and Arthur Samuel (IBM) became the leaders and founders of research in the field of Artificial Intelligence. This was when the formation of Artificial Intelligence as an academic discipline began.

Alan Turing (1912-1954) a British mathematician, understood as early as the 1940s that there would be an endless debate about the difference between artificial intelligence and original intelligence. He realized that asking whether a machine could think was the wrong question. The right question is: “Can machines do what we (as thinking entities) can do?”

He fabricated a version of what we today call the ‘‘Turing Test’’. In it, a jury asks questions to a computer. The role of the computer is to make a significant proportion of the jury believe, through its answers that it’s a human. Turing Test is a tactic of inquiry for determining whether a computer is capable of thinking in a similar manner to that of a human being.

A Turing Test for Emotion: What would it take for an AI machine to convince a Turing Test jury that it is a human? This would certainly require much more than just being able to paint, compose music or perform complex mathematics. The machine or robot would have to be able to connect with the jury members on a human level by exhibiting characteristics of human emotional intelligence, such as empathy. This parameter is popularly referred to as Emotional Quotient (EQ). In the 21st century, a chatbot or AI Entity is required to possess both factors in order to be considered valid or effective – Intelligence Quotient (IQ) and Emotional Quotient (EQ).

To many researchers and scientists, the question of whether a computer can pass a Turing Test has become insignificant. Instead of focusing on how to convince a person that they are conversing with a human and not a computer program, the real focus should be on how to make a human-machine interaction more intuitive and efficient – for example, by using a conversational interface.

Previously, theoretical AI was sub-divided into the following categories:

- **Artificial Narrow Intelligence:** The intelligence learns about a single task which it has to perform efficiently and with smartness.
- **Artificial General Intelligence:** Its smartness can be applied to perform various tasks as well as learn and improve itself. It is comparatively just as intelligent as the central nervous system of a human.
- **Artificial Super Intelligence:** An aspect of intelligence which is more powerful and sophisticated than a human’s intelligence.

2.1.1 Applications of Artificial Intelligence

Artificial Intelligence has a variety of applications ranging from Human Resource (HR) onboarding, learning management systems, Logistics and supply chain management to digital marketing techniques. The reason for its vast expansion into these fields is the ever-expanding demand for technology to automate various processes. An example of this is the evolution of regular billboards to digital or electronic billboards used as an advertising technique for marketing. Some other applications in various fields are explained below.

Gaming: AI plays an important role for a machine to think of a large number of possible positions and predictions based on permutations and combination algorithms in strategic games. For example, chess and Alpha Go.

Vision Systems: Systems comprehend, explain, and describe visual input on the computer using image or object recognition.

Speech Recognition: Speech recognition systems have the ability to hear and express sentences and understand their meanings while a person talks to it. For example, Siri, Cortana, and Google assistant.

Virtual Agents: From simple chatbots to advanced robots that interact with human beings.

Machine Learning Platforms: Providing algorithms, data, APIs, development and training toolkits and computing power to design, train, and deploy software models into other machines, applications, and processes.

AI-optimized Hardware: Graphics processing units (GPU) such as NVIDIA and appliances specifically designed and architected to efficiently run computational tasks.

Decision Management: Programs that insert rules and logic into AI systems and are used for initial setup, training, ongoing maintenance and tuning.

Deep Learning Platforms: A very specific type of machine learning concept that consists of artificial neural networks with multiple abstraction layers.

Text Analytics and NLP: Natural language processing supports text analytics by facilitating the understanding of sentence structure including the meaning, sentiment, and intent through statistical and machine learning method.

Biometrics: Biometrics is used nowadays for extremely secure purposes including, fingerprint recognition, image recognition (Face ID in mobile phones) as well as speech recognition.

Robotic Process Automation: Implementing the use of scripts to automate human action to support efficient and seamless business interactions.

2.1.2 Overview of Sub-branches of Artificial Intelligence

Artificial Intelligence is directly and indirectly related to many other sub-technologies. The figure below depicts an overview of the sub-branches of Artificial Intelligence. These technologies are said to have branched out from Artificial Intelligence and are as follows;

- **Neural Networks:** Brain modeling, time series prediction & classification.
- **Planning:** Scheduling & game playing.
- **Robotics:** Intelligent control and automation exploration.
- **Machine Learning:** Decision tree learning and version space learning.
- **Natural Language Processing**: Machine translation.
- **Evolutionary Computation:** Genetic algorithms and genetic programming.
- **Vision:** Object rejection and image understanding.
- **Speech Processing:** Speech recognition and production.

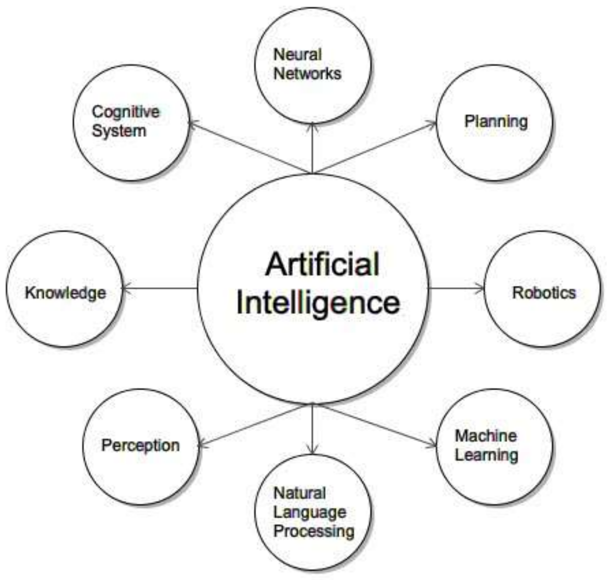


Figure 2. 1: Classification of sub-technologies of Artificial Intelligence

2.2. Machine Learning

Machine learning (ML) is a sub-division of artificial intelligence that uses statistical techniques to give computer systems the ability to ‘learn and absorb’ (For example: progressively improve performance on a specific task) from data, without being explicitly programmed. It is a data analytics method that coaches computers to do what comes naturally to humans and animals, which is to learn from experience. The term ‘Machine Learning’ was coined by Arthur Samuel in 1959. Machine learning explores the fabrication of algorithms that can acquire and understand from and making predictions on data. These algorithms prevail over strict static program instructions by making data-driven predictions and decisions, by developing models with the help of sample inputs.

Machine learning is applied in a range of computing tasks where designing and programming explicit algorithms which perform rather good, are impractical. Example applications include computer vision, email filtering, and detection of network intruders. Machine learning is closely related to (and most often overlaps with) computational statistics, which also centers on prediction making models. It has strong ties to mathematical optimization (permutations and combinations), which delivers methods, theory, and application domains to the field.

In the commercial field, this is known as predictive analytics. These analytical models allow analysts, researchers, data scientists and engineers to generate reliable, repeatable decisions and end-products and uncover hidden insights through learning from historical relationships and trends in the data. The Machine learning models are infamously known to be applied in the Stocks and Shares to make predictions about the future of the stock market.

2.2.1 Artificial Intelligence and Machine Learning

The terms ‘machine learning’ and ‘artificial intelligence’ are sometimes used interchangeably due to the recent focus on imitating the human thought process. Machine learning has become an essential aspect of the contemporary understanding of AI, but for clarity – ‘artificial intelligence’ is a wider term, while ‘machine learning’ is its subfield.

Intelligence is intangible and is composed of - Reasoning, Learning, Problem Solving, Perception and Linguistic Intelligence.

The principle of machine learning is such that, rather than having to be taught to do everything step by step, machines can learn to work and improve by observing, classifying and failing, just like humans do. In short, machines imitate humans and learn through experience.

2.2.2 Supervised and Unsupervised Learning

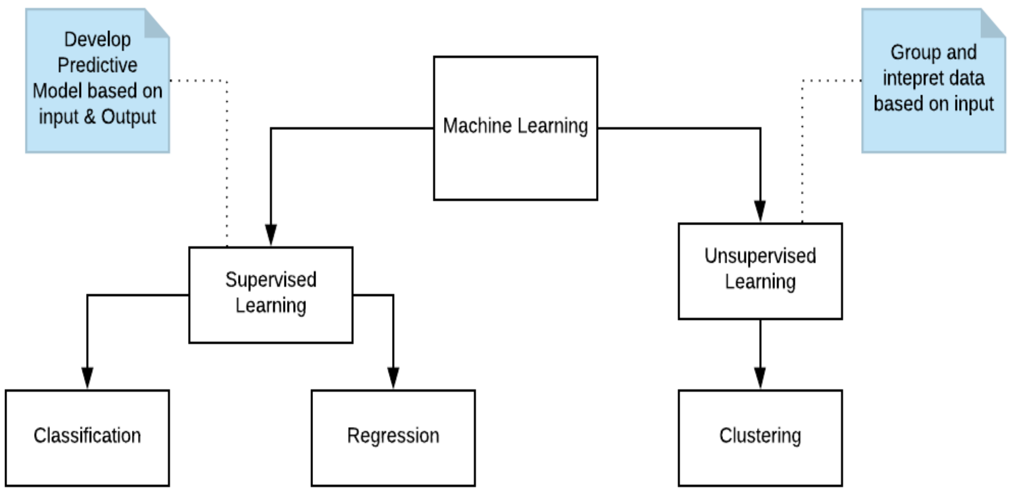


Figure 2. 2: Classification of Machine Learning Techniques

Machine Learning uses two types of techniques as described below:

1. **Supervised Learning:** Trains the model on known input and output data so that it can predict reasonable future outputs or results. The model makes predictions based on evidence in the presence of uncertainty. In order to develop predictive models, Classification and Regression techniques are developed.

    - Classification techniques predict discrete responses and classify information into categories. For example: detecting whether a tumor is cancerous or benign or if an incoming email is genuine or spam. Typical applications include medical imaging and speech recognition.

    - Continuous responses can be predicted by Regression techniques. For example: changes in temperature or fluctuations in power demand. Typical applications include water or electricity load forecasting.

2. **Unsupervised Learning:** Technique that finds hidden patterns or intrinsic structures in the input data.

    - Clustering is the most familiar unsupervised learning approach. It is applied for exploratory data analysis to find clusters, groups or hidden patterns in the data corpus. Applications for cluster analysis include object recognition, gene sequence analysis, and market research.

Deep diving into the branches of different types of models developed by the three algorithms:

1. **Classification Algorithms:**
   These include Support vector machines, Discriminant analysis, Naïve Bayes and Nearest neighbor algorithms.

2. **Regression Algorithms:**
   Regression algorithms include Linear regression and GLM techniques, SVR, GPR, Ensemble methods, Neural networks as well as decision trees.

3. **Clustering Algorithms:**
   This set of algorithms mainly deal with K-means, K-medoids, Fuzzy, C-means, hierarchical, Gaussian mixture, neural networks, and the hidden Markov model.

An overview of these models and the techniques adopted is given below in a simplified manner.

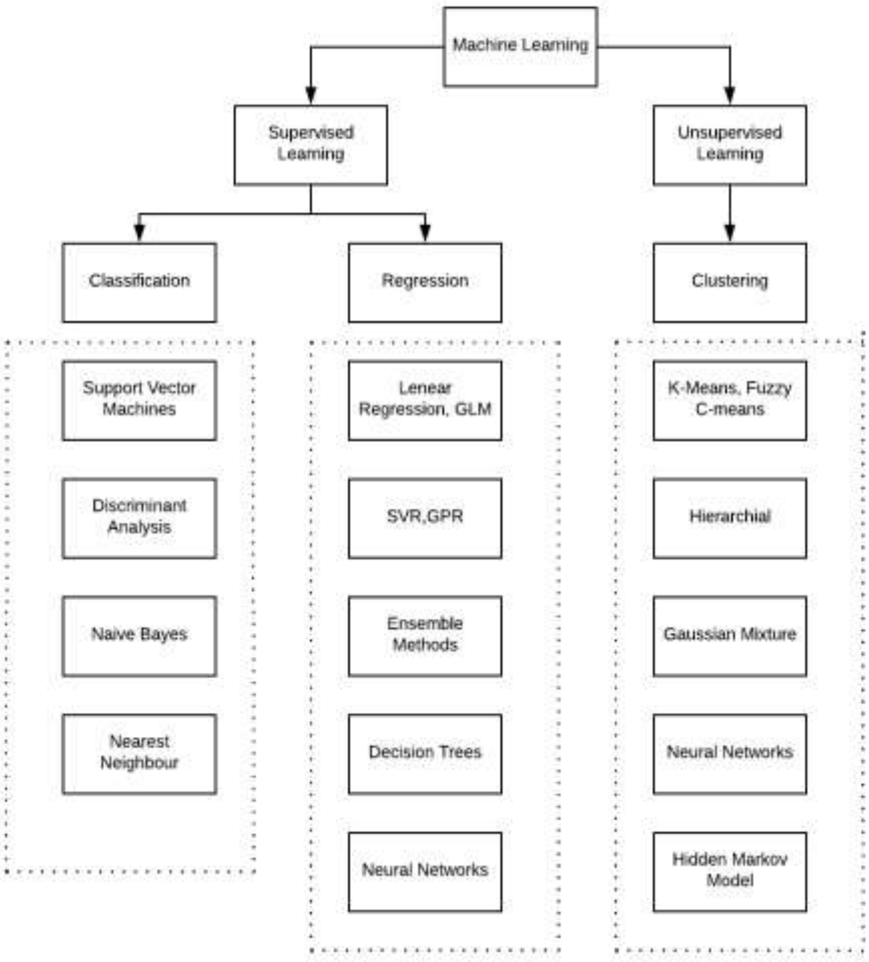


Figure 2. 3: Supervised & Unsupervised Learning Methods

2.3. Deep Learning

Deep Learning is a subfield of Artificial Intelligence that deals with algorithms inspired by the function of the brain and its neurons called artificial neural networks. Fundamentally, learning can be supervised, semi-supervised and unsupervised.

Primarily, Deep Learning is a machine learning technique that instructs computers to try and do what comes naturally to humans: to learn from experience or examples. Deep learning is a key feature behind autonomous driverless cars, enabling them to recognize a stop sign or to ascertain a pedestrian from a different object such as a speedbump or streetlight. Just as the transition from fuel-based cars to electric cars brought about a remarkable shift in the industry, most automobile manufacturers (Tesla, BMW, Volkswagen, Audi) are now extensively focusing on autonomous driverless cars. Deep leaning plays a significant factor for voice control in consumer devices like phones, TVs and tablets.

Deep learning methods use architectures derived from neural networks, which is why deep learning models are often referred to as deep neural networks. The terminology ‘deep’ usually refers to a number of hidden layers. Traditional neural networks only contain 2 or 3 hidden layers, while deep networks usually have more than 100. These models are designed and trained by using large classes of labeled data and neural network architectures that learn features directly from data. These models do not require manual feature extraction which is an essential aspect in Machine Learning models.

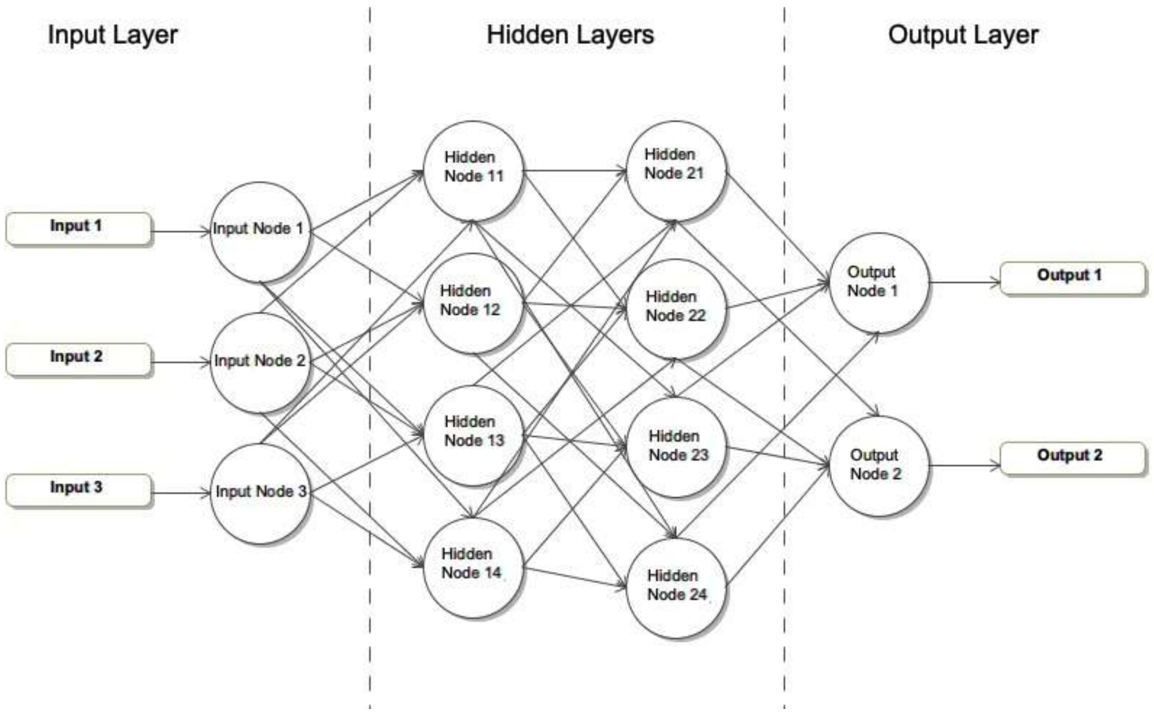


Figure 2. 4: Artificial Neural Network Structure

Convolutional neural networks (CNN) is one of the most popular types of Deep Neural Networks. A convolutional neural network convolves learned features and highlights with input data, and uses 2-dimensional convolutional layers, making this architecture well adapted for handling 2D data, such as in images. CNN's are regularized versions of multilayer perceptron’s. Multilayer perceptron’s usually imply to fully connected networks wherein each neuron in one layer is connected to the subsequent layer’s neurons until it reaches the output node.

Convolutional Neural Network functions by extracting features directly from images. Relevant features are not predefined but are learned while the network trains on a collection of images or pictures. Automated feature extraction makes deep learning models extremely precise as well as complex for computer vision tasks such as object and image classification. These networks detect different features of an image by comparing with hundreds of hidden layers. Every hidden layer tends to add up to the complexity of the learned image features.

For example, the first hidden layer could learn how to detect points, the second layer to thoroughly detect edges and the last learns how to detect more complex shapes.

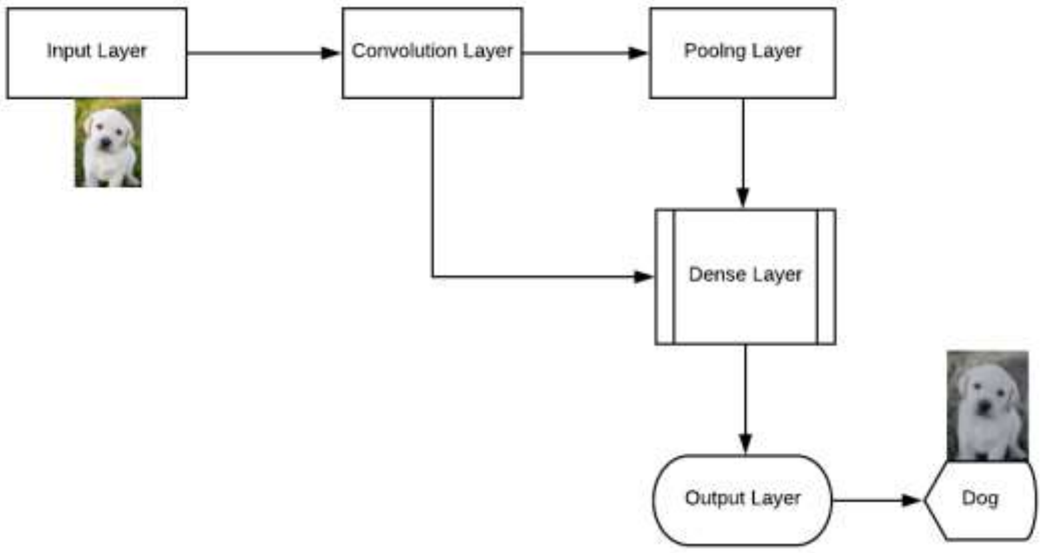


Figure 2. 5: Process of Object Detection

From the illustration provided above, an image is fed into the input layer which is first passed through a convolution layer, followed by a pooling n-layer, a dense layer and finally through the output layer. From the output layer, the system arrives at a conclusion to detect if the image at the input layer is a Dog or not (in this case) which is then displayed as the output. Each layer performs a specific task such as RBG to Grey conversion, selecting foreground and removing background, noise cancellation, detecting salt and pepper noise and various other image processing techniques to accurately detect the object. Google's Image Recognition system adopts a similar model.

Convolutional Neural Networks serve a wide range of applications in image and video recognition, natural language processing, recommender systems, image classification, and medical image analysis.

2.3.1 Machine Learning versus Deep Learning

Deep learning is a more advanced form of machine learning. A machine learning model starts learning with relevant features being physically extracted and secured from images. These intents are then used to create a model that categorizes the objects in the image. Whereas in a deep learning workflow, relevant features are automatically extracted from images. Additionally, deep learning performs 'end-to-end learning' – where a network is given raw information and a specific task to perform, such as classification, and it learns how to do this automatically.

Another perk of the deep learning model is that they often continue to improve as the size of the dataset increases. In machine learning, it is necessary to manually choose features and a classifier is used to sort these images. Feature extraction and modeling steps are automated in deep learning.

2.4. Classification of Conversational Agents

Conversational Agents are divided into various classes based on the medium, device, purpose, and aim, as depicted below.

Mediums:
Mediums of communication is a channel of sensory input or output between the conversational agent and the user. The interaction between the two can be text-based or speech-based, or it can also be based on a graphical user interface (GPU).

Devices:
Previously, conversational agents were used on desktops or laptops but are now widely used in phone-based applications externally or even integrated into applications on Android or IOS operating systems.

Purposes:
The purposes can be categorized into three sorts. One is an informational conversational agent which provides information to its users like travel information. Second is transactional where users can perform actions with conversational agents such as purchasing a bus ticket. The third is conversational, which is designed just for the sake of conversation of chatbots with their users as seen in mobile mental health applications.

Domains:
Conversational agents which serve in one specific domain as single domain agents (Example: medical chatbots). Multiple domains agents which serve in many domains are called generic if they serve for general conversations like small talk.

People converse with each other to exchange information and thoughts, to discuss an issue, resolve conflicts, increase mutual understanding, or just for fun. Conversation plays a vital role in the communication of one human with another human. Humans can comfortably express their thoughts in natural language.

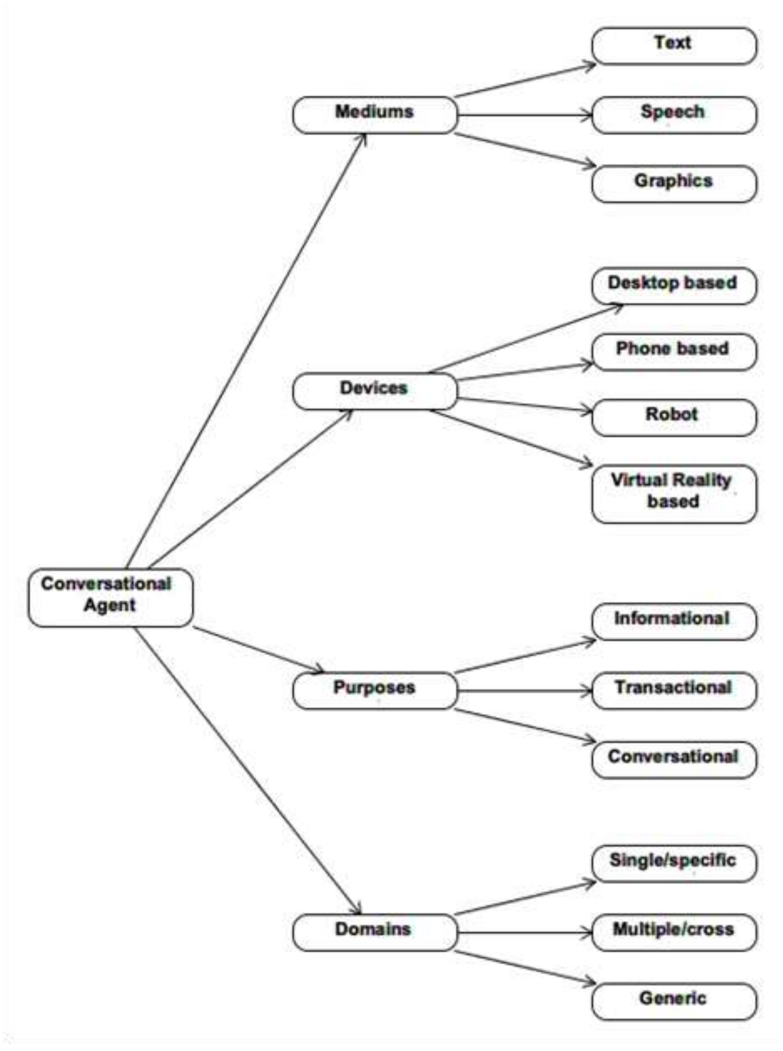


Figure 2. 6: Classification of Conversational Agents

These classes of utterances help to determine intent, context, sequence and understanding in a discussion. Usually, human beings don't just effectuate utterances while conversing with each other, they also ask questions, make promises, share compliments and so on. And while interacting in natural language, they express and convey much more than just the meanings of the words spoken. Face-to-face interaction conveys meaning with their nonverbal behaviors such as their gestures, facial expression and body language. This makes the conversation more expressive and relatable, and people can effectively share their thoughts and provide information to each other.

In recent times, machines and robots are designed in a way that they are not only intelligent but also empathetic. This means that they try to extract the emotional state of the users and respond to them appropriately. Some of these robots are even deployed in hospitals around the world. Table 2.1 gives examples of Classification of Utterances.

Class	**Description**	**Examples**
Assertiveness	Convincing the addressee about a specific task	Can this be fast-forwarded?
Greetings	Opening a conversation	Hi, Welcome
Directives	Attempt to get the addressee to perform a task	This must be completed by the end of tomorrow.
Expressives	Explain the psychological state of addresses	Thank you, I apologize
Declarations	Making bold statements	I quit, This does not make sense
Commissives	Making predictions or committing to the future course of action	I promise to fulfill

Table 2. 1: Examples of classification of utterances.

2.5. Practical Examples of Applied Deep Learning

Industrial Automation: Deep learning is helping comply with employee safety regulations and standards around heavy machinery particularly around construction sites, by automatically detecting when people or objects are within an unsafe distance of machines.

Defense and Aerospace: Identifying objects using satellites is achieved with the help of deep learning for military troops or even for commercial planes in cases of emergency landing.

Medical Research: Cancer researchers depend on deep learning algorithms to automatically detect cancer cells.

Electronics: Deep learning is being applied for automated hearing and speech recognition as well as translation.

Some practical implementations of Deep Learning that serve multiple domains and recently gained the spotlight are described in detail in the next section.

2.5.1 ROBOT SOPHIA

Source: https://en.wikipedia.org/wiki/Sophia_(robot)

Figure 2. 7: Robot Sophia by Hanson Robotics

Conceptually, Sophia is similar to the computer program ELIZA, which was one of the very first endeavors at simulating a human conversation. For specific questions or phrases, the robot has been programmed to output some pre-trained responses. These responses are used to create a deception that the robot is able to understand the intents and contexts in conversation and form responses to questions like "Is the girl sitting or standing?". The data is uploaded on cloud network which allows input and responses to be analyzed with blockchain technology.

2.5.2 IBM WATSON

IBM Watson is at the forefront of the new era of computing. Over the last couple of years, the Watson limits have been expanded (Watson on IBM Cloud) and propelled machine learning capacities and upgraded hardware open to architects and authorities. It’s no longer a request answering system and can now ‘see’, ‘hear’, ‘read’, ‘talk’, ‘translate’, ‘learn’ and ‘endorse’.

Summary of Chapter: Because of the large research gap in the design of hybrid models which combine the architectural advantages of both the statistical (deep) learning models and rule-based learning models, it is necessary to understand the working behind both these models.

To summarize, this chapter covers the History of Artificial Intelligence, explains in detail the concepts of Machine and Deep Learning. Supervised and unsupervised learning techniques, being the two distinct classifications of Machine Learning techniques are clearly described. A simplified illustration of the classification of Conversational Agents is given and two important applications of Deep Learning (Robot Sophia and IBM Watson) are explained.

Chapter 3

Concepts and Parameters in Chatbot Design

3.1. Machine Translation

Machine or automatic translation is perhaps one of the most challenging artificial intelligence tasks given the complexity and fluidity of human language or natural language as termed in the field of AI. Typically, rule-based systems were used for this task, which were replaced in the 1990s with statistical methods. In recent years, deep neural network models achieve state-of-the-art results in natural language translation of one language to another. One common and widely used example is Google Translate.

Machine translation is the task of mechanically altering or converting the source text in one language to text in another language. The main reason for the complexity of this task is because there is more than one way in which a certain text can be translated. Just as in the language of English, a certain sentence can be interpreted in different ways, so also the computer or the algorithm tends to make translations or comprehend certain statements differently. This is the reason for the popularity of programming languages which contain a strict set of instructions leaving no room for ambiguity. Historically, machine translation can be divided into two main categories, as shown below.

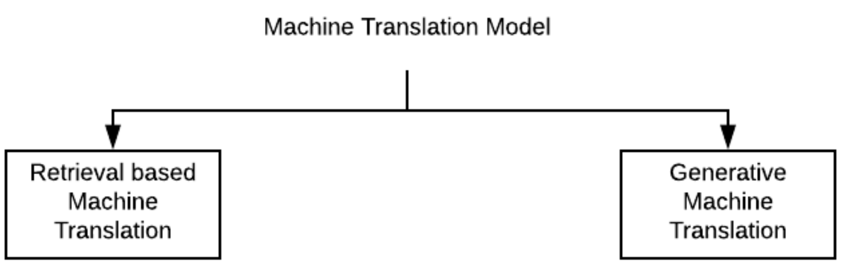

Machine Translation Model
Retrieval based Machine Translation
Generative Machine Translation

Figure 3. 1: Classification of Machine Translation Categories

1. Rule-based Machine translation (RBMT): Classical techniques of translation usually involve certain rules developed by linguists operating at the lexical, syntactic, or semantic level. Because of these rules, the name of this technique is - Rule-based Machine translation or Retrieval-based Machine Translation.

The Handbook of Natural Language Processing and Machine Translation, states that ‘‘Rule-based Machine Translation or RBMT is symbolized with the explicit use and manual creation of linguistically informed rules and representations.’’ This is the main disadvantage of the rule-based system.

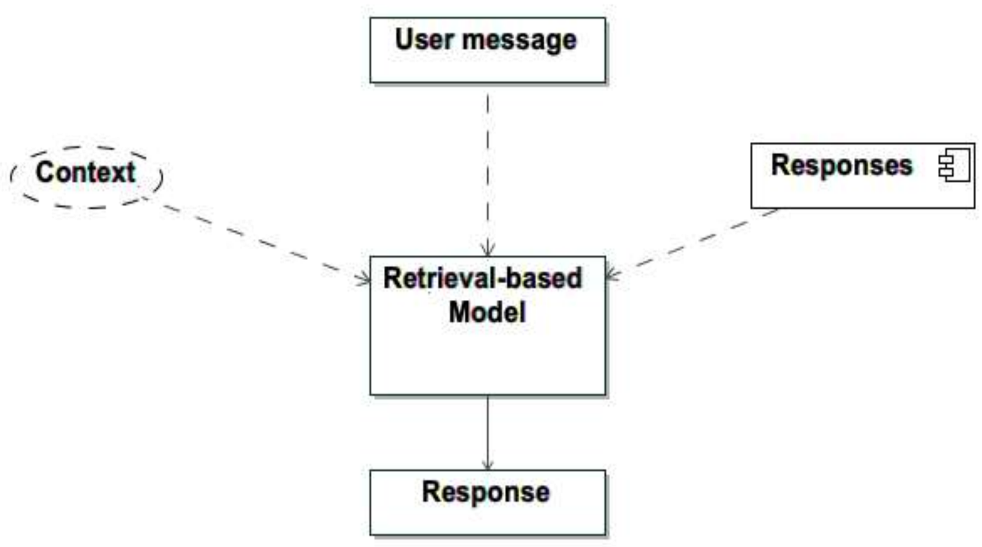


Figure 3. 2: Rule-based Machine Translation Model

2.Statistical machine translation (SMT): Statistical Machine Translation or Generative Machine translation is the application of statistical models that learn to translate text from a source to a target language, which is fed with a large corpus of examples. It is a data-driven approach and it necessitates only a corpus of examples with both source and target language text. This means that linguists are no longer required to specify the rules of translation.

Although valuable, statistical machine translation methods suffer from a limited or finite focus on the phrases being translated, misplacing the broader nature of the target text. The importance given to data-driven approaches also means that some important syntax distinctions known by linguists may have been ignored.

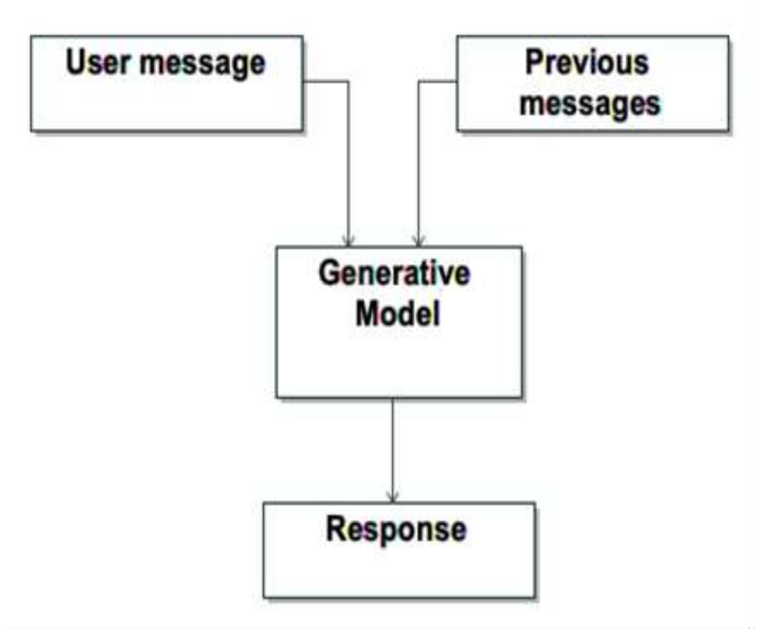


Figure 3. 3: Statistical Machine Translation Model

3.1.1 Neural Machine Translation (NMT)

Neural machine translation is the use of neural network models to learn a statistical model for machine translation. Neural Machine Translation (NMT), is a vast artificial neural network that makes use of feature learning and deep learning to model full sentences with machine translation. This methodology specializes in producing continuous sequences of words better than the traditional approach of using a recurrent neural network because it mimics how humans translate sentences. Essentially, when humans translate, they read through the entire sentence, understand what the sentence means, and then work on the sentence translation.

Neural Machine Translation performs this process with the architecture of encoder-decoder. The encoder transforms the sentence into a vector of numbers that represent the meaning of the sentence and the decoder takes this meaning vector and constructs a translated sentence. Essentially, the NMT model consists of an encoder BRNN and a decoder BRNN, in a multi-layered bidirectional recurrent neural network. This sequence-to-sequence model is often used for machine translation, text summarization, and speech recognition.

The key advantage to the approach is that a single system can be trained immediately on the source and target text, no longer needing the funnel of specialized systems used in statistical machine learning. These systems are said to be end-to-end as only one model is required for the translation.

From Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation, ‘‘The strength of NMT lies in its ability to learn directly, in an end-to-end fashion, the mapping from input text to associated output text.’’

3.2. Recurrent Neural Networks (RNN)

Neural Machine Translation models are broadly classified into two main categories:

1. Recurrent Neural Networks (RNN)
2. Bidirectional Recurrent Neural Networks (BRNN)

In Figure 3.4, the structure of a Recurrent Neural Network is illustrated. The input words from the text are denoted as x_0, x_1, x_2 …, their subsequent predicted words denoted as y_0, y_1, y_2 …and the information for the previous input words denoted as s_0, s_1…s_i.

For Machine learning tasks that contain enormous amounts of sequential data, Recurrent Neural Networks provide a state-of-the-art algorithm that contains an internal memory. This algorithm was primarily used by Apple’s virtual assistant (Siri) and Google Voice Search.

Before the adaptation to Recurrent neural networks, machine translation proved to be inefficient because the length of certain statements was long, and the system was unable to efficiently remember the lengthy sequences and then provide an appropriate translation for the same. An example of sequential data could be a Time series data, which is just a series of data points that are listed in the order of time.

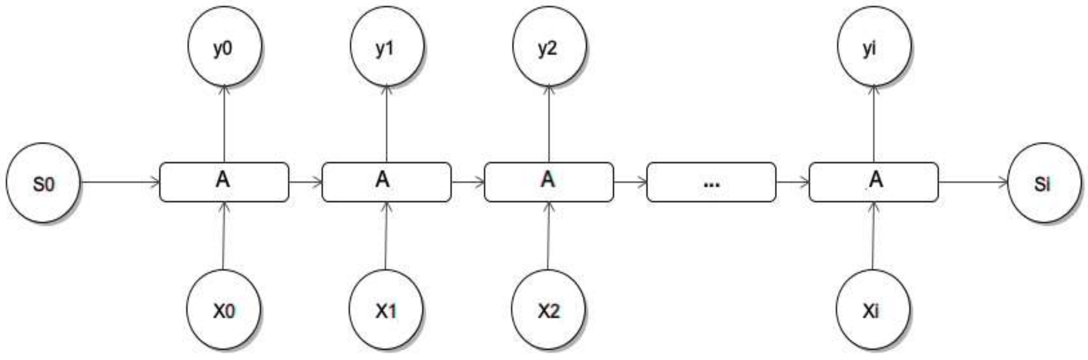


Figure 3. 4: Recurrent Neural Network Model

Consider an example of an input statement:

User: "Hi there, Can I know when the next available flights from Berlin to Bangalore along with their layover timings and prices?"

Now for a system without a memory, translating a sentence with 22 words (226 characters) efficiently definitely seems like a challenge. Previously, chatbots were able to retain a maximum of 5-word sentence structures and then provide appropriate responses for the same. With the internal memory used in Recurrent Neural Networks, this issue can be resolved and the chatbot can retain much lengthier data sequences and provide better translations for the same.

Recurrent Neural Networks are relatively old, like many other deep learning algorithms. They were initially created in the late 1980s. Recurrent Neural Networks have become increasingly famous in the recent years, because of the increase in available speed and computational power, the massive amounts of data that we have nowadays and the invention of LSTM in the 1990s. Because of their internal memory, RNN's can remember important things about the input received, which enables them to be very precise in predicting future words. This is the reason they are the preferred algorithm for sequential data like audio, time series, speech, text, financial data, video, weather and much more as they can form a much deeper understanding of a sequence and its context when compared to other algorithms.

3.2.1 Architecture and Working of Recurrent Neural Networks

In a Feed-Forward neural network, the information moves only in one direction, starting from the input layer through the hidden and finally to the output layer. The information is directed straight through the network. As a result of this, any information passing through the network never touches the same node twice.

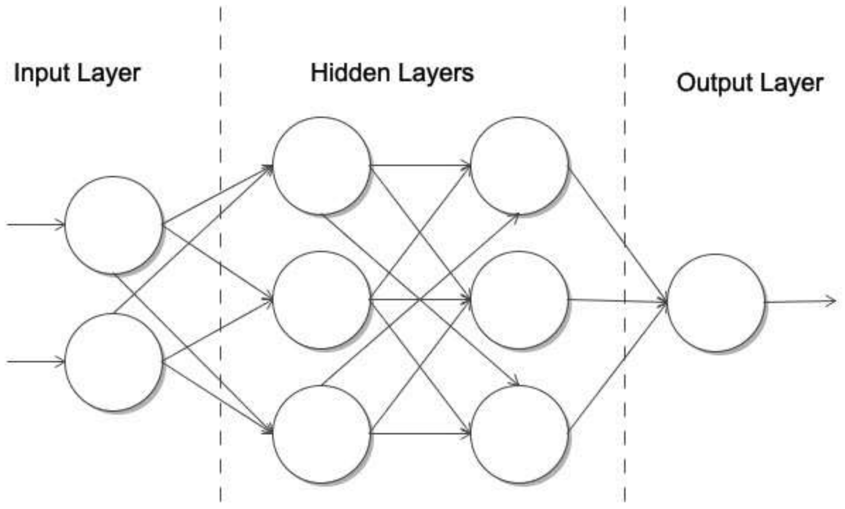


Figure 3. 5: Feed-Forward Neural Network Structure

Feed-Forward Neural Networks have no recollection of the input received previously and are therefore faulty in estimating what’s coming next. Since a feed-forward network considers only the current state of the input, it has no notion of order in time. They do not recall anything about what happened in the past except for their training.

In a Recurrent Neural Network, the information cycles through a loop. When a decision is made, it takes into account the information gathered from the previous input as well as the current input.

Figure 3.6 illustrates the difference in the information flow between a Recurrent Neural Network and a Feed-Forward Neural Network. Another good approach to illustrate the concept of an RNN’s memory is to explain it with an example:

Imagine you have a feed-forward neural network and feed the word ‘proton’ as input and it processes the word character by character. By the time it reaches the character ‘t’, it has already forgotten about ‘p’, ‘r’ and ‘o’, which makes it almost impossible for this type of neural network to predict which character comes is next in sequence.

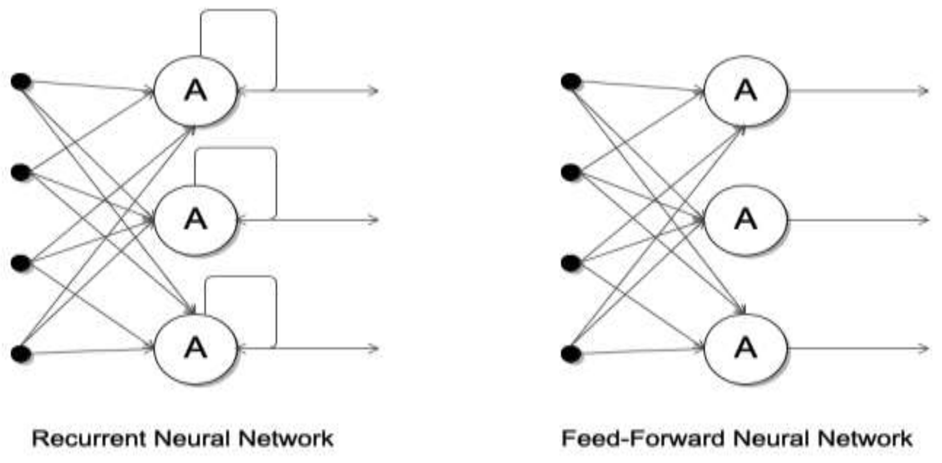


Figure 3. 6: Difference between Recurrent & Feed-Forward Networks

A Recurrent Neural Network is able to remember exactly that due to its internal memory. It produces an output, which it copies and loops back into the network. Therefore, a Recurrent Neural Network has a series of two inputs, the present input and the recent past input.

Similar to all other Deep Learning algorithms, a Feed-Forward Neural Network assigns, a weight matrix to its inputs and then produce the output. On the other hand, Recurrent Neural Networks apply a specific weight to the previous and also to the current input.

There are two major drawbacks of Recurrent Neural Networks, that led to the development of Bidirectional Recurrent Neural Networks, which is currently the most popular and reliable network used both in Machine Learning and Deep Learning.

1. Exploding Gradients: When the algorithm assigns an unnecessarily high importance to the weights, without much reason.

   Solution: The problem of exploding gradients can be resolved by truncating the gradients.

2. Vanishing Gradients: When the values of a gradient are below a threshold value, the model stops learning or deprecates the learning rate to a very large extent.

   Solution: The problem of Vanishing gradients can be solved by using LSTM, which was later developed Sepp Hochreiter and Juergen Schmidhuber, discussed in the next subsection.

3.2.2 Long-Short Term Memory (LSTM)

Long Short-Term Memory (LSTM) networks are an extension of recurrent neural networks, which basically extends their memory. These models are extremely well adapted to learn from important experiences that have very long-time lags in between.

LSTM’s enable Recurrent Neural Networks, store and recall their inputs for long periods of time. The reason for this is that the memory unit of the LSTM is similar to that of a computer memory which can read, write and delete information from memory. As the information enters through the input gate, based on the input, the network assigns the incoming input certain weights. This means that it learns over time, which information is important and which not.

An LSTM has three gates: input, forget, and output gate, as illustrated in Figure 3.7. These gates are responsible of various decision tasks such as: letting new input flow (input gate), delete the information because it isn’t important (forget gate) or to let it impact the output at the current time step (output gate).

In an LSTM model, the gates are in the form of sigmoids (analog) and they typically range from 0 to 1. The fact that they are analog enables backpropagation through the network.

The issue of vanishing gradients is resolved with the help of Long Short-Term Memory (LSTM) because it keeps the gradients steep enough and therefore, the training is relatively short while the accuracy is high.

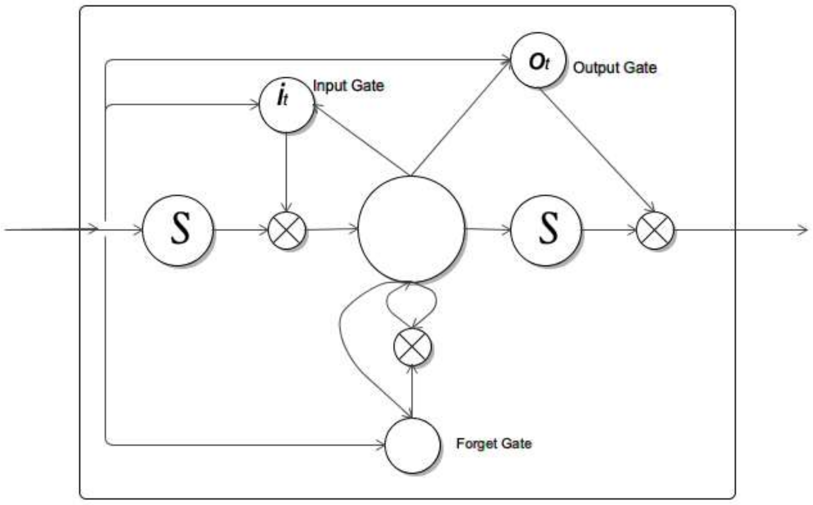


Figure 3. 7: Long-Short Term Memory (LSTM) Model

3.3. Bidirectional Recurrent Neural Network (BRNN)

Bidirectional recurrent neural networks are just concatenating two independent RNNs. While the input sequence is fed in both positive and negative directions through the network, the outputs are usually summed or concatenated at each time step.

This configuration allows the networks to have backward as well as forward information about the sequence at every step. They're used for making predictions over a sequence with both past and future context. With this form of generative deep learning, the output layer can obtain information from the past (backward) and future (forward) states simultaneously.

Invented in 1997 by Schuster and Paliwal, BRNNs were introduced to multiply the amount of input information available to the network. Standard recurrent neural network (RNNs) have restrictions as the future input information cannot be reached from the current state. On the other hand, BRNNs do not require their input data to be fixed and their future input information is reachable from the current state.

BRNN is especially convenient when the context of the input is needed. For example, in the application of handwriting recognition, the performance of the model can be enhanced by predicting the upcoming alphabets and by knowledge of previous alphabets with respect to the current alphabets.

Figure 3.8 displays a typical structure of a Bidirectional Recurrent Neural Network formed by combining two Recurrent Neural Networks in both positive and negative axes. The input words from the text are denoted as x_0, x_1, x_2 … and predicted next words denoted as y_0, y_1, y_2 …The information for the previous input words denoted as s_0, s_1…s_i in the positive direction and as s_0', s_1',s_2'..s_i' in the negative direction.

3.3.1 Architecture and Working of BRNN

The principle of BRNN is to split the neurons of a regular RNN in two directions, one for positive time direction (forward states), and the other in negative time direction (backward states). Those two states' output is left disconnected to inputs of the opposite direction states. The general structure of RNN and BRNN can be depicted in the diagram. By using two-time directions, input information from the past and future of the current time frame can be used unlike the standard RNN, which requires delays for including future information.

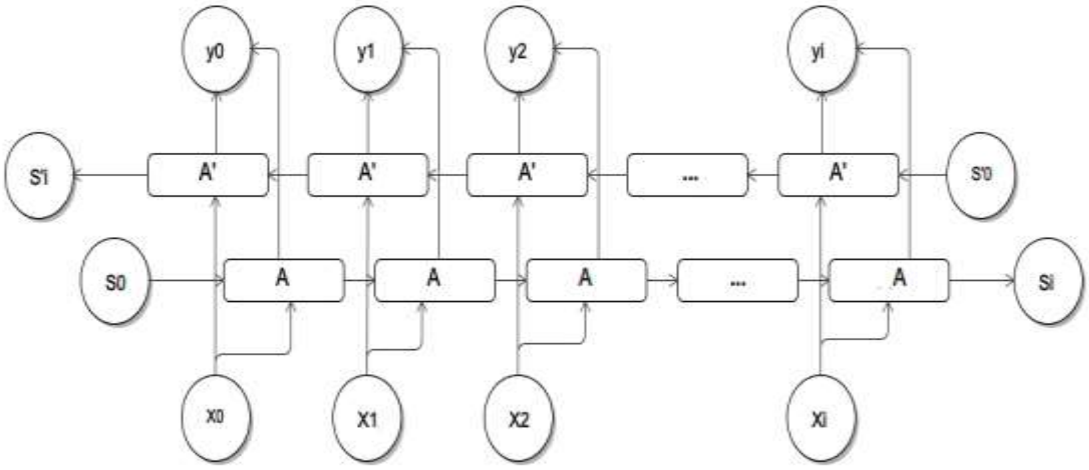


Figure 3. 8: Bidirectional Recurrent Neural Network

BRNNs can be trained using identical algorithms to RNNs because the two directional neurons do not have any interactions. When back-propagation is applied, additional processes are needed because updating input and output layers cannot be done at once.

Summary of Chapter: In this chapter, all important concepts related to Deep Learning and building a chatbot are discussed in detail. The classification of machine learning categories (Rube-based and Statistical Machine Translation) models are discussed. The Architectures of Recurrent and Bidirectional Recurrent Neural Networks, Long-shot Term memory network, Feed-forward neural networks are demonstrated and explained.

Chapter 4

Architecture of Chatbot and Software Implementation

Conversational agents are software programs that support communication between humans and machines in natural language. They assist, organize, manage and perform a user’s day-to-day tasks like participating in meetings and conferences, booking airline tickets, provide weather or traffic updates, resolve queries with customer care service and so on. Generally, a text-based conversational agent is called a chatbot. Chatbots now have evolved from basic texts to Voice Recognition and respond just as humans do. Messaging applications such as Facebook Messenger, WeChat, and Telegram are widely used by millions of people to communicate with their friends and colleagues. The ever-growing trendiness of messaging platforms is driving a new era of interaction with chatbots.

4.1. Working Procedure of a Typical Chatbot

The process of the system is simple and is designed in two stages. A high-level description is given below.

1. Request Stage
2. Response or Action Stage

To understand both stages, consider the example of a user requesting certain information from a chatbot;

- The user interacts with the chatbot through a user interface or channel (desktops or mobile applications).

- The chatbot receives the request from a user which is commonly inferred from the context to understand the intention of users request and the associated information is then processed using Natural Language Processing (NLP).

- The user interacts with the chatbot through a user interface (desktops, mobile applications).
- Once the input is understood, the chatbot is then triggered to perform some sort of action or by sending a response back to the user.
- Dialogue manager keeps the state of the conversation to know if the request is related to the previous conversation or if the conversation has switched over to a new topic.
- If the chatbot is informational, the required information is retrieved from its data sources. If the chatbot is conversational, the required response is retrieved from the data sources or with the chatbot's own strategy to answer the user. If the chatbot is designed for transactional purposes, then further actions are executed by the task manager.
- Task manager executes the action which the user asked to perform or retrieves the required information from the data sources to provide to the user. After this, a response message is generated by a response generation component and sent back to the user. This can be better understood with the system design of the chatbot model depicted in the next section.

4.2. System Design and Working Procedure

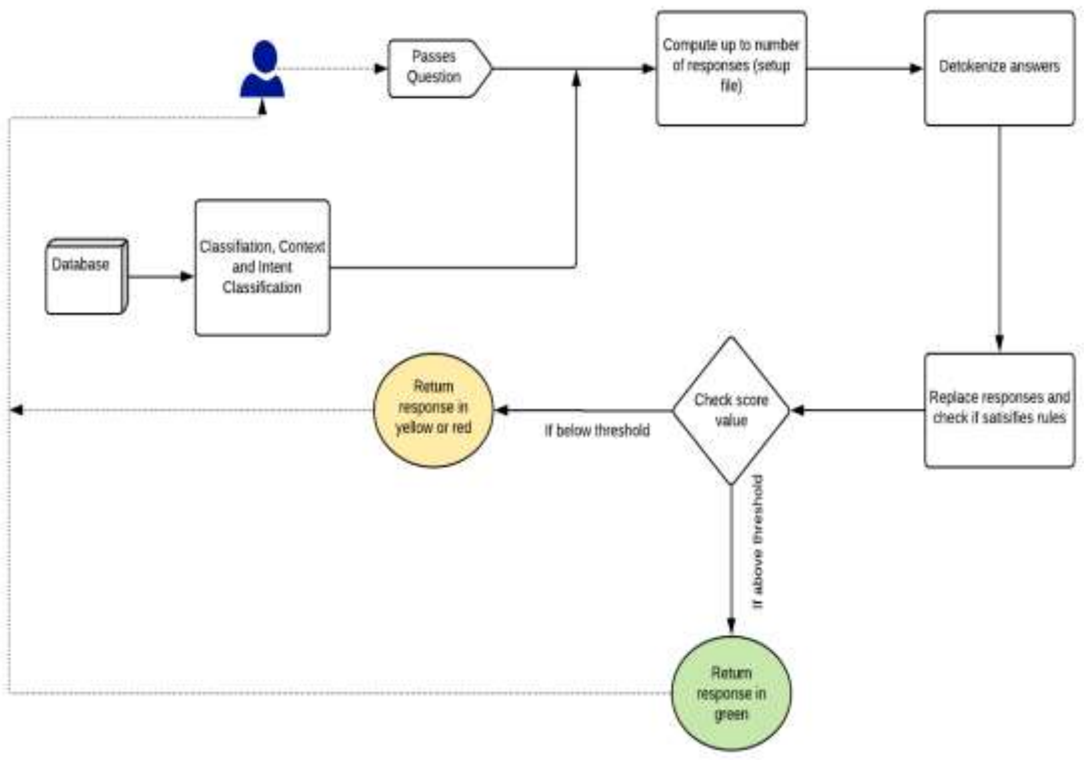

Passes Question
Compute up to number of responses (setup file)
Detokenize answers
Database
Classifiation, Context and Intent Classification
Return response in yellow or red
If below threshold
Check score value
Replace responses and check if satisfies rules
If above threshold
Return response in green

Figure 4. 1: System Design of the chatbot

The figure illustrates the process of system flow through the chatbot. The chatbot can generate responses from scratch based on the type of machine learning models or use certain heuristic approaches to select a response from a library of predefined responses. Since the model here is based on deep learning statistical approach, the models are usually harder to train and build.

The user first lands on the webpage and clicks on the chat widget which opens up a small chat window that enables interaction with the chatbot. In this scenario, let us consider that the user requires some information from the website such as the contact number of customer care. The user passes a question to the chatbot requesting for this information. After understanding the context and pulling up the desired response from the database, the desired output is retrieved. The chatbot model then computes up-to-the number of responses from the setup files. Next, this retrieved output is detokenized by passing through the rules file and the output is replaced, or certain parts are replaced accordingly. The final step is to pass this output through a decision block to check if it successfully meets the threshold value of the predefined score. If the value is above the threshold, then it returns the output response with a green highlight. If it below the recommended score, it returns the output response in yellow or red. This is implemented in order to show how the model rates certain responses from the model.

4.3. Software Architecture of Model

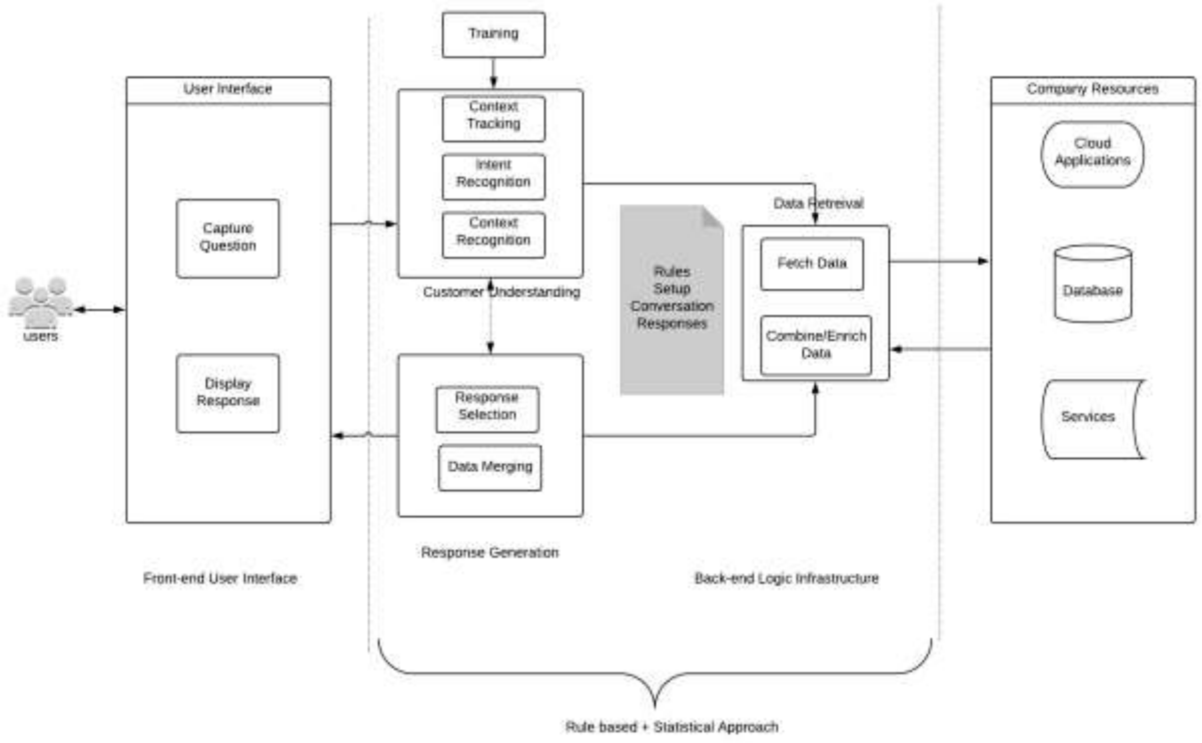


Figure 4. 2: Architecture of Chabot

To effectively describe the architecture of the model, it is divided into two phases. The first being the Front-end user interface and second, being the Back-end Logic Infrastructure.

Front-end User Interface:

The first phase involves the user interacting with the chatbot with the help of a medium. The chatbots are integrated usually on websites or as stand-alone applications on mobile web applications. Most popularly, they are integrated into messaging applications such as Skype or Messenger. The front-end user interface takes the input from the user and after processing this input, returns an output response usually through the same user interface. The user experience of the chatbot solely depends on the type of medium selected for the purpose of interaction. Examples of front-end user interfaces can be mobile applications or computers.

The front-end user interface will capture the user's question or query from the website or from the mobile platform, integrated on iOS or android applications.

- A site is loaded in browser from the server. Responsive front-end design allows a site to adapt to the user's device.
- Client-side scripts: Run in the browser and process requests without call-backs to the server. Everything a user sees in the browser in a mix of HTML, CSS and JavaScript.
- When a call to the database is required, JavaScript and AJAX send requests to the back end.
- The back-end server-side scripts process the request, pull what they need from the database then send it back.
- Server-side scripts process the data, then update the site – populating drop-down menus, loading products to a page, updating a user profile, and more.

Back-end User Interface:

- Frameworks are libraries of server-side programming languages that construct the back-end structure of a site.
- The 'STACK' comprises the database, server-side framework, server, and operating system (OS).
- API'S structure how data is exchanged between a database and any software accessing it.

Most websites make use of a small widget on their homepage, which is usually present on the bottom right corner of the screen. If the user clicks on this widget, it opens up a chat window and the user then initiate the interaction with the chatbot or virtual assistant. Some websites enable the chatbot itself to initiate a conversation whenever a user reaches their homepage.

Once the input is received from any of the above-mentioned mediums through the front-end user interface, the statement then passes through a module, here defined as ‘Customer Understanding.’ This module is concatenated with the pre-trained database of example queries and their corresponding responses. In this module, the input is processed through various stages involving Context tracking, Intent recognition, and Entity recognition.

If the incoming request requires third-party integrations or information retrieval from company resources such as cloud, external database or services, this is linked to the Customer Understanding module through a ‘Data Retrieval’ module.

If the data which is retrieved from these external integrations do not meet the requirements (score is below threshold value), then the output is enriched or tweaked in the same Data Retrieval module. The outgoing statement is then made to pass through ‘Response Generation’ module, which contains Response selection as well as Data merging.

Both the Customer Understanding and Response Generation modules are connected to a central database that has a pre-trained or pre-defined database of examples of Intents, Entities, Conversations and Responses from previous interactions. This is extremely useful for chatbots used to perform a specific set of applications because it possesses a much quicker response rate. The outgoing statement then returns back to the front-end user interface and displays the response to the user.

4.4. Logical Chatbot Design and Functionality

The step-by-step implementation of the chatbot developed is illustrated in the figure below.

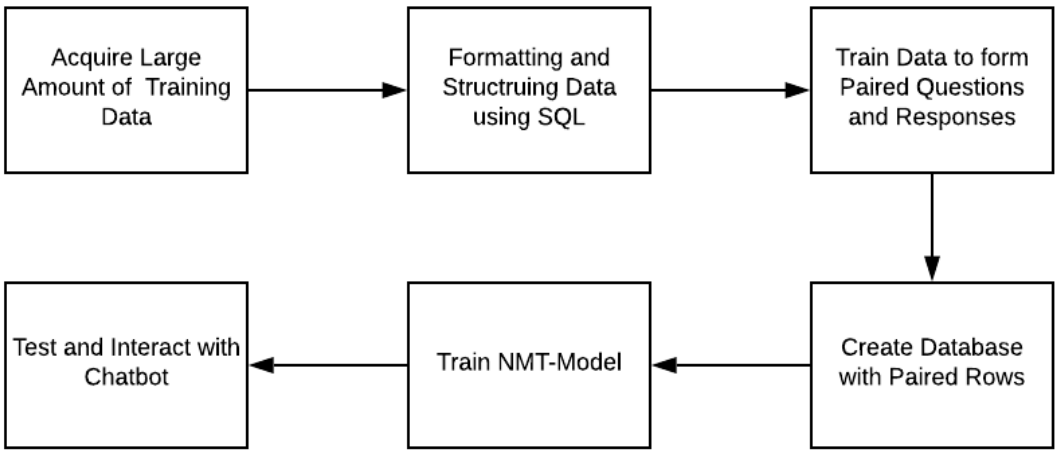

Acquire Large Amount of Training Data
Formatting and Structruing Data using SQL
Train Data to form Paired Questions and Responses
Create Database with Paired Rows
Train NMT-Model
Test and Interact with Chatbot

Figure 4. 3: Step-by-Step Implementation of Chatbot

Programming Language and Libraries:

Taking into account the context of the project and the company's software environment, the programming language Python is chosen for software implementation. The main reason for this is the ease of implementation due to the language's prolific machine learning dictionary. It is very easy to grasp and understand for future improvisations or feature enhancements.

The libraries imported in Python are mainly Pandas and TensorFlow. The first one is Pandas library whose overall goal is to provide a common and easy-to-use interface for several deep learning frameworks. Another important benefit of using Pandas is to efficiently store and organize datasets since it is useful for manipulating large amounts of data.

TensorFlow is a symbolic math library mainly used for Machine learning applications such as Neural networks that deals mainly with organizing and structuring the dataflow and for differentiable programming. Pandas and TensorFlow have been further discussed in the practical part of this report.

4.4.1 Class Diagram

The class diagram demonstrated below explains all the inter-connected modules with the conversational agent.

There are several chatbot components connected, each one performing a specific task. At the top of the Class diagram is the Client which is a generic client program that makes use of the chatbot. The chatbot's five main components are Intent Classifier, Action Architect, Context Manager, and Action Executor. The Action Planner has access to two submodules, one of which (Backend Interface) is shared with another submodule.

Intent Assortment:
The user's intent can be determined or predicted using the Intent Assortment. It relies heavily on artificial neural networks to perform its function. The input is the user message, and the output is a dictionary that maps every supported user intent with its corresponding probability for that particular input.

Context Manager:
Handling the structure to organize contexts is the role of the Context Manager. A context, in this case, is defined as the sequence of messages, as well as metadata attached to them.

Action Architect:
Given a probability vector, decide which actions the agent should consider and perform is the top-most goal of the Architect. Action architect determines the actual reply to be sent to the user and relies on two submodules: Locales Handler and the backend interface.

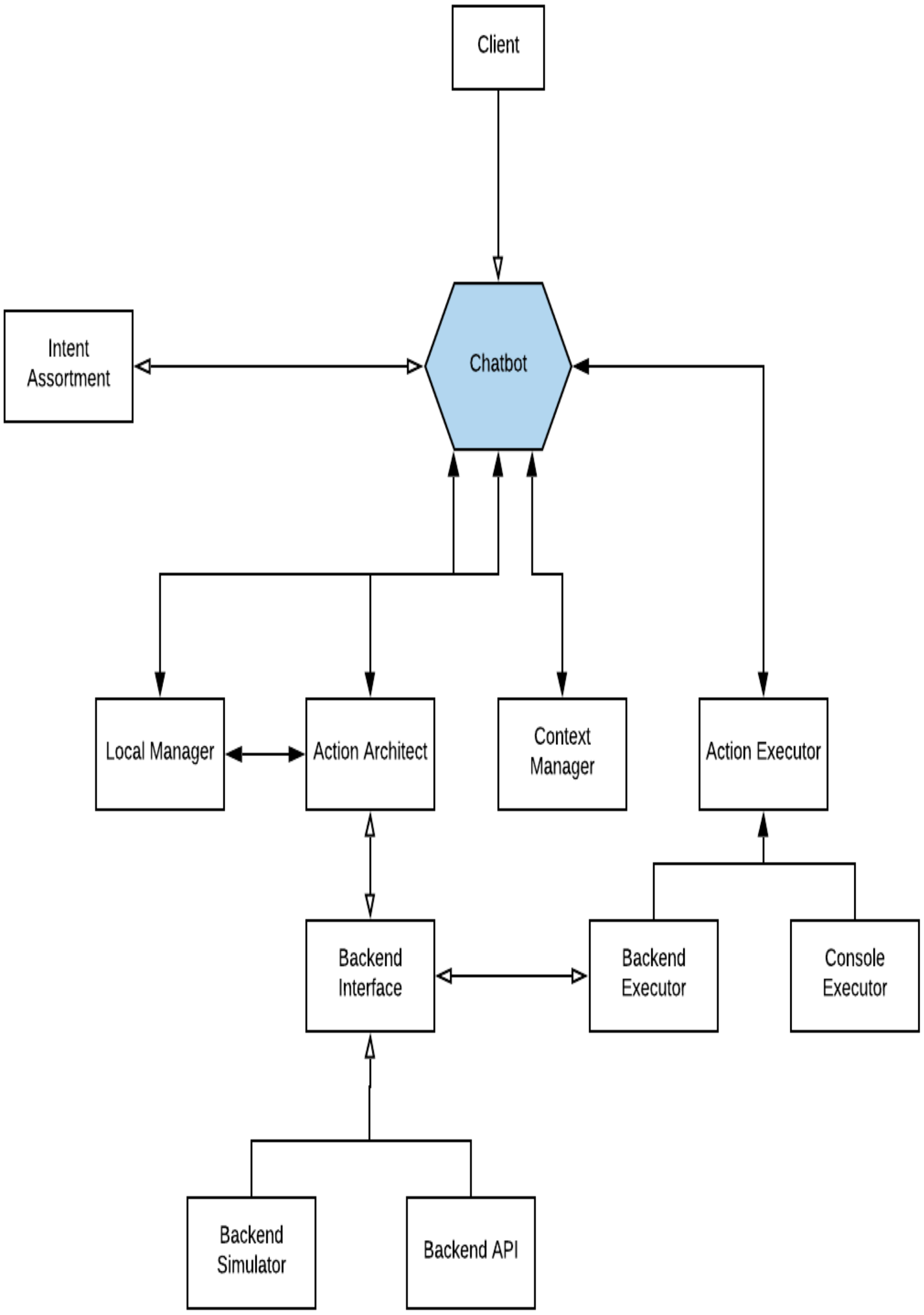


Figure 4. 4: Class Diagram of the Designed Model

Locales Manager:
Enable the code without having the need to localize every constant or string that appears in the codebase.

Backend Interface:
The backend interface is flexible and simple interface class over the system's database that specifies which functions are available to the agent. It adds versatility to the chatbot module.

Action Executor:
Action Executor is responsible for executing sequences of instructions. It works directly on the Action planner's output, sends the messages and performs the planned instructions on the backend.

Summary of Chapter: In Chapter 4, the core architectural concepts of the chatbot model is explained with reference to Neural Machine Translation (NMT). The system diagram with the step-by-step process flow is illustrated and explained in the subsections of this chapter. The class diagram and software framework, as well as the module implementation, are explained, thereby covering all theoretical concepts in chatbot development.

PART II

PRACTICAL

Chapter 5

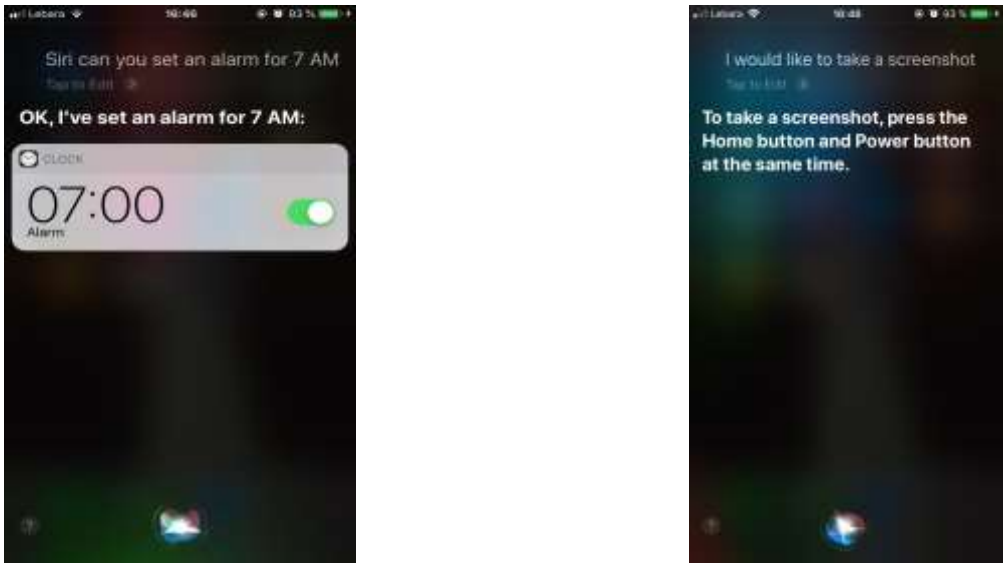
Building Database and Inserting Logic

Based on the type of application we want to model, we can design two types of chatbots as per our requirements. As explained in the introduction, chatbots can be either hold open-ended conversations or have limited-scope of conversations.

Rule-based bots are more task-oriented or classical in their approach. They are limited in their conversational capabilities and are unable to respond to arbitrary utterances. They are deployed in scenarios which require pulling or locating data from the database or knowledge base. The advantage of such models is that they are usually very accurate at executing task-specific commands and are robust in architecture. Domain-specific chatbot models have a finite database and usually rely on machine learning models. They do not require deep learning networks as the trained data need not be perpetually updated or expanded.

Chatbots which can hold open-ended conversations are currently deployed on Apple’s Siri, Amazon’s Alexa and Google Home. These chatbots or dialog agents have a wider dictionary and serves wider domains. They are integrated with many internal and external capabilities or functionalities to execute various commands from the user. Live examples from Apple’s Siri is demonstrated below in Figure 5.1 in two scenarios. In scenario (a), the User requests the chatbot to perform a certain task such as setting up an alarm. In scenario (b), the user is requesting some information which the virtual assistant retrieves.

Open-ended chatbots try to imitate human dialog is all its facets. They are designed in such a way that humans find it hard to distinguish if they are conversing with a machine or a human being. Their speech is incorporated with empathy, mimicking a human’s emotional quotient (EQ). These models don’t possess a well-defined goal, but they are required to have a certain amount of general knowledge and common-sense reasoning capabilities in order to hold conversations about any topic.


Siri can you set an alarm for 7 AM

Tap to Edit

OK, I've set an alarm for 7 AM:

CLOCK

07:00

Alarm

I would like to take a screenshot

Tap to Edit

To take a screenshot, press the Home button and Power button at the same time.

(a) (b)

(a): Screenshot of asking a bot to set an alarm

(b): Screenshot of requesting some information from the robot

Figure 5. 1: User requesting Siri to perform tasks

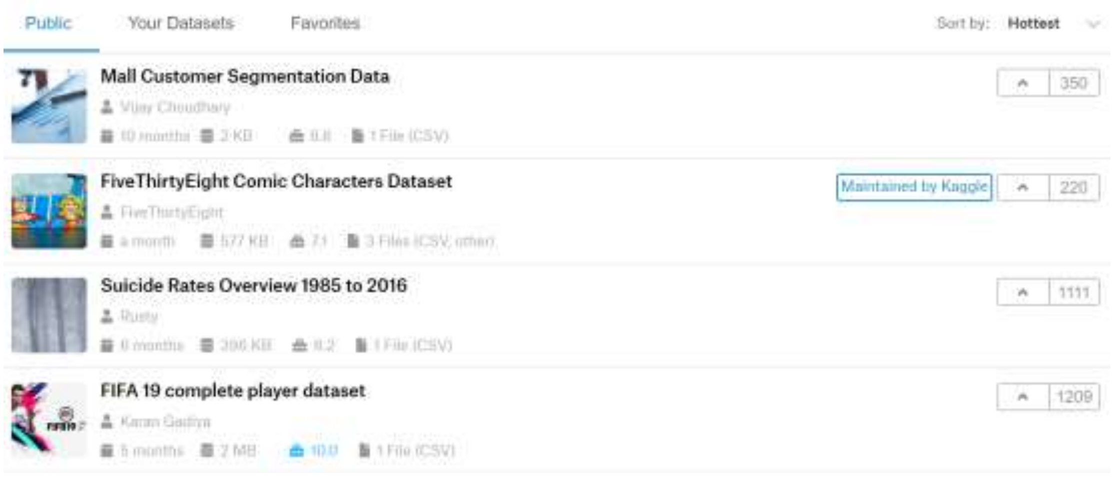
The goal is to combine the positive aspects of both these types of conversational agents to design a hybrid model. The robust abilities of task-oriented dialog systems and the human-like chattiness of open-domain chatbots. This is beneficial because the user is more likely to engage with a task-oriented dialog agent if it’s more natural and handles out of domain responses well.

Since the overall focus is on the development of a generic chatbot that can hold open-ended conversations, it is crucial to acquire large amounts of data from which the model can learn or train. The larger the dataset for training, the wider the vocabulary of the chatbot. It is also crucial to consider the quality of the data that the chatbot is getting trained on. Higher quality data produces better results or responses.

5.1. Acquisition of Training Data

Large datasets are usually noisy, as is the case with the OpenSubtitles dataset and many other datasets available online. The other prevalent type of dataset is based on message boards and post-reply websites, like the Ubuntu dialog corpus, Twitter or Reddit. The underlying problem with these datasets is that they don’t actually have one on one conversations. These conversations are public which are not as natural as private ones.

Some examples of online and free datasets that can be used for training the chatbot are given below in Figure 5.2. Figure 5.2 (a) shows the corpus from Kaggle, where datasets are sorted out with different topics. Figure 5.2 (b) shows the dataset available from the UCI Machine Learning Repository which can be further classified by instances, attributes and year.

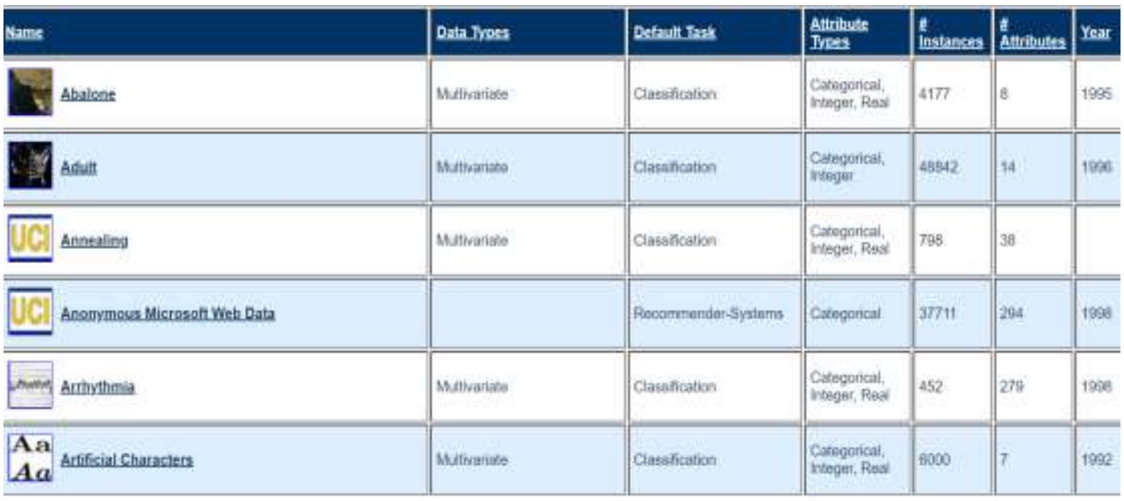


Public | Your Datasets | Favorites | Sort by: Hottest

Mall Customer Segmentation Data Vijay Choudhary 10 months · 3 KB · 8.8 · 1 File (CSV)		350
FiveThirtyEight Comic Characters Dataset FiveThirtyEight a month · 577 KB · 7.1 · 3 Files (CSV, other)	Maintained by Kaggle	220
Suicide Rates Overview 1985 to 2016 Rusty 6 months · 396 KB · 8.2 · 1 File (CSV)		1111
FIFA 19 complete player dataset Karan Gadiya 5 months · 2 MB · 10.0 · 1 File (CSV)		1209

Source: *https://www.kaggle.com/datasets*

(a): Dataset from Kaggle



Name	Data Types	Default Task	Attribute Types	# Instances	# Attributes	Year
Abalone	Multivariate	Classification	Categorical, Integer, Real	4177	8	1995
Adult	Multivariate	Classification	Categorical, Integer	48842	14	1996
Annealing	Multivariate	Classification	Categorical, Integer, Real	798	38	
Anonymous Microsoft Web Data		Recommender-Systems	Categorical	37711	294	1998
Arrhythmia	Multivariate	Classification	Categorical, Integer, Real	452	279	1998
Artificial Characters	Multivariate	Classification	Categorical, Integer, Real	6000	7	1992

Source: *https://archive.ics.uci.edu/ml/datasets.php*

(b): Datasets from UCI Machine Learning Repository

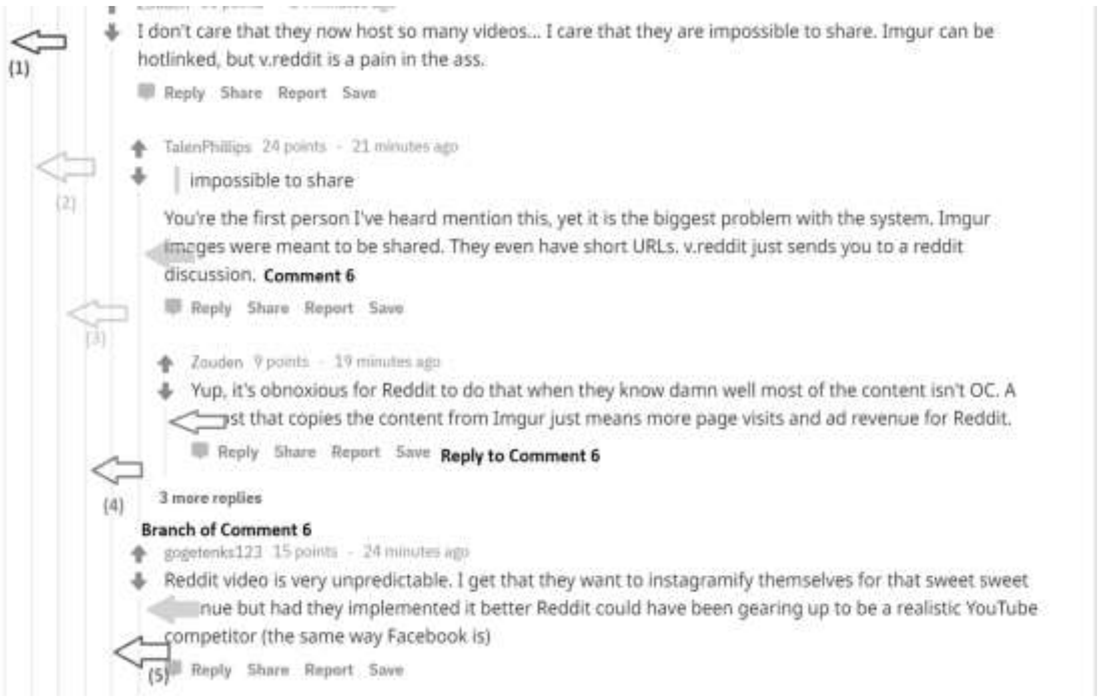
Figure 5. 2: Online Dataset Repositories

Since we require a chatbot that has more raw responses, we use Reddit to create the dataset for training data. Basically, Reddit is a social news aggregation, web content rating, and discussion website. The members submit content to the site such as images, links, texts or messages, which are upvoted or downvoted by other members. Posts are arranged and filtered by subject into user-created boards called 'subreddits', which range from a variety of topics including news, travel, science, food, movies, video games, music, books, and fitness.

The first practical task in the development of a chatbot is acquiring enormous amounts of training data. This data is then structured and formatted in an 'input' and 'output' manner that a machine learning algorithm can digest. The structure of Reddit data is in a tree-form. For the chatbot design, we require a linear data structure. The parent comments are linear but replies to parent comments branch out.

A user of Reddit usually referred to as ‘Redditor’, makes a post and other Redditors comment on the post. There can be different combinations for a particular post:

- Comment with a reply
- Comment with zero (none) replies
- Comment with a reply that also has a reply (branching of comments)



I don't care that they now host so many videos... I care that they are impossible to share. Imgur can be hotlinked, but v.reddit is a pain in the ass.

Reply Share Report Save

TalenPhillips 24 points - 21 minutes ago

> impossible to share

You're the first person I've heard mention this, yet it is the biggest problem with the system. Imgur images were meant to be shared. They even have short URLs. v.reddit just sends you to a reddit discussion. **Comment 6**

Reply Share Report Save

Zouden 9 points - 19 minutes ago

Yup, it's obnoxious for Reddit to do that when they know damn well most of the content isn't OC. A post that copies the content from Imgur just means more page visits and ad revenue for Reddit.

Reply Share Report Save **Reply to Comment 6**

3 more replies

Branch of Comment 6

gogetenks123 15 points - 24 minutes ago

Reddit video is very unpredictable. I get that they want to instagramify themselves for that sweet sweet revenue but had they implemented it better Reddit could have been gearing up to be a realistic YouTube competitor (the same way Facebook is)

Reply Share Report Save

Figure 5. 3: Reddit Parent and Comment pairs

The markings (1) indicates the parent post, while (2), (3), (4) and (5) are comments to the parent post, here referred to as child post. The shaded arrows indicated are sub-replies to the child post, indicating that child post contains a sub-child post.

The dataset used from Reddit contains approximately 54 million rows and about 1.65 billion comments, which is a reasonable amount of data to train and build a decent chatbot.

5.2. Formatting Reddit Data

An example of a typical comment data is given below from which the parent id, comment id, the parent comment, the reply (comment), subreddit, the time, and score (votes) for the comment is extracted.

{"author":"Jane","link_id":"t3_5yba3","score":1,"body":"Can we please deprecate the word \"Ajax\" now? \r\n\r\n(Isnt _this_ much nicer?)","score_hidden":false,"author_flair_text":null,"gilded":1,"subreddit":"reddit.com","edited":false,"author_flair_css_class":null,"retrieved_on":14276409,"name":"t1_c02998ap","created_utc":"119245066643","parent_id":"t1_c02999p","controversiality":0,"ups":4,"distinguished":null,"id":"c0299ap","subreddit_id":"t5_6","downs":0,"archived":false }

5.2.1 Creating an SQLite Database

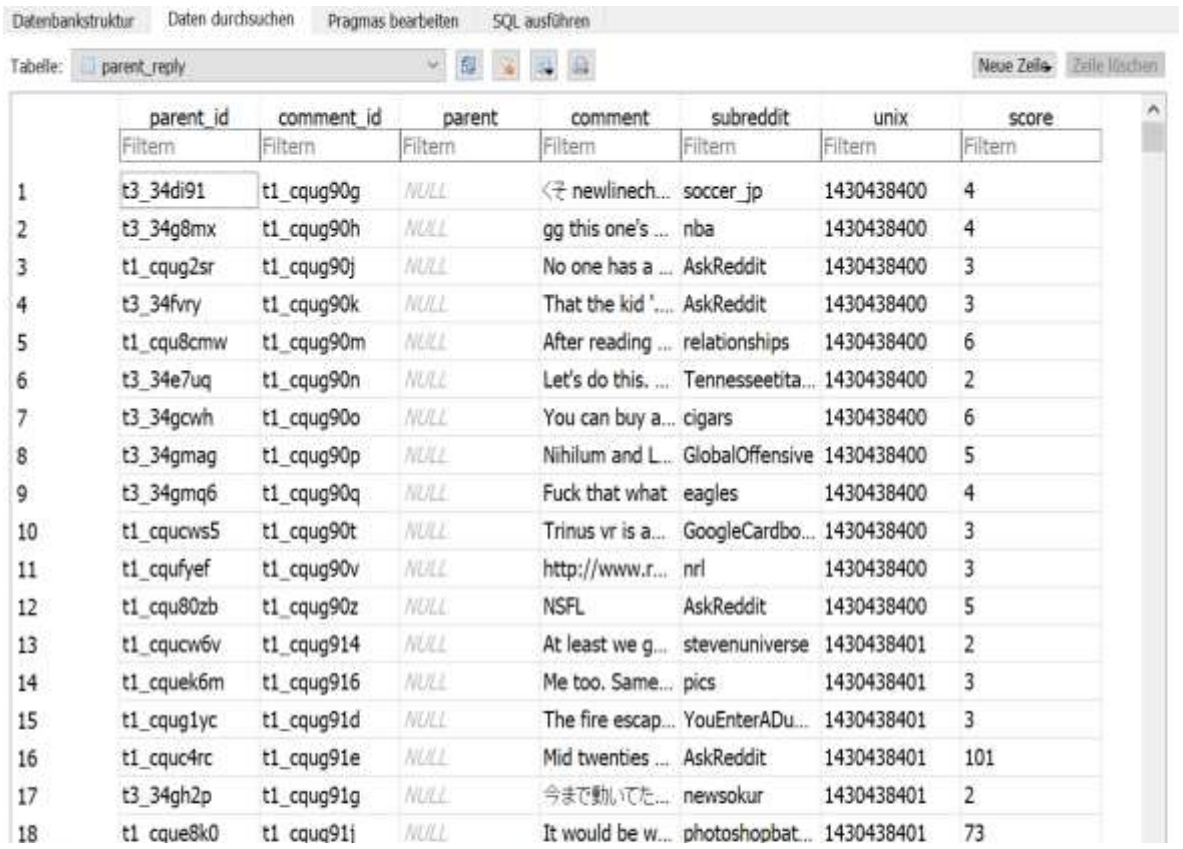
SQLite is a lite version of the SQL database. SQLite3 comes integrated in Python-3 standard library. Databases offer a superior method of high-volume data input and output over a typical file such as a text file. SQL is also a programming language in itself and a popular database language. For this reason, many websites use MySQL.

Setting up an SQLite database is nearly instantaneous. There is no requirement of setting up servers, permissions and as a result of this, it is often used as a developmental and prototyping database that can be used in production.

The main concern with SQLite is that it winds up being much like any other flat-file, so high volume input or output, especially with simultaneous queries, can be problematic and slow.

On the other hand, SQLite structures the data as a database which can easily be queried. Buffering through data in the database is seemingly quicker and easier. Also, edits do not require the entire file to be re-saved. This improves the performance of large databases significantly. The following script is used to create the database locally.

```
import sqlite3
import json
from datetime import datetime
timeframe = '2019-05'
sql_transaction = []
connection = sqlite3.connect('{}2.db'.format(timeframe))
c = connection.cursor()
def create_table():
   c.execute("CREATE TABLE IF NOT EXISTS parent_reply(parent_id TEXT PRIMARY KEY, comment_id TEXT UNIQUE, parent TEXT, comment TEXT, subreddit TEXT, unix INT, score INT)")
if __name__ == '__main__':
   create_table()
```



	parent_id	comment_id	parent	comment	subreddit	unix	score
	Filtern	Filtern	Filtern	Filtern	Filtern	Filtern	Filtern
1	t3_34di91	t1_cqug90g	NULL	くそ newlinech...	soccer_jp	1430438400	4
2	t3_34g8mx	t1_cqug90h	NULL	gg this one's ...	nba	1430438400	4
3	t1_cqug2sr	t1_cqug90j	NULL	No one has a ...	AskReddit	1430438400	3
4	t3_34fvry	t1_cqug90k	NULL	That the kid '....	AskReddit	1430438400	3
5	t1_cqu8cmw	t1_cqug90m	NULL	After reading ...	relationships	1430438400	6
6	t3_34e7uq	t1_cqug90n	NULL	Let's do this. ...	Tennesseetita...	1430438400	2
7	t3_34gcwh	t1_cqug90o	NULL	You can buy a...	cigars	1430438400	6
8	t3_34gmag	t1_cqug90p	NULL	Nihilum and L...	GlobalOffensive	1430438400	5
9	t3_34gmq6	t1_cqug90q	NULL	Fuck that what	eagles	1430438400	4
10	t1_cqucws5	t1_cqug90t	NULL	Trinus vr is a...	GoogleCardbo...	1430438400	3
11	t1_cqufyef	t1_cqug90v	NULL	http://www.r...	nrl	1430438400	3
12	t1_cqu80zb	t1_cqug90z	NULL	NSFL	AskReddit	1430438400	5
13	t1_cqucw6v	t1_cqug914	NULL	At least we g...	stevenuniverse	1430438401	2
14	t1_cquek6m	t1_cqug916	NULL	Me too. Same...	pics	1430438401	3
15	t1_cqug1yc	t1_cqug91d	NULL	The fire escap...	YouEnterADu...	1430438401	3
16	t1_cquc4rc	t1_cqug91e	NULL	Mid twenties ...	AskReddit	1430438401	101
17	t3_34gh2p	t1_cqug91g	NULL	今まで動いてた...	newsokur	1430438401	2
18	t1_cque8k0	t1_cqug91j	NULL	It would be w...	photoshopbat...	1430438401	73

Figure 5. 4: SQLite Database created with training data

5.2.2 Buffering through Data

JavaScript Object Notation is used for reading and buffering through the data. JSON (JavaScript Object Notation) is a lightweight data-exchange format, which is easy for humans to read and write, and easy for computers to parse and generate. Basically, it is an independent language and is in text format and uses conventions that are familiar to programmers of the C-family of languages, such as Perl, C, C++, C#, Java and so on. These properties make JSON an ideal data-interchange language.

JSON is devised using two structures:

- Set of name or value pairs - In various languages, this is realized as an associative array, object, record, struct, dictionary, hash table, keyed list.

- An ordered list of values - In most languages, this is realized as a sequence, array, vector, list.

When exchanging data between a server and a browser, the data can be a text. JSON is text; it is possible to convert any JavaScript object into JSON and send JSON to the server. It is also convenient to convert any JSON received from the server into JavaScript objects. JavaScript objects are easy to work with as they do not require complicated parsing and translations.

5.3. Inserting Logic into the Data

Inserting Logic into the structure of data is done in order to sanitize or clean up the data dump, in order to receive more useful and accurate responses. For example, if a certain reply comment has been deleted, then it is necessary to block this statement or delete it altogether from the database so that the chatbot doesn't respond with a blank comment for that particular question. Below defined are a set of restrictions imposed on the data dump.

Restriction 1: Insert a restriction on all comments which do not have comment-reply pairs with minimum upvote of 2. A method for adding this restriction is given below.

```
if score >= 2:
existing_comment_score = find_existing_score(parent_id)
```

Restriction 2: Check if there's an existing reply to the parent and its corresponding score. If there is already an existing comment, and if the score is higher than the existing comment's score, then this must be replaced.

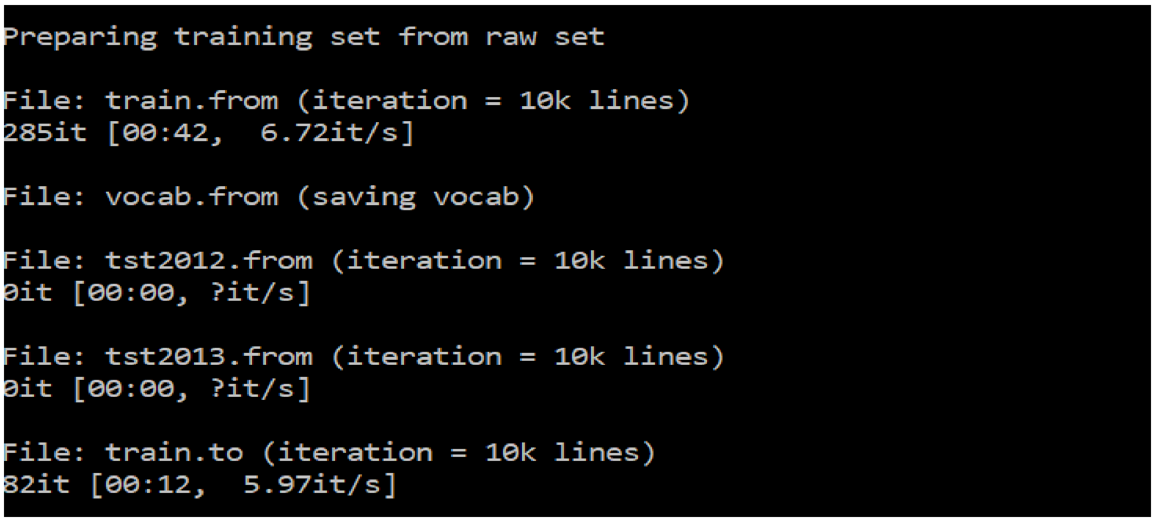
Restriction 3: Check if the comment has a maximum length of 50 words (bucketing) and a minimum length of 1 word. Many comments are either deleted or removed and some comments are very long or very short. It is important to make sure comments are of an acceptable length for training.

Restriction 4: If the data or comment has been deleted or removed, then this parent-reply pair is also deleted from the database.

For all the above restrictions, the code is set to return 'False', hence blocking from adding these on to the tables in the database. The above restrictions are implemented in the code as shown below.

```
def acceptable(data):
    if len(data.split(' ')) > 50 or len(data) < 1:
        return False
    elif len(data) > 1000:
        return False
    elif data == '[deleted]':
        return False
    elif data == '[removed]':
        return False
    else:
        return True
```

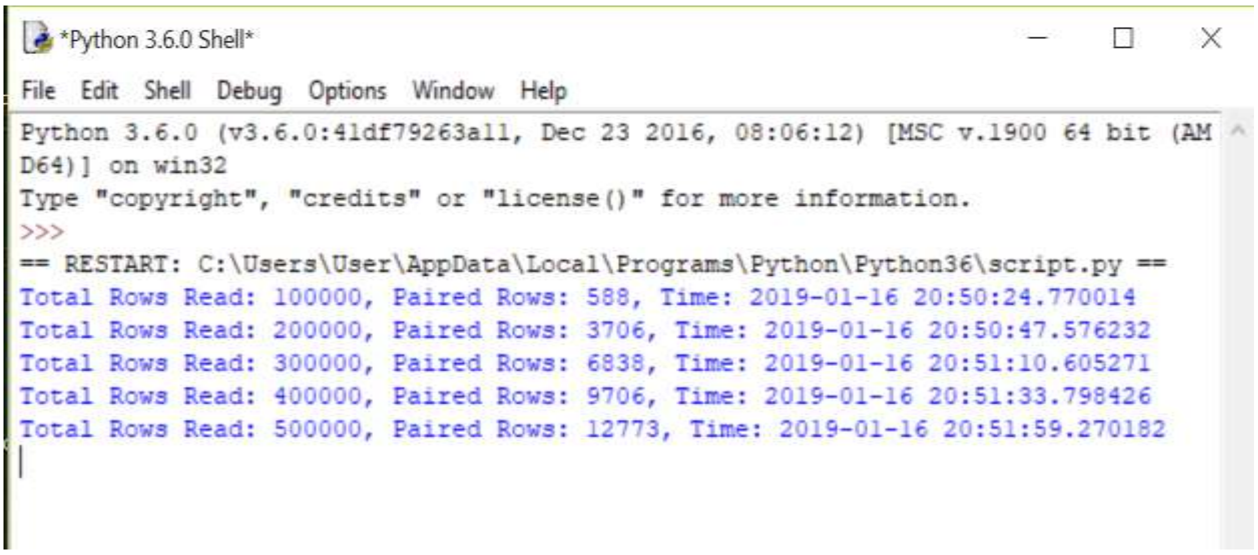
Once all the restrictions have been defined, the data is ready to be built or pulled into rows into the SQLite database. Upon debugging and running, the code will first prepare the training set from the raw dataset and will start iterating for every 10,000 lines (as defined).



```
Preparing training set from raw set

File: train.from (iteration = 10k lines)
285it [00:42,  6.72it/s]

File: vocab.from (saving vocab)

File: tst2012.from (iteration = 10k lines)
0it [00:00, ?it/s]

File: tst2013.from (iteration = 10k lines)
0it [00:00, ?it/s]

File: train.to (iteration = 10k lines)
82it [00:12,  5.97it/s]
```

Figure 5. 5: Initializing Training Data

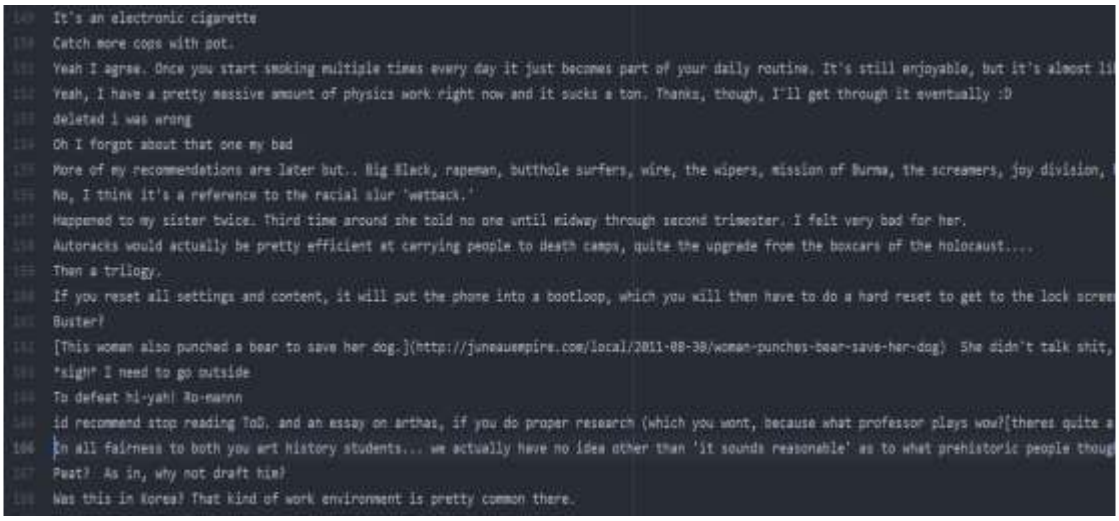
After the dataset has been initialized, the code begins to start appending the rows of the database with information. For every 10,000 rows as shown below, only about 588 rows have been paired. This is because of the imposed restrictions, either the score value is too low, or the comment reply is too long or short, or it has been deleted. The number of paired rows are the ones that will result in the actual question-response pairs and will be used to train the chatbot model.



```
Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 08:06:12) [MSC v.1900 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
== RESTART: C:\Users\User\AppData\Local\Programs\Python\Python36\script.py ==
Total Rows Read: 100000, Paired Rows: 588, Time: 2019-01-16 20:50:24.770014
Total Rows Read: 200000, Paired Rows: 3706, Time: 2019-01-16 20:50:47.576232
Total Rows Read: 300000, Paired Rows: 6838, Time: 2019-01-16 20:51:10.605271
Total Rows Read: 400000, Paired Rows: 9706, Time: 2019-01-16 20:51:33.798426
Total Rows Read: 500000, Paired Rows: 12773, Time: 2019-01-16 20:51:59.270182
```

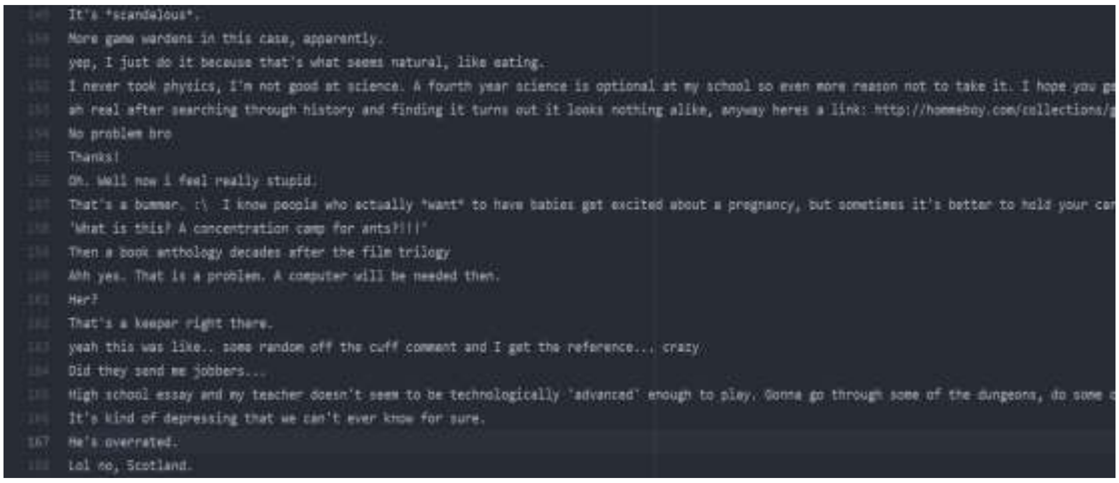
Figure 5. 6: Training Data & Forming Paired Rows

After the data has been trained, the paired rows are used to form the Parent and Reply files as shown below. From the Reddit data dump used in this model, the total number of paired rows are 2843673 parent and reply comments. The more amount of data available to train the chatbot, the wider the chatbot’s dictionary tends to be. The chatbot can also then serve a wider domain and answer a variety of questions.



```
It's an electronic cigarette
Catch more cops with pot.
Yeah I agree. Once you start smoking multiple times every day it just becomes part of your daily routine. It's still enjoyable, but it's almost li
Yeah, I have a pretty massive amount of physics work right now and it sucks a ton. Thanks, though, I'll get through it eventually :D
deleted i was wrong
Oh I forgot about that one my bad
More of my recommendations are later but.. Big Black, rapeman, butthole surfers, wire, the wipers, mission of Burma, the screamers, joy division,
No, I think it's a reference to the racial slur 'wetback.'
Happened to my sister twice. Third time around she told no one until midway through second trimester. I felt very bad for her.
Autoracks would actually be pretty efficient at carrying people to death camps, quite the upgrade from the boxcars of the holocaust....
Then a trilogy.
If you reset all settings and content, it will put the phone into a bootloop, which you will then have to do a hard reset to get to the lock scree
Buster?
[This woman also punched a bear to save her dog.](http://juneauempire.com/local/2011-08-30/woman-punches-bear-save-her-dog)  She didn't talk shit,
*sigh* I need to go outside
To defeat hi-yah! Ro-mannn
id recommend stop reading ToD. and an essay on arthas, if you do proper research (which you wont, because what professor plays wow?[theres quite a
In all fairness to both you art history students... we actually have no idea other than 'it sounds reasonable' as to what prehistoric people thoug
Peat?  As in, why not draft him?
Was this in Korea? That kind of work environment is pretty common there.
```

(a) Parent File



```
It's *scandalous*.
More game wardens in this case, apparently.
yep, I just do it because that's what seems natural, like eating.
I never took physics, I'm not good at science. A fourth year science is optional at my school so even more reason not to take it. I hope you ge
ah real after searching through history and finding it turns out it looks nothing alike, anyway heres a link: http://hommeboy.com/collections/g
No problem bro
Thanks!
Oh. Well now i feel really stupid.
That's a bummer. :\  I know people who actually *want* to have babies get excited about a pregnancy, but sometimes it's better to hold your car
'What is this? A concentration camp for ants?!!!'
Then a book anthology decades after the film trilogy
Ahh yes. That is a problem. A computer will be needed then.
Her?
That's a keeper right there.
yeah this was like.. some random off the cuff comment and I get the reference... crazy
Did they send me jobbers...
High school essay and my teacher doesn't seem to be technologically 'advanced' enough to play. Gonna go through some of the dungeons, do some c
It's kind of depressing that we can't ever know for sure.
He's overrated.
Lol no, Scotland.
```

(b) Comment File

Figure 5. 7: (a) & (b): Training files in the database

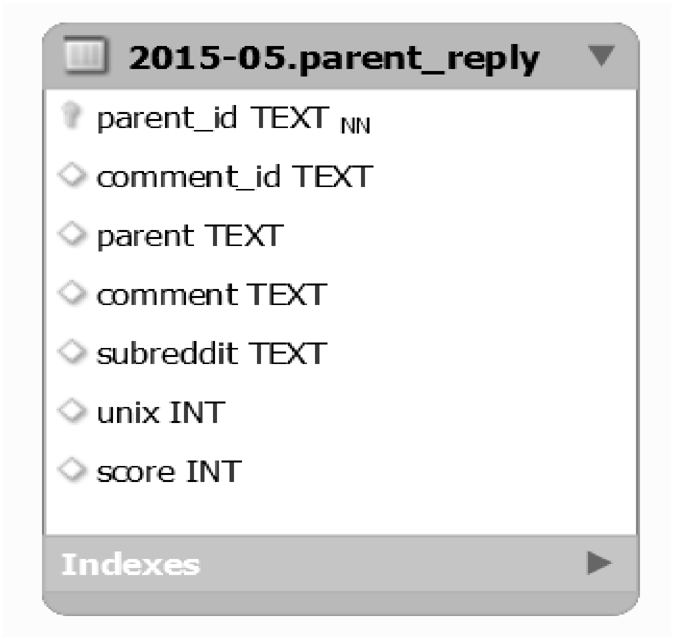
Summary of Chapter: In Chapter 5, we start with the practical implementation of the designed hybrid model, concentrating mainly on the acquisition of training data. This is by far the most important and difficult step because finding large datasets of good quality data, online is quite challenging. Once the data is obtained, it is formatted with the help of SQLite, which is used for creating the database. Logic is inserted into the data through which useful information can be retrieved and the rest of unnecessary data can be discarded, thus saving a considerable amount of space on the database.

Chapter 6

Training the Model and Deployment on website

Once the database with training data has been created, we obtain two files, one is the parent file and the other is the comment file. The parent file will be typical questions asked by the user and the comment files contain corresponding responses to those questions or queries. In layman’s terms, line 12 in the parent file is the parent comment, and then line 12 in the reply file is its corresponding response.

The database that is created at the end of the previous chapter is illustrated below.



```
2015-05.parent_reply
parent_id TEXT NN
comment_id TEXT
parent TEXT
comment TEXT
subreddit TEXT
unix INT
score INT
Indexes
```

Figure 6. 1: Database created with Training Data

6.1. Training the Model

In order to start training the dataset, we need to secure the pairs from the database and supplement them to the respective training files. The chatbot model has a sequence to sequence (seq2seq) architecture, which is a set of utilities built on top of TensorFlow's NMT code.

Consider a task like a sequence to sequence, where the sequences aren't exactly the same length. Here, two-word statements could yield fifteen-word responses, and long statements could return single-word responses. Each input varies from the previous one in terms of characters, words, and more. Words themselves will be assigned either meaningful or arbitrary word-ids (via word vectors).

One concern that arises with this type of model is handling of variable length sentences, for which one obvious solution is padding. Consider padding of all strings of words 50 words long (for example). Then, when statements are 35 words long, the remaining 15 can be padded. Any data longer than 50 words, can be discarded from training. This can make training hard, unfavorably, especially for shorter responses, which might be the most common, and most of the words will just be padding. Some older models used bucketing to solve this and trained with 4 buckets. 5-10, 10-15, 20-25, and 40-50 but this isn't quite ideal.

The best way to resolve this issue is to use the NMT code that works with variable inputs, without the requirement for bucketing or padding. The NMT code also contains support for attention mechanisms, which are an attempt at adding longer-term memory to recurrent neural networks.

In general, an LSTM can remember sequences of tokens up to 10 to 20 in length fairly well. Beyond this sequence length, the performance drops, and the network forgets the older tokens because the memory now contains the newer ones. In this case, tokens are words, so a basic LSTM should be capable of learning 10 to 20 word-length sentences. If the word-length is increased, the accuracy and efficiency of the model gradually decreases.

Attention mechanisms result in longer attention spans, which help a network to reach a greater number of words. Most of these hindrances are resolved by using Bidirectional Recurrent Neural Networks (BRNN). The bidirectional recurrent neural network (BRNN) suspects that current data, past data, and the future data present in the input sequence. The input sequence goes in both the positive and negative directions.

6.1.1 Using Pandas for Data Analysis

Pandas is an open source library for programming language - Python, which provides high-performance, user-friendly data structures and data analysis tools.

Pandas is compatible with different types of data:

- Tabular data with heterogeneously-typed columns, such as in an Excel spreadsheet.
- Unordered and ordered time series data representations.
- Any other type of unlabeled or labeled statistical or observational data sets.

Since the data contained in the database is also in tabular form, pandas can quickly analyze and manipulate this data as required. The following function is used to read from the database. The function reads the SQL statement and performs alterations on the data if it has been requested.

df = pd.read_sql

6.1.2 Training on Cloud and Virtual Machines

Recently, Amazon Web Services (AWS) and PaperSpace have received a lot of recognition for containing cloud-based platforms and servers to train large amounts of data and for performing high computations. PaperSpace contains a particular tool called ‘‘ML IN A BOX’’ which contains a powerful GPU-backed virtual machine along with dedicated servers for faster computations. While training the model on the cloud offers very high computational power and runs at much higher speeds, the prototype model discussed herein, has been trained locally on a personal computer for cost efficiency.

6.1.3 Tokenization, Detokenization & Word Vectors

There are two main techniques of word-tokenization’s that can be adopted for the chatbot model:

1. **Standard Tokenizer:** Standard tokenizer is based on Moses-smt one. It's a highly modified own python implementation. Standard tokenizer splits sentences by space, period (and other grammar chars), separates digits and so on. The designed hybrid model adopts the standard tokenizer.
    - Advantage – Lack of duplicates in vocab file.
    - Disadvantage – This model needs a bunch of regex-based rules for detokenization process and it's hard to write ones that cover all cases.
2. **BPE-WPM Tokenizer**: BPE-WPM like tokenizer is based on subword-nmt one mainly for speed. This tokenize type is very similar to the Standard Tokenizer, but it additionally splits every entity by character, and it counts most common pairs of characters and it joins that pairs to make a vocabulary of the desired number of tokens. In layman’s terms, it splits a word based on syllables.

As a result, most common words are joined back together, when the rarest ones will stay split into multiple pieces shared with other words.

- Advantage - Ability to fit any number of words (tokens) in much smaller vocab thanks to sub-words.

Detokenization: An Embedded detokenizer performs a very accurate detokenization at the exchange for a higher number of tokens in vocabulary. Detokenization is simple – it replaces two characters in the resulting sentence - first, remove all spaces, then replace meta-symbol '▁' with a space character.

- Advantage – Lack of any rules for the process.

Word Embeddings and Word Vectors:

Word embeddings are vectors (real-numbers) that represent words in a vocabulary. For example, the word ‘root’ can have an arbitrary value of 5 and the word ‘juice’ can have a value of 67. Word Vectors have a wide area of application, especially in Natural Language Processing and Speech Recognition. The core concept of word embeddings is that every word used within the vocabulary of a language can be represented by a set of real numbers. The word-meanings and their respective contexts are determined and are assigned N-dimensional vector values.

An example is taken into account to better understand the concept of Word Embeddings. If the word ‘kitchen’ is taken into account, it is noticeable on the chart that it is at close proximity to the words ‘sink’, ‘bathroom’, ‘table’ since they all come under the household category. If the word ‘charger’ is considered, then the distance between ‘kitchen’ and ‘charger’ is a lot more when compared to the distance between ‘charger’ and ‘drill.’ Word vectors are arranged in this way, with their closest matching pairs at closer proximities when compared to other words.

6.2. Determining the Score of Responses

Determining the value of the returned responses is vital if we have to understand how the chatbot is performing. The performance of the chatbot depends on certain factors such as BLUE score, Perplexity and Loss. For certain responses obtained from the model, the value of score can be pre-defined.

Consider chatting with a virtual assistant and for a certain question that the user asks, the chatbot responds with ‘Unknown’ or ‘No response’ or ‘I don’t know.’ These responses are supposed to be forbidden and must never be returned. Because of this, the designed model includes a file which has some predefined values for certain responses.

Given below is an example of possible responses and a score value of either 0 or -1.

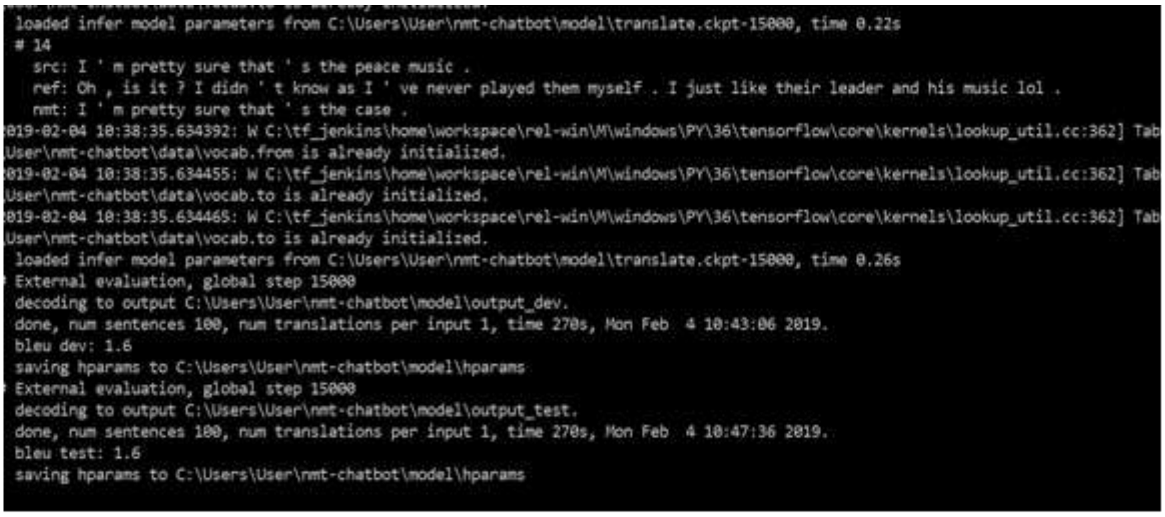
```
['<unk>', -1],
['Word <unk> word', -1],
['[ ]', 0],
['I \'', 0],
['It \'', 0],
['You \' re right , I', 0],
['That \' s what I \'', 0],
['Thank you', 1],
['You', 0],
['What \' s your', 0],
```

6.3. Architecture of Response Selection:

Heuristics for determining a response can be engineered in many different ways, from if-else conditional logic statements to machine learning classifiers. The simplest technique is using a set of rules and patterns as conditions for the rules.

During training, the object is a list of tuples, which contains:

- Source phrase
- Target phrase
- NMT phrase



```
 loaded infer model parameters from C:\Users\User\nmt-chatbot\model\translate.ckpt-15000, time 0.22s
 # 14
   src: I ' m pretty sure that ' s the peace music .
   ref: Oh , is it ? I didn ' t know as I ' ve never played them myself . I just like their leader and his music lol .
   nmt: I ' m pretty sure that ' s the case .
019-02-04 10:38:35.634392: W C:\tf_jenkins\home\workspace\rel-win\M\windows\PY\36\tensorflow\core\kernels\lookup_util.cc:362] Tab
User\nmt-chatbot\data\vocab.from is already initialized.
019-02-04 10:38:35.634455: W C:\tf_jenkins\home\workspace\rel-win\M\windows\PY\36\tensorflow\core\kernels\lookup_util.cc:362] Tab
User\nmt-chatbot\data\vocab.to is already initialized.
019-02-04 10:38:35.634465: W C:\tf_jenkins\home\workspace\rel-win\M\windows\PY\36\tensorflow\core\kernels\lookup_util.cc:362] Tab
User\nmt-chatbot\data\vocab.to is already initialized.
 loaded infer model parameters from C:\Users\User\nmt-chatbot\model\translate.ckpt-15000, time 0.26s
 External evaluation, global step 15000
 decoding to output C:\Users\User\nmt-chatbot\model\output_dev.
 done, num sentences 100, num translations per input 1, time 270s, Mon Feb  4 10:43:06 2019.
 bleu dev: 1.6
 saving hparams to C:\Users\User\nmt-chatbot\model\hparams
 External evaluation, global step 15000
 decoding to output C:\Users\User\nmt-chatbot\model\output_test.
 done, num sentences 100, num translations per input 1, time 270s, Mon Feb  4 10:47:36 2019.
 bleu test: 1.6
 saving hparams to C:\Users\User\nmt-chatbot\model\hparams
```

Figure 6. 2: Source, Target and NMT Phrase

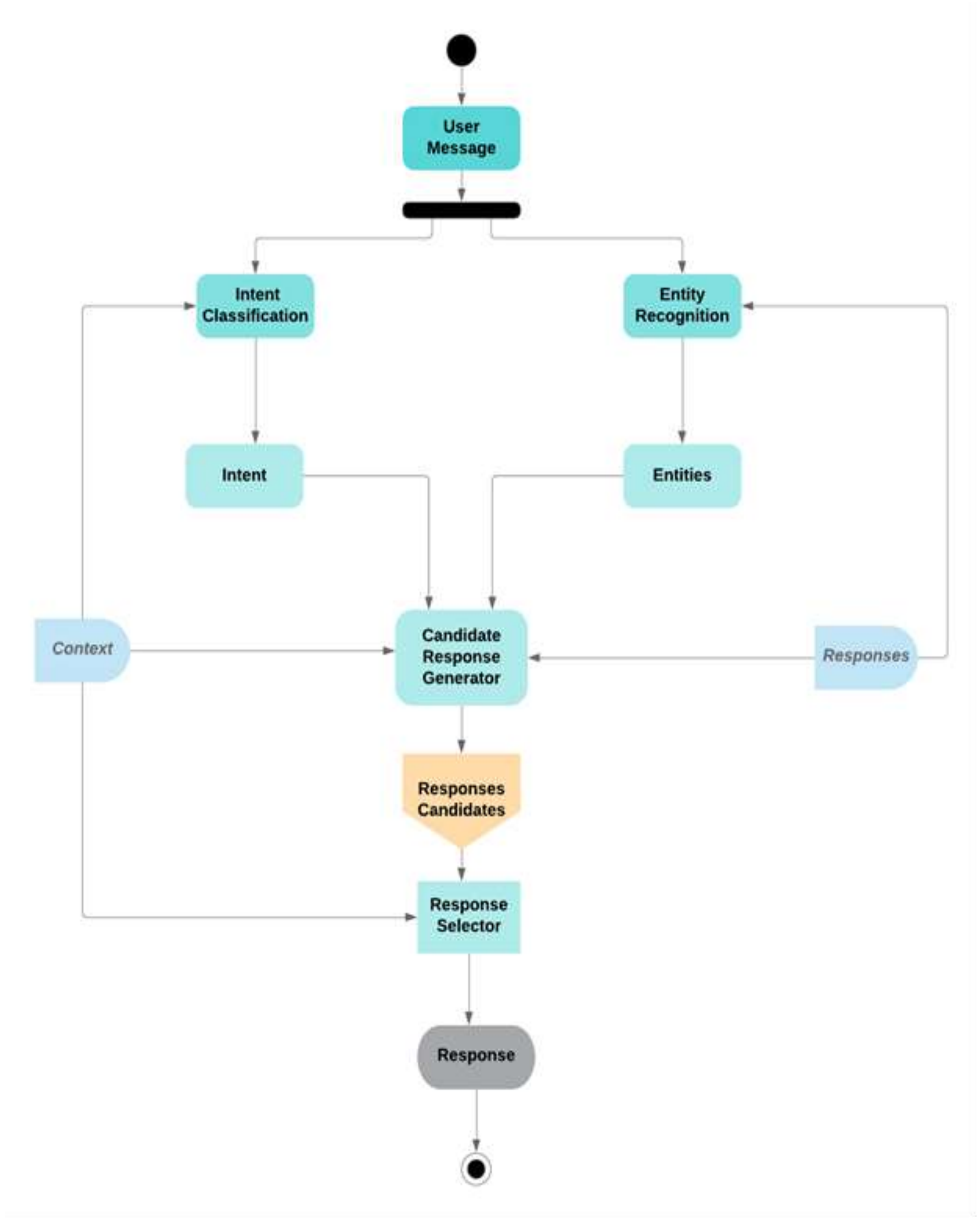


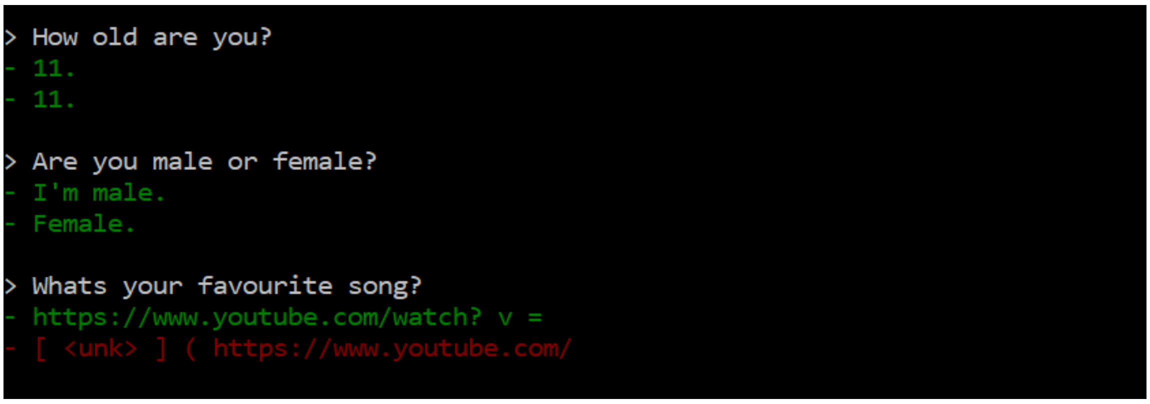
Figure 6. 3: Architecture of Hybrid Model with Response Selection

During training, the model is coded to show its progress over time. The factors that are outputted are mentioned below and discussed in the further chapters:

- Global step: The current step of training
- Step-time: Time taken for each step to be completed
- WPS. Words per second
- PPL: Perplexity
- gN: Gradient Norm
- BLEU: Bilingual Evaluation Understudy

When interacting with the chatbot, we are interested only in:

- Key - lowercase ASCII letters only plus underscore
- Value - float value



```
> How old are you?
- 11.
- 11.

> Are you male or female?
- I'm male.
- Female.

> Whats your favourite song?
- https://www.youtube.com/watch? v =
- [ <unk> ] ( https://www.youtube.com/
```

Figure 6. 4: Output: Question and Response Phrase

While the designed model is trained using deep learning algorithms, it is still necessary to add some restrictions on the data. These restrictions impose rules on the behavior of the model and eventually on the type of responses generated by the model. This makes the designed model a hybrid one, combining both the deep learning algorithms and algorithms of a rule-based model.

The rules are defined and added into the root directory on a setup file and called into the main program. They are described below;

1. Replace Answers: These are a list of regex-based replacements for the answers. The format is defined given below. In this example, the word ‘help’ is replaced with ‘assistance.’

   regex help –> assistance

2. Protected Phrases: These are a list of regex-based sentences that bypass tokenization. They are especially useful for hyperlinks and the format is simply open and closed parenthesis ().

Consider the website *www.datascience.com*. If this is made to pass through the model, then the tokenizer splits this into -

www data science . com
Protected phrases solve this issue by means of which it bypasses tokenization of certain words or hyperlinks.

3. Blacklist Answers: This is a very important regex-based rule file as it blocks the chatbot model from responding with certain words or phrases.

4. Replace Vocabulary and Vocabulary Blacklist are similar to Replace Answers and Blacklist Answers files except that these deal with vocabulary.

5. Detokenize Answers: This is a list of regex-based rules for detokenizing special words or sentences.

In the settings file, the following factors have been defined, which trains a decent model.

- Vocabulary size: 15000
- Test size: 100
- Learning rate: 0.001
- Beam width: 10
- Length Penalty Weight: 1.0
- Number of translations per input: 10

6.4. Detecting Emoticons or Emojis from the User:

Communication has come a long way since the internet revolution. From sending letters through post to the digital era of writing text messages or emails on computers that are sent over the internet, real-time communication is at its peak. Never before has it been so easy to communicate in real-time with people from around the globe. These days communication has changed with the introduction of multimedia. People are communicating with images, GIF'S, videos, and even emoticons.

An emoji is a visual depiction of a symbol, emotion or object. Emoji's are accessible in modern communication apps such as in smartphone's text messaging or social networking apps such as Instagram, Facebook, Twitter and WhatsApp. Emoticons communicate with the help of smiley's, the mood or reaction of the user. Its usage keeps increasing, and they even host the benefit of saving time from typing out on the keyboard.

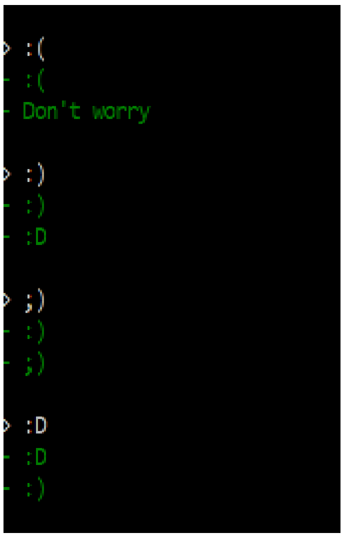
Facebook has released statistics showing an average of 5 billion emojis are being shared between users, each day on Messenger. Emojis are present in nearly a quarter of the tweets (19.6%) and are used by more than a third of the users (37.6%). Studies show that, while chatting with a chatbot or virtual assistant, the user may expect or could send emoticons in the text which is why it is necessary to detect these characters and send appropriate responses to them.

The logic used to detect emojis from the user is by examining the input to check for multiple characters that are not of ASCII value. If there are multiple characters which do not resemble any word in the data corpus, then the software suspects the usage of emoticons. It reads through each character and tries to match it with a list of existing emoticons which have been trained in the data corpus. For the chatbot designed, the goal is to replicate the mood or the emoticon and send the same emoticon as a response back to the user. The chatbot has been tested with some instances and the result is demonstrated below.

Instance 1: The character ☹ which is basically a **:** and **(** characters combined without spacing denote a sad face. The chatbot responds with a 'Don't worry' or by responding with the same emoticon.

Instance 2: The character 😊 which is basically a **:** and **)** characters combined without spacing denote a happy face. The chatbot responds with a :D (laughing) or by responding with the same emoticon.

One important scenario which must be taken into account is if the response of the chatbot is very similar to the question or the input from the user. For such cases, the response is penalized. First, the software is coded to try to remove any punctuation from the input (if exists) and next to remove the punctuation from the response (if exists). Next, the software checks if the question matches the response using a comparison operator and we give it a score of -5.



```
> :(
- :(
- Don't worry

> :)
- :)
- :D

> ;)
- :)
- ;)

> :D
- :D
- :)
```

Figure 6. 5: Chatbot responding to Emoticons

Consider if the user asks the chatbot to reserve a table at a restaurant, the chatbot then responds by sending the user with a link for reservation. A lot of times, these hyperlinks end with a '=' which must be avoided or removed altogether. The software runs through the code and checks the end of every response for the '='

character and if it finds the character, then the score is automatically deprecated to -3. By regression testing, another finding is that the chatbot responds with a lot of ‘unks’ or ‘unknown’ for any input that it does not understand. The score of such responses is deprecated to -4.

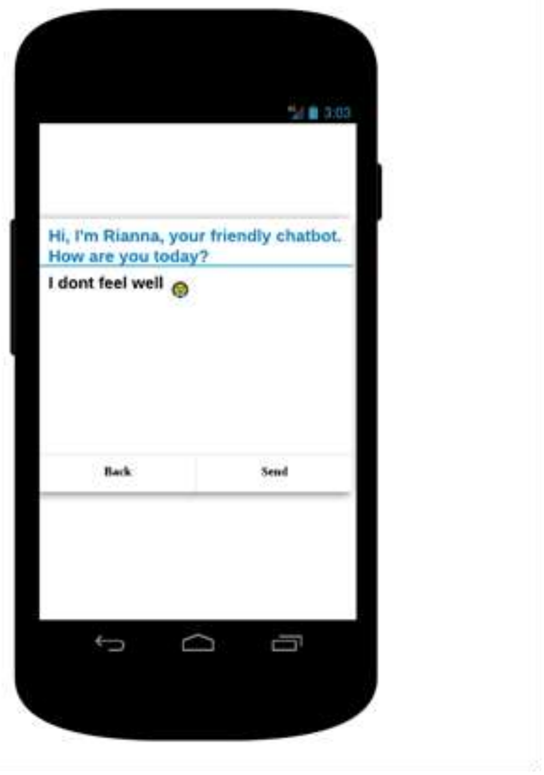


Figure 6. 6: Prototype Design on mobile interface

6.5 Deploying the Chatbot

Based on the model’s efficiency and accuracy, a chatbot can then be integrated on the website or deployed as a standalone application. Deploying the chatbot requires special permissions and access grants for third-party websites.
The process for deployment on a website is given below:

1. After the training is complete, the chatbot can be tested automatically deployed using prepare for deployment utility in python.

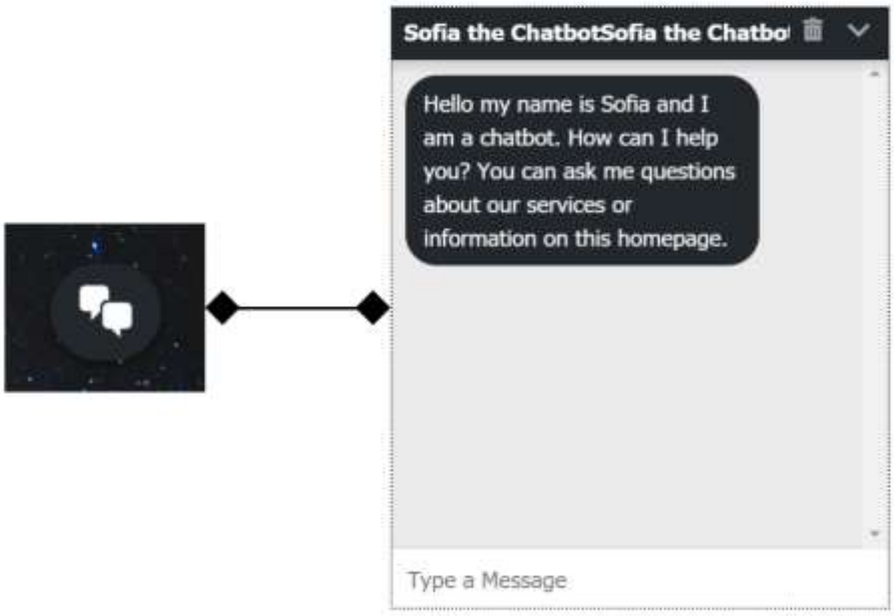
   cd utils
   python prepare_for_deployment.py

   The script creates a ‘deployment’ folder inside the project's root directory (NMT code, hprams, checkpoint file)

2. Copy the embedded code and files to the destination webpage.

3. Choose if chatbot should automatically begin a conversation when the webpage is loaded or wait until the chatbot widget is clicked. Paste the embedded code on the HTML site code, configure and enable plugins.

The figure below shows a prototype model of the designed chatbot renamed as ‘Sofia the Chatbot’ deployed on the company website.


Sofia the ChatbotSofia the Chatbo

Hello my name is Sofia and I am a chatbot. How can I help you? You can ask me questions about our services or information on this homepage.

Figure 6. 7: Prototype version of Deployed chatbot on website

Summary of Chapter: In this chapter, the training of the chatbot model and the architecture of response selection mechanism is discussed. Various kinds of tokenization, detokenization, Detecting the score of responses, Architecture of response selection. A new implementation of emoji detection is also considered, and its implementation in the design is discussed, concluding with the deployment on website.

Chapter 7

Results and Evaluation

Using the Python version 3.6, TensorFlow library and SQL a prototyped version of a chatbot which can hold open-ended conversations has been designed and developed. For experimental purposes, the model has been deployed both locally and, on the cloud, (PaperSpace), thus to locally access the chatbot for interaction, command prompt serves as interface medium. For evaluating the model, TensorFlow is used.

Python is a general-purpose, high-level programming language. Python is both object-oriented and imperative and can be used in a functional style as well. Python programs are portable, i.e., They can be ported to other operating systems like Windows, Linux, Unix, and Mac OS X, and they can be run on Java and .NET virtual machines. TensorFlow provides an open source artificial intelligence library which uses data flow graphs to build models. Besides that, it allows developers to create large-scale neural networks with many layers. TensorFlow is mainly used for Creation, Classification, Perception, Understanding, Discovering, and Prediction.

Overall the chatbot behaves and responds as expected. Although multiple training modules are required sometimes to get the desired response, the chatbot has a moderate response rate. One disadvantage is that it consumes a fairly lengthy training period (54 million rows of a dataset containing 1.7 billion comments) both locally and on the cloud. When trained locally, there were a total of three time-sinks described in the table below.

Activity	Time Required
Downloading and Acquisition of Data	7 to 8 hours
Formatting and Training Data	3 to 4 hours
Train Model	2 months to 3 months

Table 7.1: Time-sink in Development

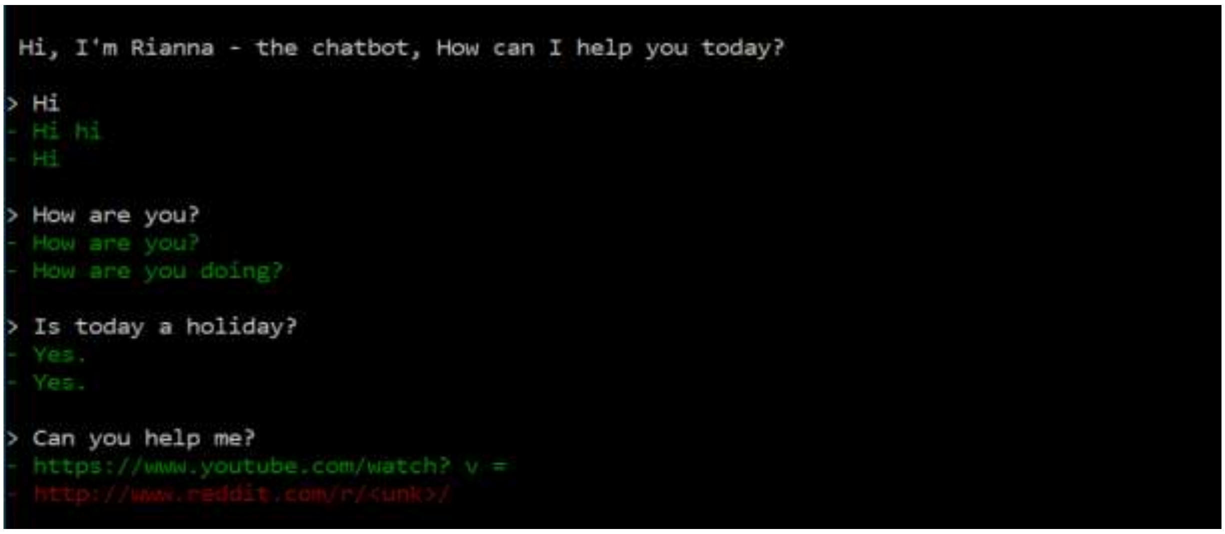
One important requirement for this project is approximately 50 to 60 Terabytes of space and a Graphical Processing Unit (GPU) to perform mathematically high computations without taking up too much time.

In some scenarios, the response returned show ASCII values for apostrophes, which means that the training module and dataset need to extend its capabilities to include punctuation marks. Certain incomplete hyperlinks or backlinks are returned when requested to redirect to certain information or to check on third-party websites.

The chatbot is structured in such a way that the output responses can be tweaked as required. For example, if the application expects a maximum output response of 10 words, then this can be set up internally. It might affect the quality of the chat outputs, but it just serves as an added functionality. Apart from this, certain phrases or words can be blocked from being trained or being expressed as a response. In the setup file, a ‘Blacklist’ file is created wherein all the blocked content is stored, and its usage is hindered.

With the objective of designing a hybrid model that relies on the combination of both deep and rule-based architectures, through which companies can better allocate their resources to perform more important and non-repetitive tasks, thus helping Organizations to reduce unnecessary overhead costs. This is mainly advantageous for start-ups because they tend to have lesser capital or reserves as investments.

The first objective, as discussed in Chapter 1 has been accomplished, and the results are shown in the table below, where the chatbot is able to hold average conversations. Further training is required in order to improve the quality of its chat responses.



```
 Hi, I'm Rianna - the chatbot, How can I help you today?

> Hi
- Hi hi
- Hi

> How are you?
- How are you?
- How are you doing?

> Is today a holiday?
- Yes.
- Yes.

> Can you help me?
- https://www.youtube.com/watch? v =
- http://www.reddit.com/r/<unk>/
```

Figure 7. 1: Chatbot Interacting with a user

From the screenshot above, the chatbots output translation rate is set to two, which is why we see two outputs (denoted in green for good responses and red for bad responses) for every question passed.

In the figure below, the models training corpus is displayed using TensorFlow. The blue dots are basically words. The dots are spread out in a way that two words which are synonyms or are used in the same context are spaced close together, when compared to other words which are placed farther apart. Figure 7.2 shows a corpus dictionary of words.

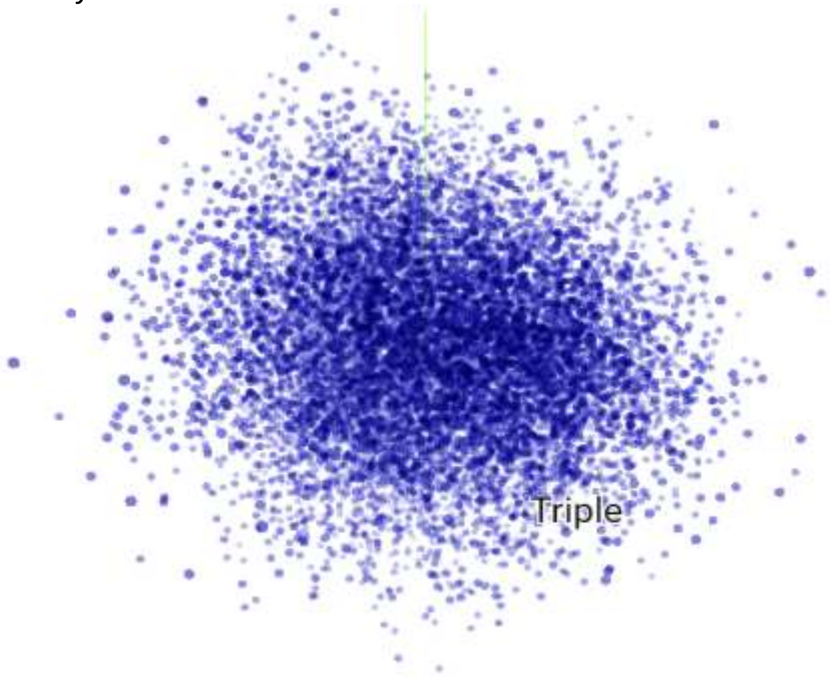


Figure 7. 2: Word Vector and Model Training

From Figure 7.3 and 7.4 displayed below, we take the example of the word ‘disliked’ from the corpus training set. This then highlights the closest matching pairs in terms of synonyms, antonyms and other words in a similar context.

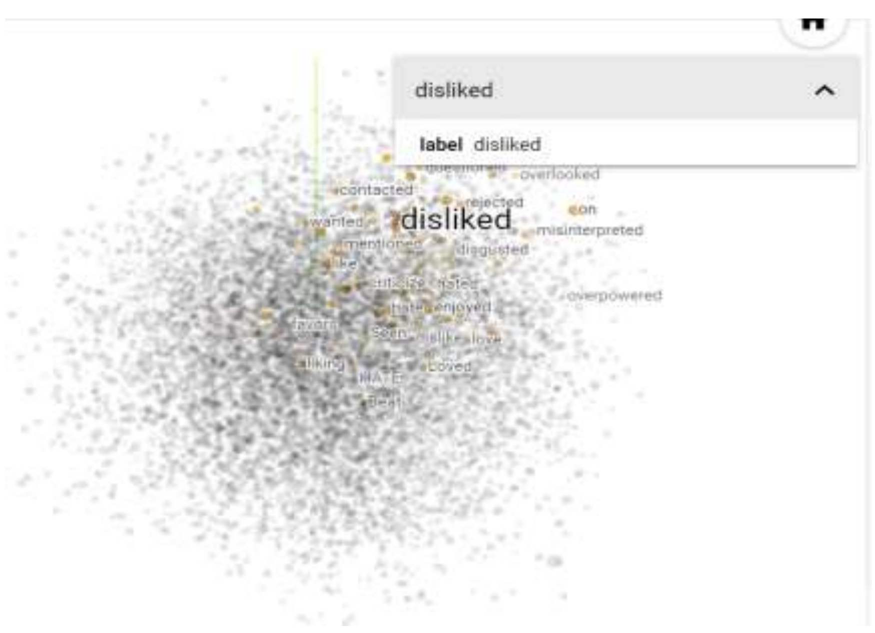
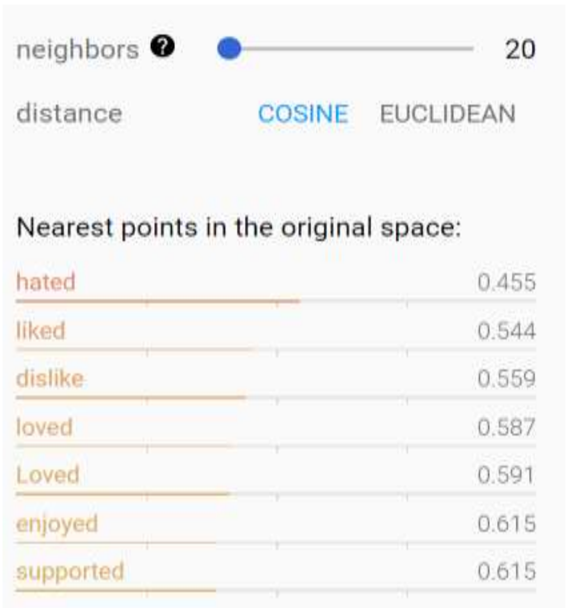


Figure 7. 3: Word Embedding ‘Disliked’

In Figure 7.4, the neighboring words as well as their vector values are displayed.

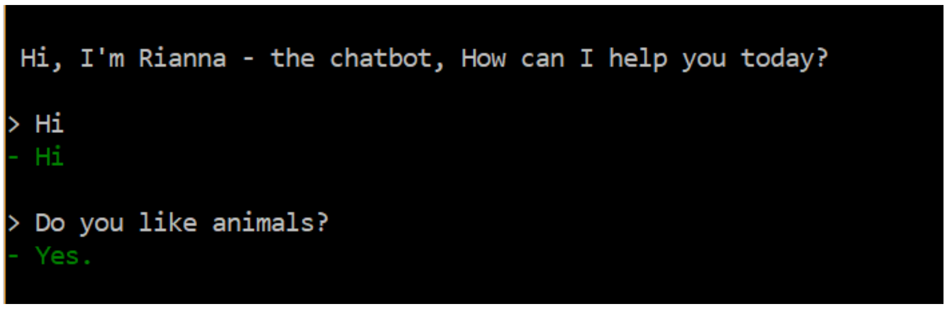


Nearest points in the original space:

hated	0.455
liked	0.544
dislike	0.559
loved	0.587
Loved	0.591
enjoyed	0.615
supported	0.615

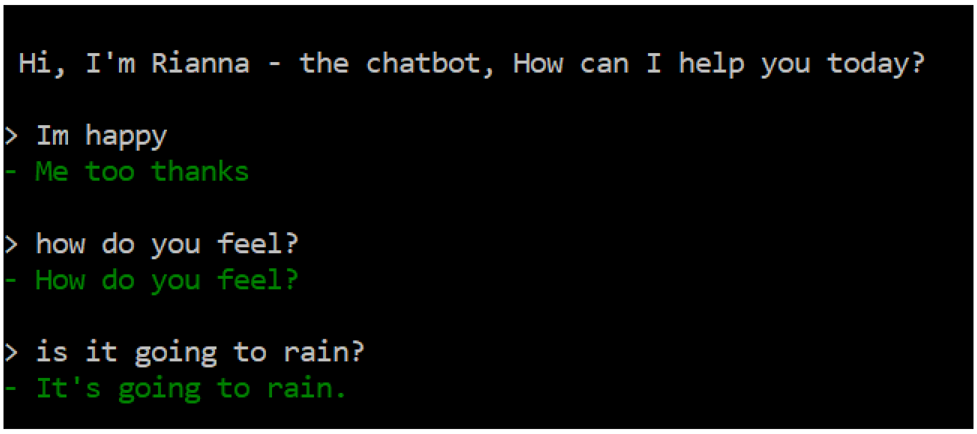
Figure 7. 4: Nearest Neighbors & Vector values ‘Disliked’

In the below images, the chatbot which is interacting with two users simultaneously is displayed. The response rate is average usually with a delay of 2 or 3 seconds per response.



```
Hi, I'm Rianna - the chatbot, How can I help you today?

> Hi
- Hi

> Do you like animals?
- Yes.
```

(a)



```
Hi, I'm Rianna - the chatbot, How can I help you today?

> Im happy
- Me too thanks

> how do you feel?
- How do you feel?

> is it going to rain?
- It's going to rain.
```

(b)

Figure 7. 5: Chatbot interacting with two users simultaneously

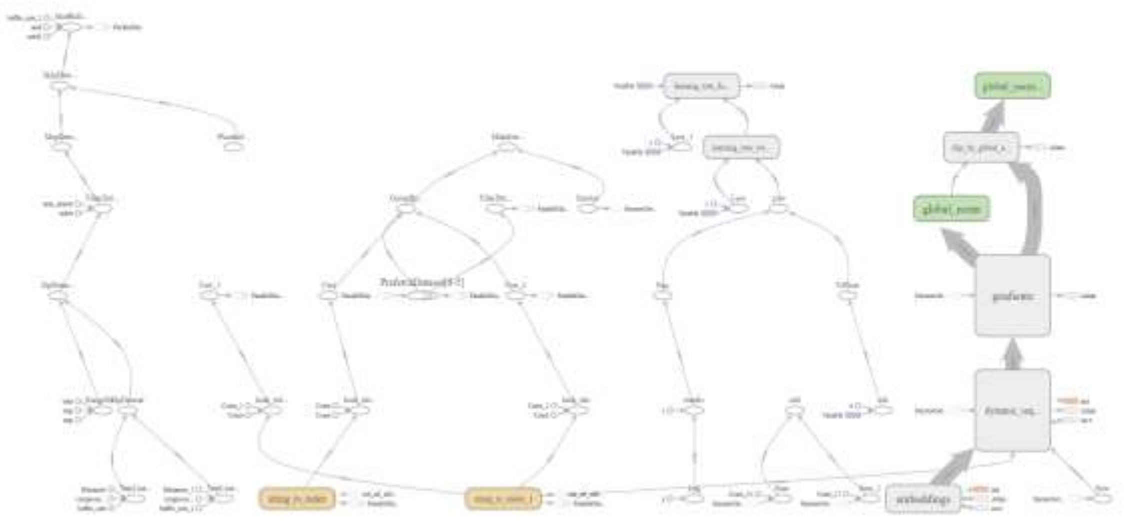


Figure 7. 6: Main graph of system from TensorFlow

Input from User	**Chat Response**
It’s a beautiful day, isn’t it?	It’s a beautiful day
How can I contact customer support?	You can send me us an email
What day is today?	Tuesday
How old are you?	How old are you?
What do you think of President Trump?	I don’t know
Are you a software?	No, I’m not…

Table 7. 2: User input and Chat Responses

7.1. Parameters to Measure Quality of Chatbot

In this section, we cover three important concepts that help measure the overall quality of the chatbot, thereby measuring the quality of chat responses. With the outcome of this section, Research Objective 2 mentioned in the Introduction is fulfilled.

7.1.1 BLEU

Bilingual Evaluation Understudy, commonly referred to as BLEU, is a popular algorithm for analyzing the quality of text which has been machine-translated from one natural language to another. The correspondence between a machine's output and the probabilistic response of a human being is compared, and the value is determined. The fundamental concept supporting BLUE is the following principle - ‘the closer a machine translation is to a professional human translation, the better it is.’

The technique adopted here is that the scores are calculated for individual translated sentences or segments by correlating them with a set of good quality reference translations. Without considering grammatical errors or intelligibility, the scores are averaged over the whole corpus to reach an estimate of the translation's overall quality. The output is always a number between 0 and 1. A score of 1 indicates a higher similarity of the candidate text to the reference text, and it is extraordinarily difficult for the NMT to achieve such scores. A score of 0 indicates an absolute mismatch of the candidate to reference text.

A very modified and renewed version of Precision is chosen to compare the candidate’s translation with reference translations. Python’s Natural Language Toolkit library (NLTK) provides an implementation of the BLEU score that can be used to evaluate the generated text against a reference text.

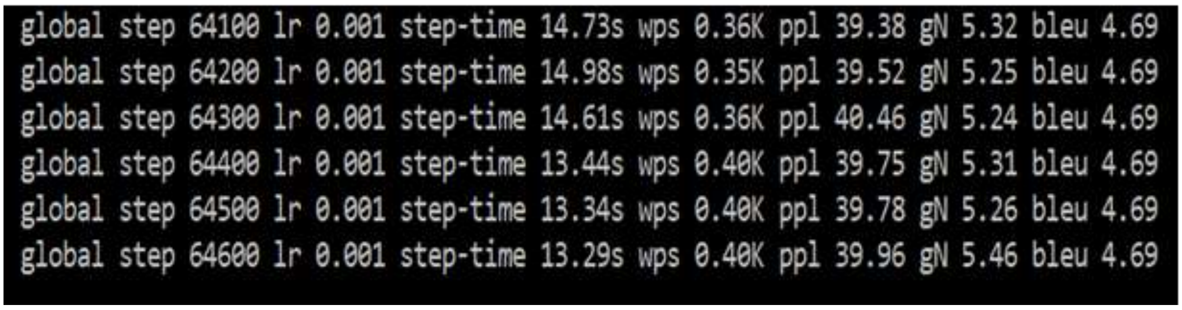
The reference sentences are considered as a list of sentences where each reference is an ordered list of tokens. The candidate sentence is provided as a list of tokens. For example:

```
from nltk.translate.bleu_score import sentence_bleu
reference = [[‘this’, ‘is’, ‘an’, ‘elephant’], [‘this’, ‘is’ ‘a’ ‘cat’]]
candidate = [‘this’, ‘is’, ‘a’, ‘cat”]
score = sentence_bleu(reference, candidate)
print(score)
```

Running this example prints a perfect score of 1 as the candidate matches one of the references exactly.

Some issues that arise when considering the BLUE score as an accurate measurement for the performance of Machine Translation systems, are given below:

- It doesn't consider meaning
- It doesn't directly consider sentence structure
- It doesn't handle morphologically rich languages well
- It doesn't map well to human judgments



```
global step 64100 lr 0.001 step-time 14.73s wps 0.36K ppl 39.38 gN 5.32 bleu 4.69
global step 64200 lr 0.001 step-time 14.98s wps 0.35K ppl 39.52 gN 5.25 bleu 4.69
global step 64300 lr 0.001 step-time 14.61s wps 0.36K ppl 40.46 gN 5.24 bleu 4.69
global step 64400 lr 0.001 step-time 13.44s wps 0.40K ppl 39.75 gN 5.31 bleu 4.69
global step 64500 lr 0.001 step-time 13.34s wps 0.40K ppl 39.78 gN 5.26 bleu 4.69
global step 64600 lr 0.001 step-time 13.29s wps 0.40K ppl 39.96 gN 5.46 bleu 4.69
```

Figure 7. 7: Trained Model displaying Step-time, words per second

Image below shows a graph of the gradual progression of BLEU score starting from step 10,000 to step 80,000 of training process. The highest recording of BLEU score is around 2.65 at step 53,000. This is the step at which the responses generated from the chatbot are reasonably good.

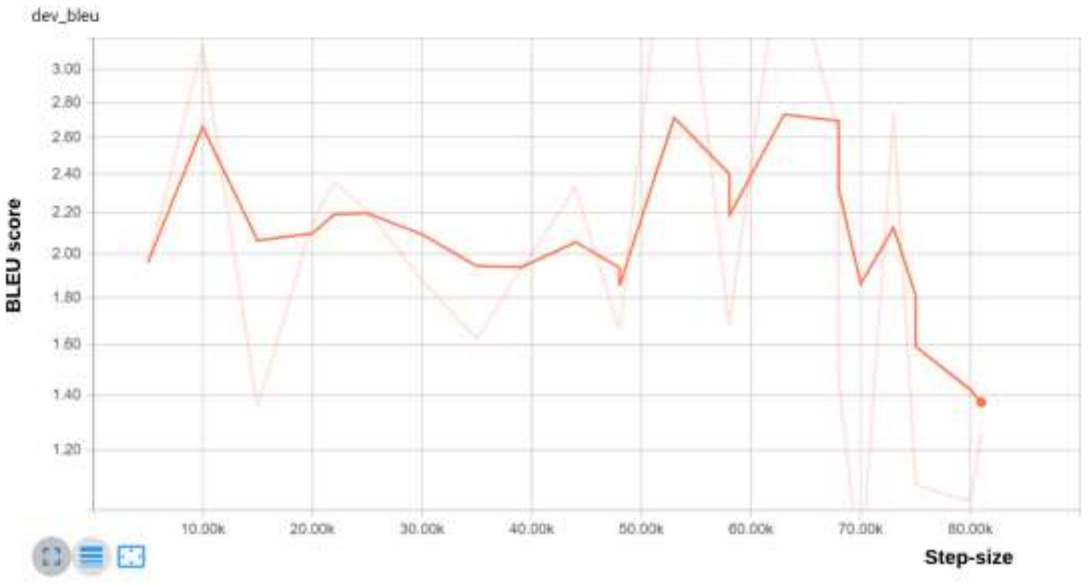


Figure 7. 8: Step-size (x-axis) versus BLEU score (y-axis) at 80k

Since it in necessary to understand how the model behaves with time, the BLEU score at step-size of 80k is compared with the BLEU score at 100k. At 80k, the model hits an all-time low of 1.40 score and then steadily rises to reach 2.0 at 100k step-size.

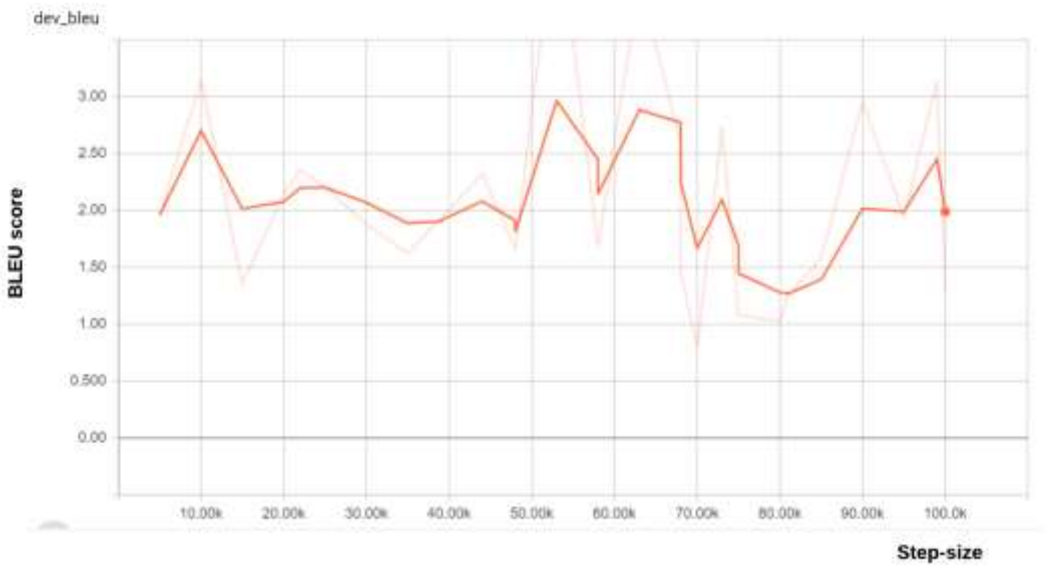


Figure 7. 9: Step-size (x-axis) versus BLEU score (y-axis) at 100k

7.1.2 Perplexity

Perplexity, commonly referred to as PPL, is a decent measurement to calculate a model's effectiveness. In information theory, perplexity is a measurement of how well a probability distribution model predicts a sample. A Lower perplexity indicates the probability distribution is good at predicting the sample.

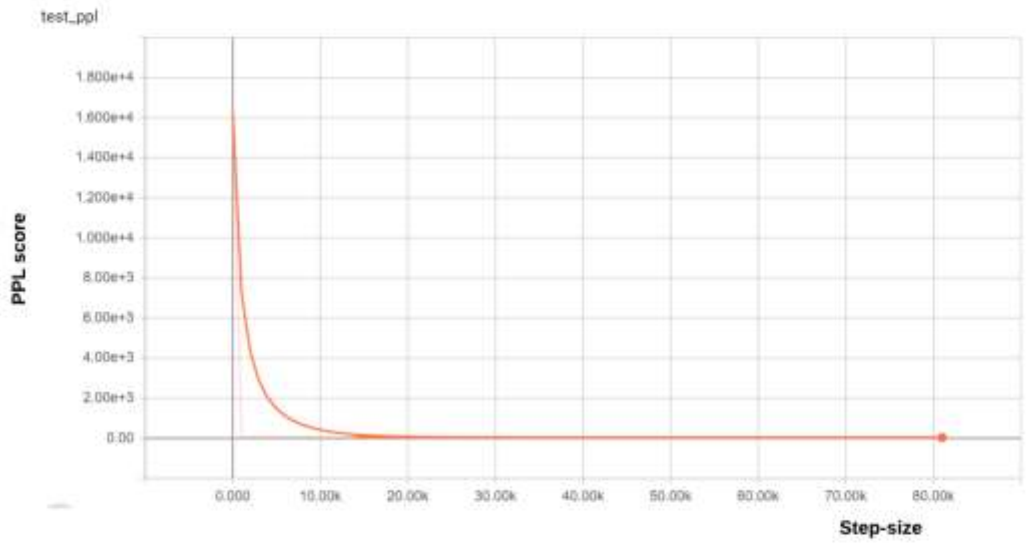


Figure 7. 10: Step-size (x-axis) versus Perplexity (y-axis) at 80k

Contrary to BLEU, the lower the better its performance, as it's a probability distribution of how effective your model is at predicting an output from a sample. While training the model, a slow but gradual increase in BLUE score and a continuous decrease in PPL indicates that the chatbot is behaving as it should. If both the BLUE and PPL scores are increasing simultaneously, it indicates a more robotic-like response, rather than highly variant ones.

Figure above shows the gradual decrease of the Perplexity factor which starts off considerably high at exponential value of 1.6 and gradually decreases until it reaches 0 from step 18,000 onwards.

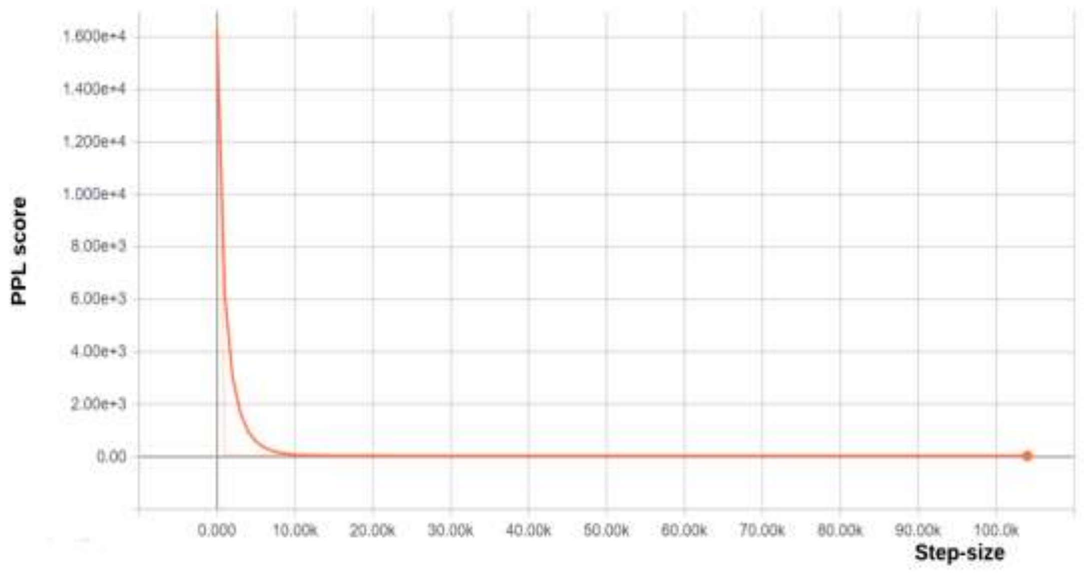


Figure 7. 11: Step-size (x-axis) versus Perplexity (y-axis) at 100k

The perplexity score is compared at step-sizes 80k and 100k although the PPL score seems to remain constant at 0.

7.1.3 LOSS

Loss is a measure of how far off the output layer of the neural network was compared to the sample data. So, naturally, the lower the loss, the better the model efficiency.

The train loss factor when compared at step-size of 80k and 100k shows a considerable difference. At step-size of 80k the train loss is at 80 and at 100k the train loss falls to 60.

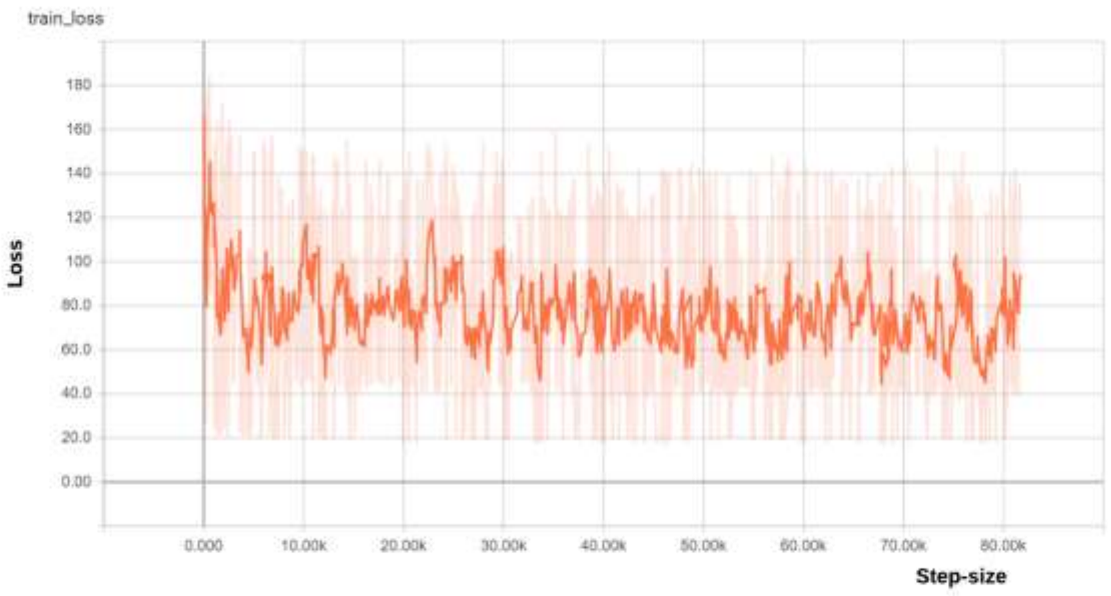


Figure 7. 12: Figure: Step-size (x-axis) versus Train loss (y-axis) at 80k

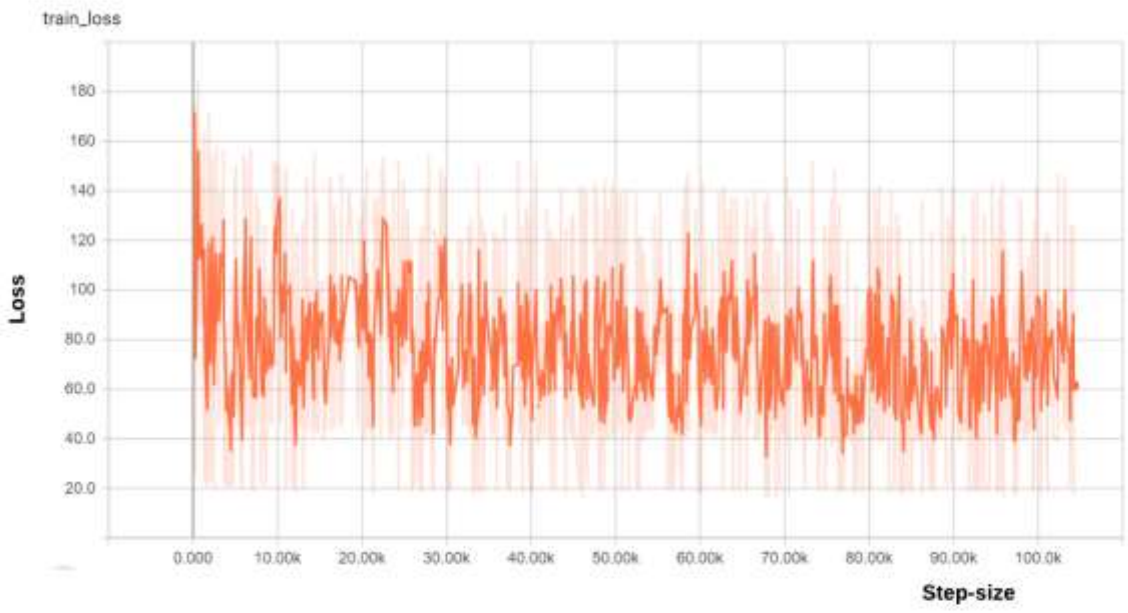


Figure 7. 13: Step-size (x-axis) versus Train loss (y-axis) at 100k

Summary of Chapter: In the Results chapter, the experimental results are discussed and shown. Some parameters used for calculating the efficiency and effectiveness of the designed model are also described in this chapter. One important note is to point is that all the above result parameters (BLEU, Perplexity, Loss) have been illustrated from step 0 to step 80,000 and from step 0 to step 100,000 for the purpose of comparison. Note that the BLEU score, Perplexity and Loss are measured against the step-size of training on the x-axis.

Chapter 8

Challenges Related to Chatbots

Despite their recognition and trendiness, chatbots are a long way from being perfect or intelligent. Many issues still arise in the development, maintenance, and reliability of the chatbots. In this section, we list out some of the most important and obvious challenges faced in the software and hardware implementation of chatbots.

8.1. Natural Language Processing:

Rule-based chatbots and machine learning chatbots both require large amounts of training data in order to understand diverse user requests. For open-ended chatbots, training data must be enormous in order to understand the user’s context or intent. The next challenge is for the chatbot to correctly derive meaning and understand the user's request. Natural Language processing techniques still face issues with long-tail sentence sequences because of the structure of the chatbot is designed to be compact and rigid, and therefore, they have a very small inbuilt memory. The chatbot models are relatively easier to apply for the English language because of the simple and easy to understand the structure of the language. But for other more complex language structures, the chatbot might be too complex to implement or understand the user intent.

8.2. Training the Model Locally:

An open-ended model usually contains a larger dictionary of sentences for the chatbot to train from. This is so that the chatbot can hold a wider domain of conversations. But the creation of the database by extracting useful data on a local desktop usually takes about 5 to 10 hours for about 4GB of training data. For this project, two experimental techniques were adopted.

One by training the chatbot model locally on the desktop and second, on the cloud using PaperSpace. Training the model locally requires high computational power, GPU, extremely high CPU load, Training for 4GB of data requires more than two months. PaperSpace, on the other hand, is a cloud platform for performing high load computations with the help of virtual machines.

Its machine learning template is intended to provide a fully functional machine learning environment for interactive development. PaperSpace provides a package called ‘ML in a box,’ a Linux machine which comes pre-installed with the following software:

- NVIDIA Driver: 410.48
- CUDA 10.0.130-1
- cuDNN 7.3.1.20-1+cuda10.0
- TensorFlow 1.11.0-rc1
- PyTorch 1.0.0a0+39bd73a
- Keras 2.2.2
- Theano 1.0.3
- NVIDIA-docker 2.0.3
- Docker 18.06.1-ce
- Atom 1.31.2
- jupyter-notebook 5.7.0
- Python 3.6.6
- Chromium Browser 69.0.3497.81

Training on the virtual machine on the cloud consumes half the time as compared to training the model locally. But the speed significantly improves, and 4GB of data takes about two weeks for training.

8.3. Cleaning Up Dataset

For open-ended or non-task specific chatbots, large amounts of high-quality data are received and following which they are trained. But for certain application-specific bots, it is vital that the chatbot responds exactly in a specific manner or with specific responses. For this reason, sometimes it is necessary for the dataset to be cleaned before the training the model. Even this is insufficient if the application is sensitive. Consider medical chatbots that contain sensitive user information and are generally at a higher risk. Malfunctioning of such chatbots could lead to disastrous outcomes, and these applications have higher liability. These application-specific bots are not usually not designed to be open-ended so that their responses are more controlled. They usually have a database that is thoroughly and regularly checked by quality analysts for bugs and quality of data. Since the model gets continuously trained and its database increases with the increase in conversations, it is crucial to repeatedly check the chatbot response. One way to automate the quality of conversation is by asking the user to rate the quality of the conversation. Based on this, it is possible to customize or personalize the chat responses as per user requirements.

Chapter 9

Conclusion

A brief overview of the history of chatbots has been given, and the encoder-decoder NMT model has been described. Afterward, an in-depth survey of scientific literature related to conversational models, published in the last 3 years, was presented. Various techniques and architectures were discussed, that were proposed to augment the model and to make conversational agents more natural and human-like.

Looking back at the accomplished work, it can be concluded that the chatbot model developed is satisfying and allows for easy deployment and development if changes need to be performed. It can be ported on virtually any system provided it can host a Python environment. Furthermore, it leaves the door open for additional languages' support and user problems. It is not a perfect solution, but considering the requirements, it works well in practice. There might be a few pathological cases where the chatbot might not behave in the best manner possible, so the future system maintainers need to be aware of that.

9.1. Future Work

The following list should be taken lightly, as no extensive studies have been conducted to confirm they are indeed feasible in this case. They are intended more as brainstorming than an actual roadmap to follow for the future of the chatbot.

Some features and wow-factors that can be added into the application for further improvements are given as follows;

1. Website Integration of chatbot or integrations into different types of API can be implemented.

2. The biggest challenge for the future would be to switch the architecture of the AI, from the hybrid rule and deep learning based purely to a deep neural net, which is at the last stage is supervised by humans.

This is, of course, a very tedious process, for humans to go through and filter from such large amounts of data.

3. Implementing the chatbot to include speech recognition as well as multiple languages.

9.2. Closing Thoughts

The future holds promising results, especially in the field of Natural Language Processing. New techniques and technologies are born every day, and older techniques and algorithms are optimized. The growth of chatbots left a mark in history and in the minds of consumers. Besides, it might entirely be possible that, in the near future, the roles might be reversed between humans and bots.

To go further with that thought, we might see one day, bots interacting with other bots using human language as means of communication to improve the quality of life for mankind. Maybe in a few years, these technologies will finally be indistinguishable from humans.

It is necessary to subject our technological designs and developments to Isaac Asimov's ‘Laws of Robotics’ stated below;

1. A robot may not injure a human being or, through inaction, allow a human being to come to harm.
2. A robot must obey orders given it by human beings except where such orders would conflict with the First Law.
3. A robot must protect its own existence as long as such protection does not conflict with the First or Second Law.

The resources and intelligence bestowed on humans can be used either for the betterment of the world and of science. On the contrary, they can also be used for mass destruction or personal benefits. It is solely up to the scientists, architects’ designers, researchers, and users of these technologies to implement them for the betterment of humanity. There need to be a moral set of standards or codes for the use and implementation of all intelligent machines not limiting to robots or chatbots in society. Therefore, we need to be careful about the way researchers tackle these issues to make sure they do not end up in the wrong hands or for wrongful purposes.

PART III

APPENDICES

Appendix A

Code Listing

1. Training the Model

```
import sys
import os
import argparse as ap
import math
from setup.settings import hparams, preprocessing
from setup.custom_summary import custom_summary
sys.path.append(os.path.dirname(os.path.realpath(__file__)))
sys.path.append(os.path.dirname(os.path.realpath(__file__)) + "/nmt")
from nmt import nmt
import tensorflow as tf
import threading

# Cross-platform colored terminal text
from colorama import init, Fore
init()

'''
Start training model with custom decaying scheme and trainig duration
'''
def train_model():

    print('\n\n{}Started training model...{}\n'.format(Fore.GREEN, Fore.RESET))

    # Custom epoch training and decaying
    if preprocessing['epochs'] is not None:

        # Load corpus size, calculate number of steps
        with open('{}/corpus_size'.format(preprocessing['train_folder']), 'r') as f:
            corpus_size = int(f.read())

        # Load current train progress
        try:
            with open('{}epochs_passed'.format(hparams['out_dir']), 'r') as f:
                initial_epoch = int(f.read())
        except:
            initial_epoch = 0

        # Iterate thru specified epochs
        for epoch, learning_rate in enumerate(preprocessing['epochs']):

            # Check if model already passed that epoch
            if epoch < initial_epoch:
                print('{}Epoch: {}, learning rate: {} - already passed{}'.format(Fore.GREEN, epoch + 1,
learning_rate, Fore.RESET))
                continue

            # Calculate new number of training steps - up to the end of current epoch
            number_training_steps = math.ceil((epoch + 1) * corpus_size / (hparams['batch_size'] if
'batch_size' in hparams else 128))

            # Update
            print("\n{}Epoch: {}, steps per epoch: {}, epoch ends at {} steps, learning rate: {} -
training{}\n".format(
                Fore.GREEN,
                epoch + 1,
                math.ceil(corpus_size / (hparams['batch_size'] if 'batch_size' in hparams else 128)),
                number_training_steps,
                learning_rate,
                Fore.RESET
            ))

            # Override the hyperparameters
            hparams['num_train_steps'] = number_training_steps
            hparams['learning_rate'] = learning_rate
            hparams['override_loaded_hparams'] = True

            # Run TensorFlow threaded (exits on finished training, but we want to train more)
            thread = threading.Thread(target=nmt_train)
            thread.start()
            thread.join()

            # Save epoch progress so that we can handle interruption
            with open('{}epochs_passed'.format(hparams['out_dir']), 'w') as f:
                f.write(str(epoch + 1))

    #default training
    else:
        nmt_train()

    print('\n\n{}Finished training model{}\n'.format(Fore.GREEN, Fore.RESET))

'''
Start training model with default settings
'''
def nmt_train():
    nmt_parser = ap.ArgumentParser()
    nmt.add_arguments(nmt_parser)

    # Get settingds from configuration file
    nmt.FLAGS, unparsed = nmt_parser.parse_known_args(['--' + key + '=' + str(value) for key, value
in hparams.items()])

    # Custom summary callback hook
    nmt.summary_callback = custom_summary

    # Run tensorflow with modified arguments
    tf.app.run(main=nmt.main, argv=[os.getcwd() + '\nmt\nmt\nmt.py'] + unparsed)

# Start training model
train_model()
```

2. Inference

```
import sys
import os
original_cwd = os.getcwd()
os.chdir(os.path.dirname(os.path.realpath(__file__)))
```

```
sys.path.append(os.path.dirname(os.path.realpath(__file__)))
sys.path.append(os.path.dirname(os.path.realpath(__file__)) + "/nmt")
sys.path.insert(0, os.path.dirname(os.path.realpath(__file__)) + "/setup")
sys.path.insert(0, os.path.dirname(os.path.realpath(__file__)) + "/core")
from nmt import nmt
import argparse as ap
from settings import hparams, out_dir, preprocessing, score as score_settings
sys.path.remove(os.path.dirname(os.path.realpath(__file__)) + "/setup")
import tensorflow as tf
from tokenizer import tokenize, detokenize, apply_bpe, apply_bpe_load
from sentence import replace_in_answers, normalize_new_lines
from scoring import score_answers
sys.path.remove(os.path.dirname(os.path.realpath(__file__)) + "/core")
import random

# Cross-platform colored terminal text
from colorama import init, Fore
init()

# For displaying only program related output
current_stdout = None

'''
Set global tuple with model, flags and hparams which is used in inference
'''
def setup_inference_parameters(out_dir, hparams):

    # Print output on stdout while temporarily sending other output to /dev/null
    global current_stdout
    current_stdout = sys.stdout
    sys.stdout = open(os.devnull, "w")

    nmt_parser = ap.ArgumentParser()
    nmt.add_arguments(nmt_parser)
    # Get settingds from configuration file
    flags, unparsed = nmt_parser.parse_known_args(['--'+key+'='+str(value) for key,value in
hparams.items()])

    # Add output (model) folder to flags
    flags.out_dir = out_dir

    ## Exit if model folder doesn't exist
    if not tf.gfile.Exists(flags.out_dir):
        nmt.utils.print_out("# Model folder (out_dir) doesn't exist")
        sys.exit()

    # Load hyper parameters (hparams) from model folder
    hparams = nmt.create_hparams(flags)
    hparams = nmt.create_or_load_hparams(flags.out_dir, hparams, flags.hparams_path,
save_hparams=True)

    # Choose checkpoint (provided with hparams or last one)
    if not flags.ckpt:
        flags.ckpt = tf.train.latest_checkpoint(flags.out_dir)

    # Create model
    if not hparams.attention:
        model_creator = nmt.inference.nmt_model.Model
    elif hparams.attention_architecture == "standard":
        model_creator = nmt.inference.attention_model.AttentionModel
    elif hparams.attention_architecture in ["gnmt", "gnmt_v2"]:
        model_creator = nmt.inference.gnmt_model.GNMTModel
    else:
        raise ValueError("Unknown model architecture")
    infer_model = nmt.inference.model_helper.create_infer_model(model_creator, hparams, None)

    return (infer_model, flags, hparams)

'''
Do actual inference based on supplied data, model and hyper parameters returning answers
'''
def do_inference(infer_data, infer_model, flags, hparams):

    # Disable TF logs and set stdout to devnull
    os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
    global current_stdout
    if not current_stdout:
        current_stdout = sys.stdout
        sys.stdout = open(os.devnull, "w")

    # Spawn new TF session
    with tf.Session(graph=infer_model.graph, config=nmt.utils.get_config_proto()) as sess:

        # Load model
        loaded_infer_model = nmt.inference.model_helper.load_model(infer_model.model, flags.ckpt,
sess, "infer")

        # Run model (translate)
        sess.run(
            infer_model.iterator.initializer,
            feed_dict={
                infer_model.src_placeholder: infer_data,
                infer_model.batch_size_placeholder: hparams.infer_batch_size
            })


        # Calculate number of translations to be returned
        num_translations_per_input = max(min(hparams.num_translations_per_input,
hparams.beam_width), 1)

        answers = []
        while True:
            try:

                nmt_outputs, _ = loaded_infer_model.decode(sess)

                if hparams.beam_width == 0:
                    nmt_outputs = nmt.inference.nmt_model.np.expand_dims(nmt_outputs, 0)

                batch_size = nmt_outputs.shape[1]

                for sent_id in range(batch_size):

                    # Iterate through responses
                    translations = []
                    for beam_id in range(num_translations_per_input):

                        if hparams.eos: tgt_eos = hparams.eos.encode("utf-8")

                        # Select a sentence
                        output = nmt_outputs[beam_id][sent_id, :].tolist()

                        # If there is an eos symbol in outputs, cut them at that point
                        if tgt_eos and tgt_eos in output:
                            output = output[:output.index(tgt_eos)]
                        print(output)

                        # Format response
                        if hparams.subword_option == "bpe":  # BPE
                            translation = nmt.utils.format_bpe_text(output)
                        elif hparams.subword_option == "spm":  # SPM
                            translation = nmt.utils.format_spm_text(output)
                        else:
                            translation = nmt.utils.format_text(output)

                        # Add response to the list
                        translations.append(translation.decode('utf-8'))

                    answers.append(translations)

            except tf.errors.OutOfRangeError:
                print("end")
                break

        # Reset back stdout and log level
        os.environ['TF_CPP_MIN_LOG_LEVEL'] = '0'
        sys.stdout.close()
        sys.stdout = current_stdout
        current_stdout = None

        return answers

'''
Setup inference parameters and then invoke inference engine passing question and inference parameters
'''
def start_inference(question):

    global inference_helper, inference_object

    # Set global tuple with model, flags and hparams
    inference_object = setup_inference_parameters(out_dir, hparams)

    # Update inference_helper with actual inference function as we have completed inference parameter settings
    inference_helper = lambda question: do_inference(question, *inference_object)

    # Load BPE join pairs
    if preprocessing['use_bpe']: apply_bpe_load()

    # Finally start inference
    return inference_helper(question)

# Model, flags and hparams
inference_object = None

# Function call helper
inference_helper = start_inference

'''
```

```
Main inference function which returns answer list for the given set of one or more questions
'''
def inference(questions):

    # Change current working directory (needed to load relative paths properly)
    os.chdir(os.path.dirname(os.path.realpath(__file__)))

    # Process questions
    answers_list = process_questions(questions)

    # Change directory back to original directory from where we started
    os.chdir(original_cwd)

    # Return answer_list
    if not isinstance(questions, list):
        return answers_list[0]
    else:
        return answers_list

'''
Get index and score for the best answer based on one of the following three settings
- default : pick first available best scored response
- random best score : pick random best scored response
- random above threshold : pick random response with score above threshold
'''
def get_best_score(answers_score):

    # Return first available best scored response
    if score_settings['pick_random'] is None:
        max_score = max(answers_score)
        if max_score >= score_settings['bad_response_threshold']:
            return (answers_score.index(max_score), max_score)
        else:
            return (-1, None)

    # Return random best scored response
    elif score_settings['pick_random'] == 'best_score':
        indexes = [index for index, score in enumerate(answers_score) if score == max(answers_score) and score >= score_settings['bad_response_threshold']]
        if len(indexes):
            index = random.choice(indexes)
            return (index, answers_score[index])
        else:
            return (-1, None)

    # Return random response with score above threshold
    elif score_settings['pick_random'] == 'above_threshold':
        indexes = [index for index, score in enumerate(answers_score) if score > (score_settings['bad_response_threshold'] if score_settings['bad_response_threshold'] >= 0 else max(score)+score_settings['bad_response_threshold'])]
        if len(indexes):
            index = random.choice(indexes)
            return (index, answers_score[index])
        else:
            return (-1, None)

    # Else return starting score by default
    return (0, score_settings['starting_score'])

'''
```

```
Process question or list of questions and provide answers with scores
It does it in three steps:
- prepare list of questions (can be one or more)
- run inference to obtain answers list (each question can have one or more answers)
- return answers along with scores and best score answer
'''
def process_questions(questions, return_score_modifiers = False):

    # Make sure questions is list (convert to list if only one question)
    if not isinstance(questions, list):
        questions = [questions]

    # Clean and tokenize
    prepared_questions = []
    for question in questions:
        question = question.strip()
        if question:
            prepared_questions.append(apply_bpe(tokenize(question)))
        else:
            prepared_questions.append('missing_question')

    # Run inference function
    answers_list = inference_helper(prepared_questions)

    # Process answers to return list along with score
    prepared_answers_list = []
    for i, answers in enumerate(answers_list):
        answers = detokenize(answers)
        answers = replace_in_answers(answers)
        answers = normalize_new_lines(answers)
        answers_score = score_answers(questions[i], answers)
        best_index, best_score = get_best_score(answers_score['score'])

        if prepared_questions[i] == 'missing_question':
            prepared_answers_list.append(None)
        elif return_score_modifiers:
            prepared_answers_list.append({'answers': answers, 'scores': answers_score['score'],
'best_index': best_index, 'best_score': best_score, 'score_modifiers':
answers_score['score_modifiers']})
        else:
            prepared_answers_list.append({'answers': answers, 'scores': answers_score['score'],
'best_index': best_index, 'best_score': best_score})

    return prepared_answers_list

'''
Two ways to invoke inference function
- by giving input file of questions
- interactive mode and feeding each question
'''
if __name__ == "__main__":

    # Specify input file
    if sys.stdin.isatty() == False:

        # Process questions
        answers_list = process_questions(sys.stdin.readlines())

        # Print answers
        for answers in answers_list:
            print(answers['answers'][answers['best_index']])

        # Exit
        sys.exit()

    # If no file is specified, enter in interactive mode
    print('\n\n{}Starting interactive mode:{}\n'.format(Fore.GREEN, Fore.RESET))

    # In loop, get user question and print answer along with score
    while True:
        question = input("\n> ")
        answers = process_questions(question, True)[0]

        if answers is None:
            print("\n{}Question can't be empty!{}\n".format(Fore.RED, Fore.RESET))
        else:
            for i, answer in enumerate(answers['scores']):
                if answers['scores'][i] == max(answers['scores']) and answers['scores'][i] >=
score_settings['bad_response_threshold']:
                    fgcolor = Fore.GREEN
                elif answers['scores'][i] >= score_settings['bad_response_threshold']:
                    fgcolor = Fore.YELLOW
                else:
                    fgcolor = Fore.RED
                print("\n{}{}[{}]{}\n".format(fgcolor, answers['answers'][i], answers['scores'][i],
Fore.RESET))

os.chdir(original_cwd) # Change back to original directory
```

3. Scoring the Response

```
from string import punctuation
import re

#List of punctuations that usually end a sentence
end_sentence_punctuations = ["'",'"',"!","?",".",")"]
#Add what are considered as bad responses (e.g. banned or curse words)
bad_responses = ["banned_word1", "banned_word2", "curse_word1", "curse_word2"]

'''
Utility function to remove punctuations from a sentence
'''
def remove_punctuations(txt):
    for individual_punctuation in punctuation:
        txt = txt.replace(individual_punctuation,'')
    return txt

'''
Penalize if most of the answer contains repetitive words
'''
def answer_echo(index, question, answer):
    try:
        score_modifier = 0

        answer = remove_punctuations(answer)
        answer_tokenized = answer.split(' ')

        tokens_length = len(answer_tokenized)
        token_repeats = 0
        for token in answer_tokenized:
            if answer_tokenized.count(token) > 1:
                token_repeats += 1

        repeat_percentage = float(token_repeats)/ float(tokens_length)
        if repeat_percentage == 1.0:
            score_modifier = -5
        elif repeat_percentage >= 0.75:
            score_modifier = -4
        return score_modifier
    except Exception as exception:
        print(str(exception))
        return score_modifier

'''
Penalize if most of the answer contains words from question
'''
def answer_echo_question(index, question, answer):
    try:
        score_modifier = 0

        answer = remove_punctuations(answer)
        question = remove_punctuations(question)

        answer_tokenized = answer.split(' ')
        question_tokenized = question.split(' ')

        tokens_length = len(answer_tokenized)

        if tokens_length == 1:
            return score_modifier
        else:
            token_repeats = 0
            for token in answer_tokenized:
                if question_tokenized.count(token) > 0:
                    token_repeats += 1

            echo_percentage = float(token_repeats)/ float(tokens_length)
            if echo_percentage == 1.0:
                score_modifier = -5
            elif echo_percentage >= 0.75:
                score_modifier = -4
            return score_modifier
    except Exception as exception:
        print(str(exception))
        return score_modifier

'''
Penalize if answer and question are the same or are part of each other
'''
def is_answer_similar_to_question(index, question, answer):
    try:
        score_modifier = 0

        question = remove_punctuations(question)
        answer = remove_punctuations(answer)

        if question == answer:
            score_modifier = -4
        if answer in question or question in answer:
            score_modifier = -3
        return score_modifier
    except Exception as exception:
        print(str(exception))
        return score_modifier

'''
Credit if a sentence ends in a punctuation.
'''
def answer_end_in_punctuation(index, question, answer):
    try:
        score_modifier = 0

        sentence_end_with_punctuation = False
        for end_sentence_punctuation in end_sentence_punctuations:
            if answer[-1] == end_sentence_punctuation:
                sentence_end_with_punctuation = True

        if sentence_end_with_punctuation:
            score_modifier = 1
        else:
            score_modifier = 0
        return score_modifier
    except Exception as exception:
        print(str(exception))
        return score_modifier

'''
Penalize if link at the end of an answer ends with '='
'''
def answer_ends_in_equals(index, question, answer):
    try:
        score_modifier = 0

        if answer[-1] == "=":
            score_modifier = -3
        return score_modifier
    except Exception as exception:
        print(str(exception))
        return score_modifier

'''
Penalize if sentence includes <unk> or 'unk' token
'''
def unk_checker(index, question, answer):
    try:
        score_modifier = 0

        if '<unk>' in answer or '_unk' in answer:
            score_modifier = -4
        return score_modifier
    except Exception as exception:
        print(str(exception))
        return score_modifier

'''
Penalize if there are more badly formatted links than good links
'''
def messedup_link(index, question, answer):
    try:
        score_modifier = 0
        badlinks = re.findall(r'\[.*?\]\s?\(',answer)
        goodlinks = re.findall(r'\[.*?\]\s?\(.*?\)',answer)
        if len(badlinks) > len(goodlinks):
            score_modifier = -3
        else:
            score_modifier = 0
        return score_modifier
    except Exception as exception:
        print(str(exception))
        return score_modifier

'''
Penalize if answer contains any of the words considered as bad response
'''
def bad_response(index, question, answer):
    try:
        score_modifier = 0

        for bad_response in bad_responses:
            if bad_response in answer:
                score_modifier = -10
        return score_modifier
    except Exception as exception:
        print(str(exception))
        return score_modifier

'''
Calculate score for all answers based on question itself and heuristically calculated scoring algorithms
'''
def score_answers(question, answers):
    starting_score = 10 #same for all answers to start with

    # scoring functions to run
    functions = [
        answer_echo,
        answer_echo_question,
        is_answer_similar_to_question,
        answer_end_in_punctuation,
        answer_ends_in_equals,
        unk_checker,
        messedup_link,
        bad_response
    ]

    scores = {'score': [], 'score_modifiers': []}

    # Iterate thru answers, apply every scoring function
    for i, answer in enumerate(answers):
        score_modifiers = [function(i+1, question, answer) for function in functions]
        scores['score'].append(starting_score + sum(score_modifiers))
        scores['score_modifiers'].append(score_modifiers)

    # Return score
    Return scores
```

Appendix B

About the Company

Singularity Design, a start-up primarily based in Hannover, is a digital consultancy for marketing and transformation that deals with innovations and proposition design. Singularity Design combines the idea of artificial intelligence with business model design to build strategies against challengers and establish the business model through marketing tools. Singularity is a term coined by Ray Kurzweil, a renowned Google Futurist, and refers to the human pursuit of efficiency, which currently produces methods of artificial intelligence and has a far-reaching impact on people, society, businesses.

Design, especially Design Thinking, enables consultants, creative professionals, employees, etc. to develop and adapt products and ideas based on empathy to people.

Singularity Design creates suitable, competitive, and sometimes futuristic solutions without missing the right time, nor the need of the customer or end customer.

Thesis Advisor: Sofia Trojanowska
Email: sofiatrojanowska@singularitydesign.de
Phone: +49 151 61814053
Web: www. singularitydesign.de
Current Address: In Heldern 3,
31832 Springe
Hannover, Germany
Soon to be situated at: Factory Berlin Mitte,
Rheinsberger Str. 76/77,
10115 Berlin.

Appendix C

Bibliography

[1] Y. Wu, M. Schuster, Z. Chen, Q.V. Le, ‘‘Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation’’, 2016.

[2] N. Ljubesiˇ, D. Fiser, ‘‘A Global Analysis of Emoji Usage’’, 2016.

[3] Jason Brownlee ‘‘A Gentle Introduction to Neural Machine Translation’’, 2017

[4] P. Bii, "Chatbot technology: A possible means of unlocking student potential to learn how to learn", Educational Research, vol. 4, no. 2, pp. 218-221, 2013.

[5] D. Pintado, V. Sanchez, E. Adarve, M. Mata, Z. Gogebakan, B. Cabuk, C. Chiu, J. Zhan, L. Gewali, "Deep Learning Based Shopping Assistant for The Visually Impaired", Consumer Electronics (ICCE) 2019 IEEE International Conference on, pp. 1-6, 2019.

[6] N. Albayrak, A. Özdemir, E. Zeydan ‘‘An overview of artificial intelligence based chatbots and an example chatbot application’’, 2018 26th Signal Processing and Communications Applications Conference (SIU)

[7] K. Papineni, S. Roukos, T. Ward, W. Jing Zhu ‘‘BLEU: A Method for Automatic Evaluation of Machine Translation’’ Association for Computational Linguistics, (ACL), Philadelphia, July 2002, pp. 311-318.

[8] E. Arisoy, A. Sethy, B. Ramabhadran, S. Chen, ‘‘Bidirectional recurrent neural network language models for automatic speech recognition’’, 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)

[9] M. Yao, A. Zhou, M. Jia, ‘‘Applied Artificial Intelligence: A Handbook For Business Leaders’’, 2018

[10] A. M. Rahman; A. A. Mamun; A. Islam, ‘‘Programming challenges of chatbot: Current and future prospective’’, 2017 IEEE Region 10 Humanitarian Technology Conference (R10-HTC)

[11] Arshiya Begum, Farheen Fatima, Asfia Sabahath ‘‘Implementation of Deep Learning Algorithm with Perceptron using TenzorFlow Library’’ 2019 International Conference on Communication and Signal Processing (ICCSP)

[12] H. Bing, ‘‘Statistical Machine Translation Algorithm Based on Improved Neural Network’’,2017 International Conference on Robots & Intelligent System (ICRIS)

[13] Jan Baer – ‘The 6 Critical Chatbot Statistics for 2018’ https://www.convinceandconvert.com/digital-marketing/6-critical-chatbot-statistics-for-2018/ Online. Accessed: 9 July 2019

[14] BLEU - https://en.wikipedia.org/wiki/BLEU, May Li, ‘‘Deep Learning Chatbot’’, https://github.com/mayli10/deep-learning-chatbot Online. Accessed: 9 June 2019

[15] Introducing JSON - https://www.json.org/, Machine Learning - https://in.mathworks.com/discovery/machine-learning.html?s_tid=srchtitle Online. Accessed: 9 July 2019

[16] TensorFlow https://github.com/tensorflow Online. Accessed: 14 June 2019

[17] Deep Learning - https://www.mathworks.com/discovery/deep-learning.html Online. Accessed: 9 July 2019

[18] Artificial Intelligence - https://en.wikipedia.org/wiki/Artificial_intelligence https://en.wikipedia.org/wiki/Artificial_intelligence

[19] Suresh Nambiar, AI Wizards - ‘‘Machine learning techniques’’, 2018. Online. Accessed: 9 June 2019

[20] Sendtex ‘Creating a Chatbot with Deep Learning, Python, and TensorFlow’ https://pythonprogramming.net/chatbot-deep-learning-python-tensorflow/, Daniel Kukiela ‘Neural Machine Translation’ https://github.com/daniel-kukiela/nmt/tree/master/nmt Online. Accessed: 13 July 2019

[21] Word Embeddings - https://www.shanelynn.ie/get-busy-with-word-embeddings-introduction/, Perplexity - https://en.wikipedia.org/wiki/Perplexity Online. Accessed: 9 July 2019

[22] X. Du, Y. Cai, S. Wang, L. Zhang ‘‘Overview of Deep Learning’’ 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC), 2016

[23] R. Mehmood, F. Alam, N. N. Albogami, I. Katib, A. Albeshri, S. M. Altowaijri, "UTiLearn: A Personalized Ubiquitous Teaching and Learning System for Smart Societies", Access IEEE, vol. 5, pp. 2615-2635, 2017.

[24] N. S. Chauhan, ‘‘Build Your First Chatbot Using Python & NLTK’’, 2019. Reddit Data Structure https://en.wikipedia.org/wiki/Reddit Online. Accessed: 9 June 2019

[25] MathWorks – ‘‘Statistics and Machine Learning Toolbox’’ https://la.mathworks.com/help/stats/machine-learning-in-matlab.html. Online. Accessed: 9 June 2019

[26] Robert Dzisevič, Dmitrij Šešok, "Text Classification using Different Feature Extraction Approaches", Electrical Electronic and Information Sciences (eStream) 2019 Open Conference of, pp. 1-4, 2019.

[27] M. Verleger, J. Pembridge ‘‘A Pilot Study Integrating an AI-driven Chatbot in an Introductory Programming Course’’, 2018 IEEE Frontiers in Education Conference (FIE)

[28] B. A. Shawar, E Atwell, "Chatbots: are they really useful?", vol. 22, no. 1, pp. 29-49, 2007.

[29] Sophia (Robot) - https://en.wikipedia.org/wiki/Sophia_(robot), Convolutional Neural Networks https://en.wikipedia.org/wiki/Convolutional_neural_network Online. Accessed: 9 July 2019,

[30] M. Washington, ‘‘An Introduction to the Microsoft Bot Framework: Create Facebook and Skype Chatbots using Microsoft Visual Studio and C#’’, 2016

[31] S. Raj, ‘‘Building Chatbots with Python: Using Natural Language Processing and Machine Learning’’, 2018

[32] Asimov’s Three Laws of Robotics - https://en.wikipedia.org/wiki/Three_Laws_of_Robotics Online. Accessed: 9 July 2019

[33] A. Shevat, ‘‘Designing Bots: Creating Conversational Experiences’’, 2017

[34] H. Isahara, ‘‘Resource-based Natural Language Processing’’, 2007 International Conference on Natural Language Processing and Knowledge Engineering