

Rayan: A Personal Assistant Based on Human-Computer Interaction

Hovhannes Hakobyan

Computer Science with Industrial Placement Year

Candidate number: 164848

School of Engineering and Informatics

Supervisor: Dr. Julie Weeds

April 2019

Statement of Originality

This report is submitted as part of requirement for the degree BSc Computer Science at the University of Sussex. It is the product of my own labor except where specifically stated in the text. The report may be freely copied and distributed provided the source is acknowledged.

Acknowledgements

The author wishes to thank Dr. Julie Weeds for the continuous feedback and advice whilst supervising this project and Rasa community for providing advice on how to implement the distributed application.

Abstract

A chatbot is a conversational system. Based on human computer Interactions via natural conversational language, the chatbot attempts to analyze the user's requests to extract relevant entities. From a technological point of view, chatbots are only question answering systems driven by natural language processing (NLP). The ability to identify the user's intent in different languages, helps to enhance the user experience. In Cognitive Science, machine learned conversational patterns from different transcribed corpuses have been used to create a range of chatbots speaking various languages and sublanguages to streamline interactions between people and services, by doing so, they can enhance user/customer experience as well as supporting different businesses in reducing the typical cost of customer services. The main areas covered in this paper include, Introducing and comparing NLP driven approaches for creating a chatbot and matters related to personal and public interest and detailing the design and development process involved for creating a chatbot.

Contents

Statement of Originality.....	2
Acknowledgements.....	3
Abstract.....	4
1.0 Introduction	7
1.1 Objectives.....	8
1.1.1 Primary objectives.....	8
1.1.2 Secondary Objectives.....	8
2.0 Professional consideration.....	9
2.1 Public Interest	9
2.2 Professional Competence & Integrity	9
2.3 Ethical review	9
3.0 Requirement analysis.....	10
3.1 Target Users	10
3.2 Design of the System	10
3.3 Functional Requirements.....	11
3.4 Non-Functional Requirements.....	11
4.0 System Overview.....	12
4.1 Tokenization.....	13
4.2 Semantic Parsing.....	13
4.3 Part of Speech Tagging.....	14
4.4 Named Entity Recognition	15
4.5 GloVe model.....	15
5.0 ML components of my bot.....	16
5.1 NLTK vs spaCy.....	16
5.2 Recurrent Neural Network Components	18
5.2.1 Feed-Forward Neural Network vs RNN.....	18
5.2.2 Two issues with RNN.....	20
5.2.3 Long-Short Term Memory.....	20
6.0 Building the Bot with Rasa Components	21
6.1 Rasa	21
6.2 Rasa NLU	21
6.2.1 Building the NLU	23

6.3 Rasa Core	25
6.3.1 Building the Core.....	26
6.4 Interactive Learning	30
7.0 Related Work	34
7.1 Dependency Parser	34
7.2 Seq2Seq model	35
8.0 Design.....	36
8.1 Slack API	36
9.0 Testing and evaluation.....	38
9.1 Testing.....	38
9.2 Evaluation	39
10.0 Conclusion and Further work.....	39
11.0 Appendices.....	40
11.1 Project Plan	40
11.2 Interim log.....	41
11.3 How to run	43
12.0 References	43

1.0 Introduction

This project will explore the technology behind human-machine conversation based NLP. In Data-science, NLP is a technological process that allows computers to derive meaning from text inputs. As such, it tries to understand the intent of the textual document. The key research of this project is to create a chatbot that will interact with humans and as such, based on their queries it will guide, chat and notify every user about their scheduled meetings daily or show the temperature at any location. For example, from the input text 'when am I meeting John?' or 'when is my next meeting?', it will first look at words like 'me' or 'I' which are considered as pronouns in which the user is referring to him/herself in the first person. In this scenario, the user is asking about the time, or perhaps the date of the meeting, and by identifying the adverb 'when', this can be understood as 'at or during the time of'. Following the correlation of those words, the chatbot will generate a query which will follow the connection of dates from time of asking, to the next scheduled 'meeting' from the database.

The main goal of this chatbot is to quickly provide answers to user's queries, which can be examples of asking the conditions of the weather in current or other areas or to be reminded about upcoming private meetings with time and place.

A chatbot can have a conversation with a user/customer but can be imperfect in its delivery. This is due to ambiguities in human expression. For example, in the sentence 'When is the meeting?' the chatbot may not struggle to evaluate the intent of the sentence where the word 'when' is concerned but may not be able to identify *which* 'meeting' is being referred to as this is not explicit. A sentence which could be better evaluated in this case would be 'when is my next meeting?' – this could find the closest scheduled meeting from the given time. Another example would be 'when am I meeting John?' - in this sentence it is clearer that a user is asking about the 'time' when they are meeting a person named 'John' but it could still be considered ambiguous because the same sentence could refer to many people named 'John'. Thus, it is worth mentioning the surname of this person as well.

This Project will also help investigate technologies behind different NLP techniques, which will not require many resources e.g. data, computation or power to integrate the chatbot and will comparably perform well to other known integrated NLP/AI chatbots in communication changes as opposed to with another human. In addition, this project will help analyze word correlations in a vector-space where each word is given a numerical value (vector) and the distance between these vectors to show how one word is mostly used within a context in relation to other words. In addition, this project will describe the effects of the Reinforcement Learning on conversations between a chatbot and user that are applied to this project.

1.1 Objectives

1.1.1 Primary objectives

- The main objective is to create a chatbot called 'Rayan' that will generate responses to the user's queries.
- The first objective is to create a Natural Language Understanding model that will derive useful information from the user's input text for use in the core model.
- The second objective is to establish a core model for the chatbot, driven by Natural Language approaches. The core will handle the dialogues between the user and chatbot by using the data generated from the Natural Language Understanding model to generate appropriate responses for the user.
- Thirdly, to apply Interactive-Reinforcement-Learning, that based on previous history of interaction between user and bot, will learn and improve the performance of the core model.
- The final but not the main objective is answering questions even if the given input-text is not consistent with the main objectives of the chatbot or is completely ambiguous for the bot to perform information retrieval. An example of a question that would be considered out of scope would be "what can I eat today?" - this question would be considered as irrelevant because the bot's main objective is to assist in scheduling daily events or checking the weather. However, if a user were to ask the chatbot about the food or restaurant, the answer from the chatbot would be "I am sorry, I am not able to assist you" or if the question has offensive or inappropriate connotations, then this kind of question will be either avoided or responded to in an appropriate manner.

1.1.2 Secondary Objectives

- A user-Interface that will show the interaction between user and bot along with input space which will let user to interact with the chatbot.
- Implement and compare different NLP driven approaches for creating a chatbot. There are various ways to create a chatbot. For example, the chatbot could be created through dependency parsing, named entity recognition or through AI-driven machine learning approaches for basic question answering.

2.0 Professional consideration

This project follows the code of conduct published by the chartered institute for IT [1].

2.1 Public Interest

This project does not have any risk to privacy, security, public health or the environment. This is because this project is based on publicly available resources. The information is used to test and compare the usability of different methods, such as publicly available python-based libraries “libs”, and various other libraries used in conjunction with Python programming language. However, since my project is based on human-computer interaction, the integrity between chatbot and any other person will be taken into consideration to prevent any harmful or offensive queries generated through this communication.

2.2 Professional Competence & Integrity

This project is based on implementing different interaction techniques between humans and computers driven by Natural Language Processing. Undertaking this project has helped me to develop academic knowledge in cognitive science and has also supported me in furthering my awareness of the procedures used in the developing field of cognitive science.

2.3 Ethical review

All the research involved in creating this project has been undertaken in accordance with the highest ethical standards and were discussed between my academic supervisor and myself. However, for all of the tests that are run to satisfy the user experience, or for the general purpose of collecting data from human participants, users involved in these tests will be obliged to complete a compliance form to ensure that the project is in line with all twelve points of the Ethical Compliance Form for UG and PGT Projects [2].

After investigating the aforementioned form, I can confirm that this project mainly associates with 4 of the points that were defined in the form – these are points 3, 6, 8 and 9:

- Number 3 refers to a section where ‘all the participants should explicitly state that they agreed to take part and that their data could be used in this project’ [2]. There are no participants that will be obliged to take a test without them agreeing to those terms.
- Concerning number 6 - ‘No participant was under the age 18’ [2], there will be no participants that will be asked to test this project if they are under age.
- Number 8 – ‘Neither I nor my supervisor are in position of authority or influence over any of the participants’ [2] – there will be no such participant. Most of the participants will be chosen randomly based on their university subject areas which also ensure that they have

relevant knowledge about Cognitive Science and as a result, will be able to discuss the functionality of this project and give better ideas for its improvements.

- Finally, 9 - 'All participants were informed that they could withdraw at any time'. All the tests are expected to be based on queries generated by the users themselves, and so all of the participants will be aware that they are able to withdraw or decline to participate at any time.

3.0 Requirement analysis

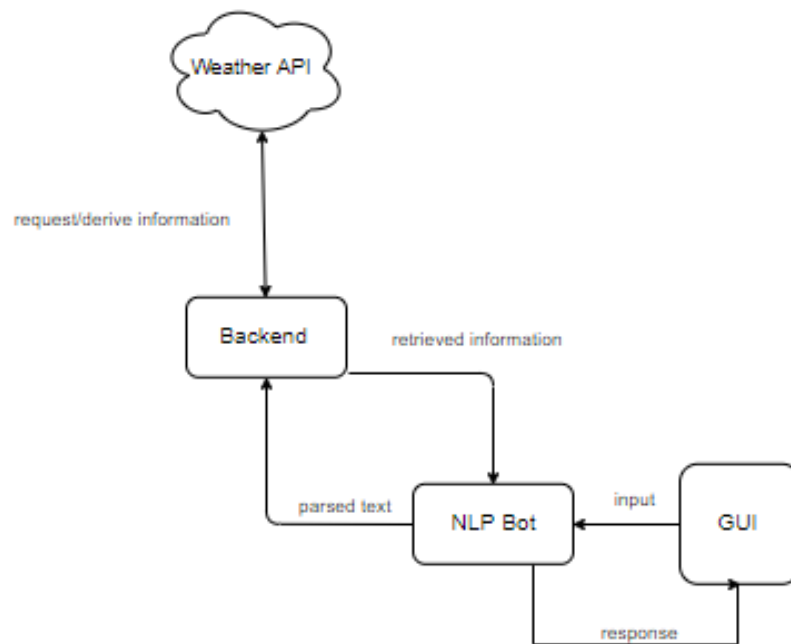
3.1 Target Users

Chatbots are a great way to automate the user's needs and assist them throughout a long day of work or providing information about the weather around the world. A chatbot can capture the most important pieces of information about a user and their intents via NLP, and that is why chatbots are able to mimic real human conversation which helps satisfy the desired output of users. For example, for questions in which a user will be asking about a date or a specific time of a specific event, they would be guided or given a reminder based on those queries.

3.2 Design of the System

As mentioned earlier, the main objectives are to create natural language understanding and core models that together will form the bot. However, this project is also based on other components such as user environment, backend (actions, servers) and a relational database. With relation to the backend, the project will utilize a local server, that through API's will connect to the online servers relevant to the information that the user asks to retrieve e.g. weather condition.

A graphical user interface will be provided for users to type their input-texts, after which the output text responses will be sent and shown in a chat environment. This is depicted in the System Design pipeline below.



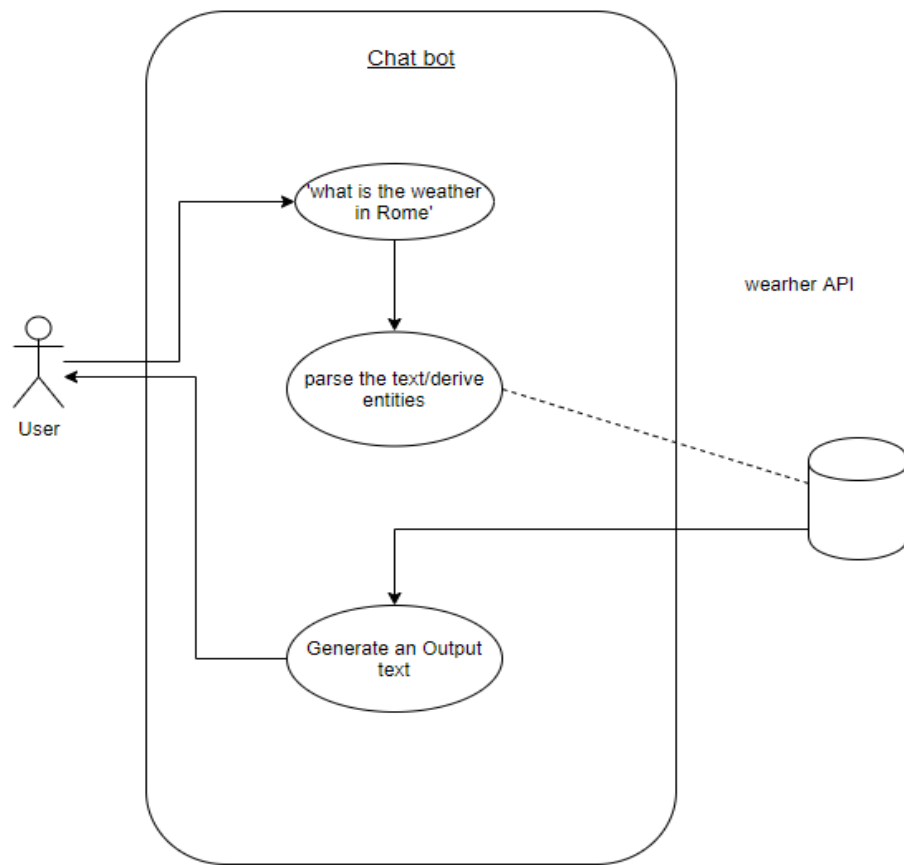
3.3 Functional Requirements

- The main functional requirement is to create a graphical environment where a user can communicate with a chatbot by inputting text.
- A secondary requirement is to create a server on which the chatbot can run live for further interactions with a user
- Third requirement is to connect online APIs' for information retrieval e.g. weather in specific locations.

3.4 Non-Functional Requirements

- In order to test the bot, we would need an example of a local relational database of extracted dates along with their scheduled events.
- Another non-functional requirement would be to connect the bot to the different API's that will let the bot derive information about the locations e.g. restaurants near the user.

A use case diagram is shown below which is based on the functional-requirements listed above.



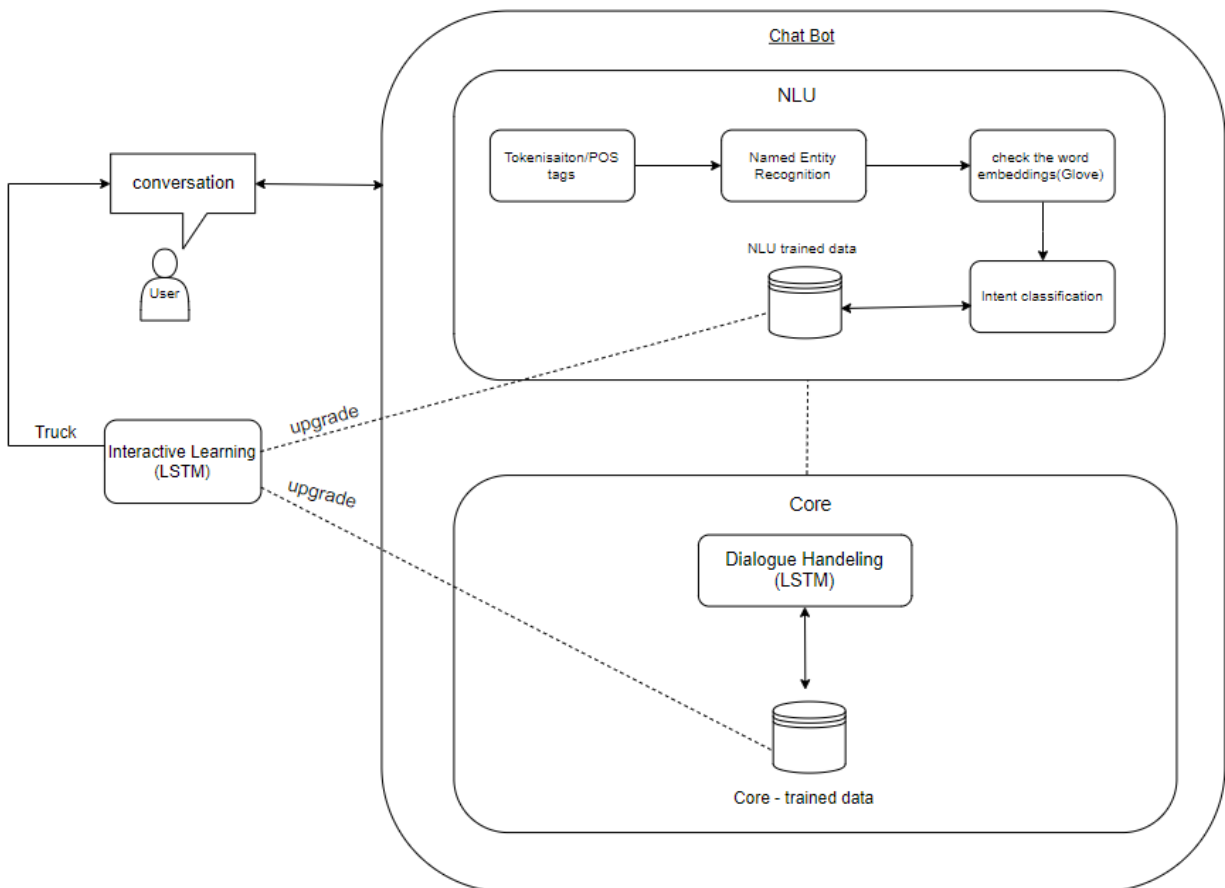
4.0 System Overview

Following the research covered for this Project, I have come across different ways that can be implemented to create a chat-bot. Such as, Rule-based if-else conditions to self-learning approaches [17].

- **Rule-based approach:** a bot can only answer the questions that it is trained on. The bots that are created by a rule-based approaches can answer simple queries but most of the time they will have issues to manage complex ones because the bot can only reply to specific queries that were predefined in the rules used to train the bot.
- **Self-learning approach:** the bots that run with this approach are examples of bots that use library of predefined heuristic responses. The chatbot uses messages and the intent of the conversation for selecting the best response from predefined list of answers. The context can be example of the current position of the dialogue tree e.g. all the previous text-messages, slots defining location and time or previously saved variables for example, names or usernames.

For this project, I have chosen to integrate self-learning approach, and to do so I have created Natural Language Understanding (NLU) and Core (dialogue handling) modules that combined will form our chatbot. In the next paragraphs I will discuss in detail the example of NLP and ML components I have used

to build this bot. Following that, I will discuss and show how I have implemented interactive learning to improve the performance of the chatbot and show the resulting evaluations just like shown in the end to end pipeline below.



4.1 Tokenization

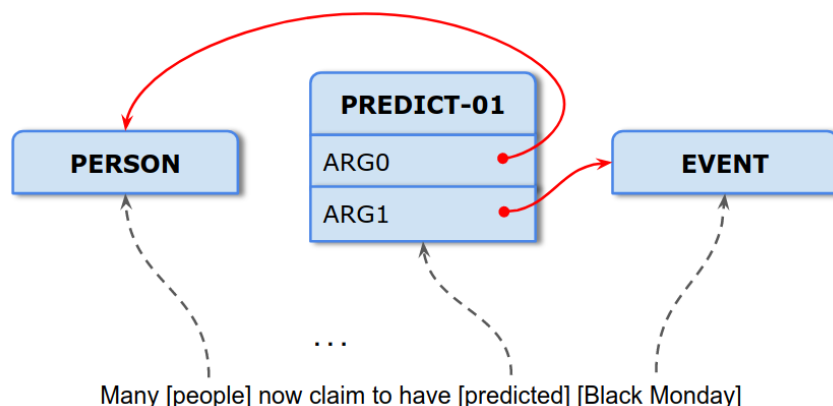
NLP is used in wide range of distributions for context disambiguation; queries must be checked before they can be used by the information mapping part of a system. The tokenizer is usually the first stage of pre-processing used in NLP, a sentence is broken down into smaller components that are called tokens. Choosing the right tokenizer is important as it will affect the subsequent stages in the greater system, such as parsing and indexing [3]. An example of tokenized sentence 'when is my next meeting?.' is provided below.

```
['when', 'is', 'my', 'next', 'meeting', '?', '.']
```

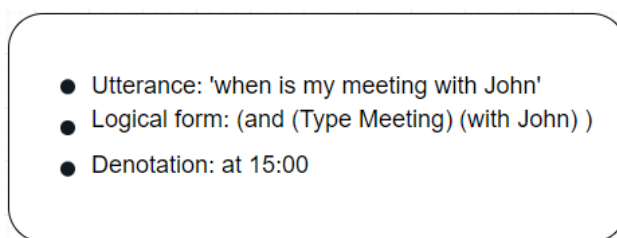
4.2 Semantic Parsing

Parsers are defined to use the grammar of the given language to perform syntactic analysis of all the words used in the sentence or document. Semantic analysis has been used extensively to reach NLU in machines. They derive the syntactic and semantic components from the sentence so that the machine

can determine the nature and function of the sentence. Semantic parsing is another technique in natural language processing - a phrase is parsed into machine-understandable logical form (example below)



Semantic parsing is concerned with making progress on understanding the precise meaning of an utterance before linking it to a logical form, see example below.



Another example of semantic parsing would be through paraphrasing described by Berant and Liang, 2009 [4]. In their example semantic parsing is done through assigning word/content to a domain-independent intermediate logical form. In other words, their approach to the problem is from the other end. Other than doing semantic analysis and assigning logical values to each word in the sentence, they paraphrase the logical sense of the phrase before correlating the words into their logical forms.

4.3 Part of Speech Tagging

Part of Speech (POS) tagging is the measure of assigning a POS tag to every word in a sentence. POS tagging is example of information extraction in grammatical form. The examples of POS tags are verb, adverb, noun, pronoun, etc. However, most of words occurring in text/sentence have ambiguity associated with them in terms of their part of speech [3]. For example, word 'power' can be treated as a 'noun' or a 'verb'. Thus, the whole phrase or patterns of different phrases should also be recognized so that words in different phrases will be POS tagged based not only on their definitions but also context within the sentence. The process of part of speech tagging is done through reading the input sentence. Then tokenize the sentence into words. After tokenization, Suffix analysis and prefix analysis is also used to tag each word of sentence correctly. Then the pre-trained corpus is used to tag each word in the sentence as noun, verb, conjunction, number tag etc [3]. Then after analysis, the accuracy of output. For example, with the sentence 'what is the weather like in France' we would get POS tags as shown in the table below.

what	what	Relative pronoun
is	be	Verb, 3rd person singular present
the	the	Determiner
weather	weather	Noun, singular or mass
like	like	Preposition or subordinating conjunction
in	in	Preposition or subordinating conjunction
france	france	

4.4 Named Entity Recognition

In NLP, named-entity recognition (NER) is an information extraction technique that primarily focuses on classifying named entity mentions from unstructured texts where name-entities can be defined as real-world objects such as persona, product, location, organizations, etc. To identify the named entities, the system attempts to understand the specifics of words, which will be very precise when it comes to analyzing a query or document. However, usually named entities have a connection with one another. Name-entity recognizers can differ based on their rules of computation. For example, some techniques use a collection of regular expressions (a character matching process), and others use machine-learned supervised algorithms where data is labeled to develop a statistical model. However, training a supervised named-entity recognizer requires a large collection of labelled data of queries. As discussed in IJIRSET 2017 [7], rule-based recognizers are simple, but it is significantly harder to achieve high accuracy without involving machine learning approaches with the NLP.

In supervised machine learning, the data is labelled before training it on different corpus-based queries - it creates an understanding of what to expect from a query other than analyzing a text-based document in a limited extent where it can analyze the semantics but struggles to understand human-expressions such as figures of speech. However, even with supervised learning it still requires a massive amount of data, but many languages do not have annotations in them, and cannot be treated in the same grammatical order as in other languages. The IJIRSET 2017 [7] paper also discusses unsupervised-learning approaches to solve this kind of problem. In this solution, there are many approaches (i.e. Generative based, Retrieval based, Heuristic based, etc.) used to build conversational bots or dialogue systems and each of these techniques make use of NER somewhere in their respective pipeline as it is one of the most important modules in building the conversational bots.

4.5 GloVe model

The Global Vectors for Word Representation (GLoVe) is an example of unsupervised learning algorithm that aims to create word-vectors based on how frequently they appear together in large text corpora, [23]. GloVe model is known as count-based model that constructs matrix based on the word co-occurrences stored in rows, which indicate on how frequently they have appeared in the context and

stored in columns. For example, word vector matrix of words ['when', 'my', 'is', 'meeting', 'today', 'next', '?'] would look like this.

counts	when	My	is	meeting	today	next	?
when	0	0	1	0	0	0	0
my	0	0	0	1	0	0	0
is	2	1	0	1	1	1	0
meeting	0	2	1	0	1	1	1
today	1	0	0	0	0	0	1
next	0	1	1	2	0	0	0
?	1	0	0	2	1	0	0

The words that have higher number with given word means they have closer points in the vector-space

The matrix above is simple example of how frequently each 'word' is acquired in the context. In this case the context is small, but in most cases, we expect the context to be large, such that, after preprocessing e.g. evaluating distance vectors between the words, it will yield a lower-dimensional matrix of features where each row represents vector-presentation for each word [23]. There is a tradeoff between information captured by word vectors and the density of the representation of that information. Smaller dimensionalities will represent the information contained in the underlying corpus more compactly and will make it 'easier' for algorithms to leverage that information. In contrast, larger dimensionalities will capture more information but are 'harder' to use. Therefore, GloVe model reduces dimensionality in the vector-space from 'v' to 'n', where 'v' is vocabulary size and 'n' is the number of dimensions that usually ranges between 300 to 400.

5.0 ML components of my bot

5.1 NLTK vs spaCy

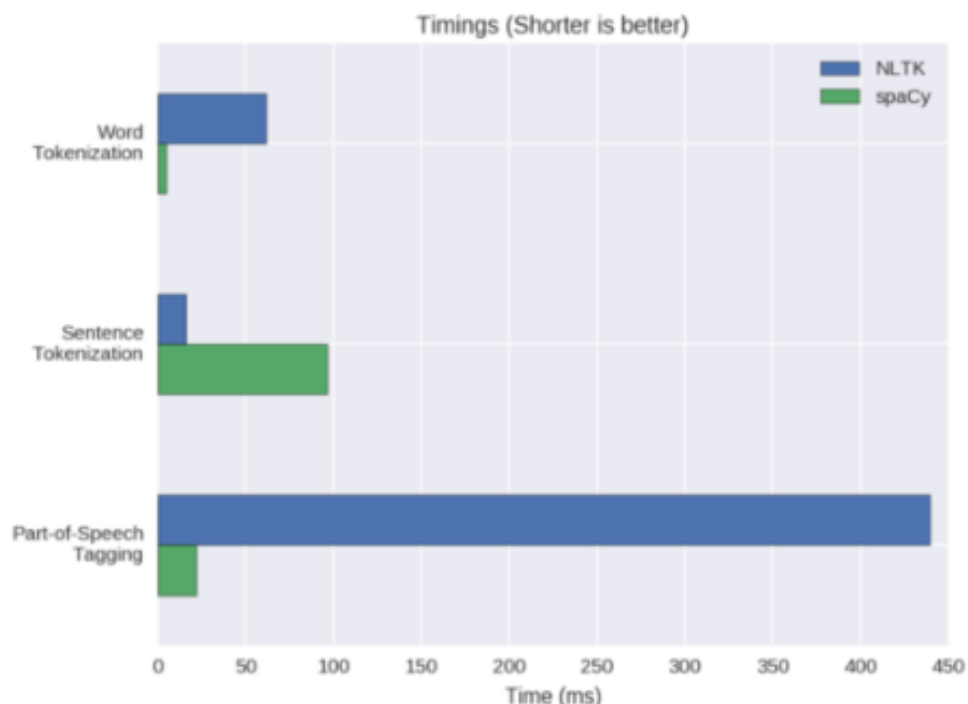
NLTK (Natural Language Toolkit) and spaCy are leading platforms for building NLP based applications on Python. Both NLTK and Spacy provide interfaces to over 50 corpora and lexical libraries such as WordNet, along with various text processing libraries for tokenization, classification, tagging, dependency parsing etc [18].

The main issue with text is that the textual data is a set of strings, meaning that for the Machine learning algorithms to be able and process the textual data, all the data (text) needs to be assigned with some sort of numerical feature (vector) such that, it is possible to perform the task. This is when it comes to text pre-processing. The examples of pre-processing can be, transferring all the textual data into uppercase or lowercase so that the algorithm will not deal with different cases and perform tokenization to transfer strings into list of tokens, tokens then can be used to create structured texts described in section 4.3.

After the initial pre-processing is done, we can then create bag of words, the bag of words can be used to check the occurrences of the given word within the document. One of the main intentions of creating bag of words is that the bag of words can define similar documents by their contents, or they can give useful information about the document from the content itself. For example, let's assume we have a dictionary of {Learning, is, the, great, not} and we want to vectorize the text "Learning is great" then we would get vectors defined like this (1,1,0,1,0) each number representing the number of time the given word from the dictionary has appeared in the text.

SpaCy have statistical models for 7 languages which are English, German, Spanish, French, Portuguese, Italian and Dutch, while NLTK supports various languages. NLTK is known to use string processing, meaning it takes strings as input and returns a string or list of strings as output. Whereas, spaCy can also return a document object that contains objects as words and sentences [19]. SpaCy also allows word vectorization that is used to derive word embeddings (vectors) described in section 4.5.

SpaCy uses latest and most known algorithms [19], its performance is usually better as compared to NLTK. The following figure describes the comparison of performance between spaCy and NLTK with the time they took to fulfil the given tasks of word/sentence tokenization and POS-tagging. The performance of spaCy is better, but not when it comes to sentence tokenization. This is because NLTK and spaCy don't use the same approach. For example, NLTK attempts to split the text into sentences [20]. In contrast, spaCy constructs a syntactic tree for each sentence, that tends to give more information about the text and essentially works more efficiently than the most toolkits [19].



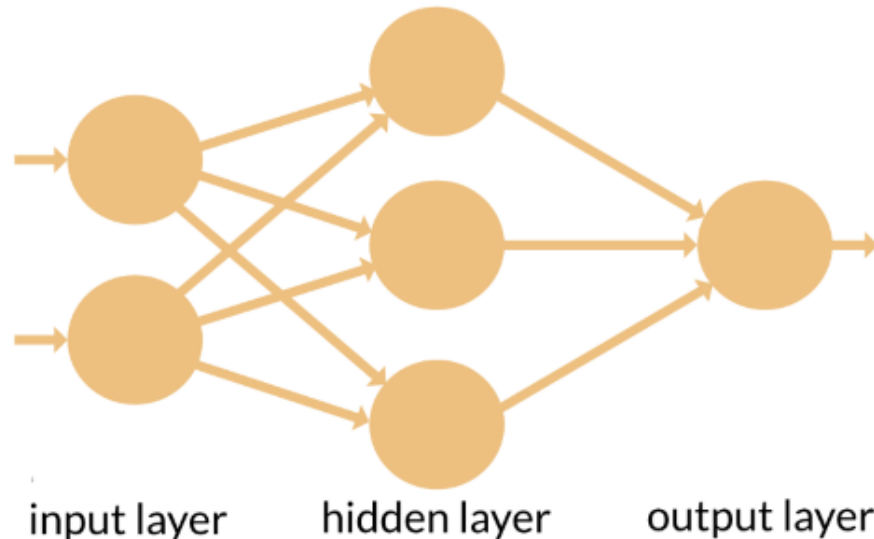
5.2 Recurrent Neural Network Components

Recurrent Neural Networks (RNN) are powerful artificial neural network algorithms. The idea of RNNs is to make use of sequential information based on original structure of neural network, where the inputs (words in sequence) and outputs are independent from each other. The idea of RNN is relatively old according to [8], just like many other deep learning algorithms, they were initially created in the 1980's but could not be implemented at that time due to lack of computational power [8]. Because of their internal memory, the RNN's can help to analyze the grammatical and semantic correctness from a sentence or document by allowing us to score arbitrary sentences based on how likely they are to occur in the real world, which enables the RNN model to be very precise when predicting what is coming next.

The RNN is primarily used in NLP to complete tasks like training language models or text generation. Recurrent networks can scale to much longer sequences than any other networks without sequence-based specialization [9].

5.2.1 Feed-Forward Neural Network vs RNN

Before giving examples of how RNNs channel the information, we will discuss the difference between RNN's and Feed-Forward neural networks. Recurrent neural networks and feed-forward neural networks are both categorized as neural networks because of the way they channel information. In a feed-forward neural network the information moves in one direction starting from the Input layer through hidden layers all the way to the Output layer. Check the example diagram below.



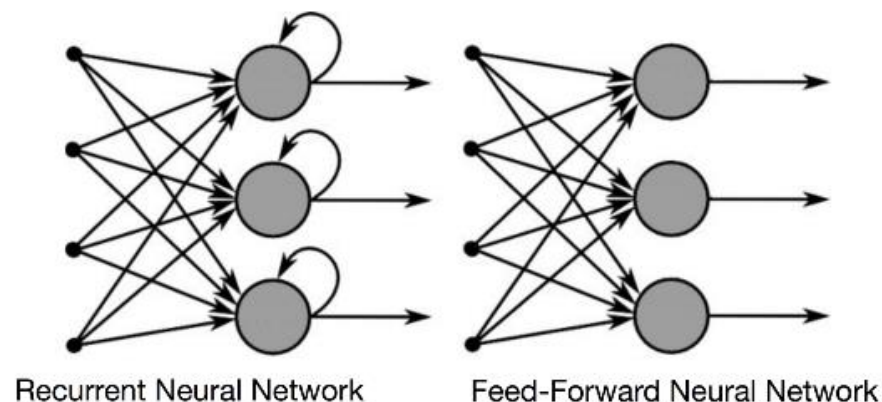
As we can see from the diagram, the connections between the nodes do not form a cycle. Thus, it is very different to recurrent neural networks. The feed-forward neural network was the first and simplest type of artificial neural network devised [11]. One of the simplest examples of a neural network is a single-layer perceptron.

In an RNN, the information cycles via a loop, before making a decision it takes into consideration the current input and all the example of information it has learned from the other inputs it has received previously.

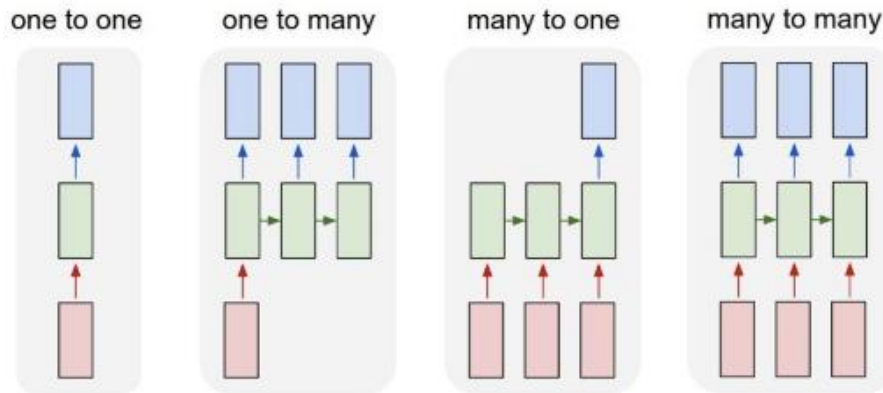
Such sharing of information can be particularly important when it comes to analyzing a text file. Specific pieces of information can occur at multiple positions within the sequence [9]. For example, consider two different example of sentences “I visited Rome in 2019” and “in 2019 I went to Rome” now let’s assign a task to read each sentence and find the year that this person has visited ‘Rome’ for our Machine Learning model, we then would want it to keep year ‘2019’ from sequence of words in sentences even if it has appeared as the second or sixth word.

Suppose we have successfully trained a feed-forward neural network that takes sentences of fixed length. The feed-forward NN then should analyze separate parameters for each input feature which means when the information passes through each node the language or the meaning of the sentence at the given position would be analyzed separately from the other nodes. For example, let’s assume we have a normal feed-forward neural network and pass it the word ‘neural’ as an input, which will later process it character by character. By doing so, before reaching the character ‘r’ it would have already forgotten the characters ‘n’, ‘e’ and ‘u’, this makes it almost impossible for feed-forward NN to predict what character would come next as it will never remember the outputs of the previous node. By comparison, as mentioned before the recurrent neural network cycles information through a loop where it keeps the weights of the input for several time steps [9], because of its internal memory, the RNN copies the output and loops it back into the network. It is essential for the RNN to have at least two inputs which are the present and recent past, because the sequence of data contains crucial information regarding what can come next.

The figure below show an example of how the information flow between RNN and feed-forward neural network.



The feed-forward NN just like any other Deep Learning algorithms, assigns a weights matrix to its inputs to produce the output. Note the RNN apply weights both to the current and to the previous inputs. Furthermore, both algorithms upgrade their weights through gradient decent and backpropagation through time. It is also worth mentioning that feed-forward NNs map only one input to one output while the RNNs can map one to many, many to many and many to one as shown in the example in the diagram below.



5.2.2 Two issues with RNN

There are two known major issues that tend to happen during the process of training standard RNNs, both of which will be discussed in later sections but first we need to understand how gradients work in neural networks.

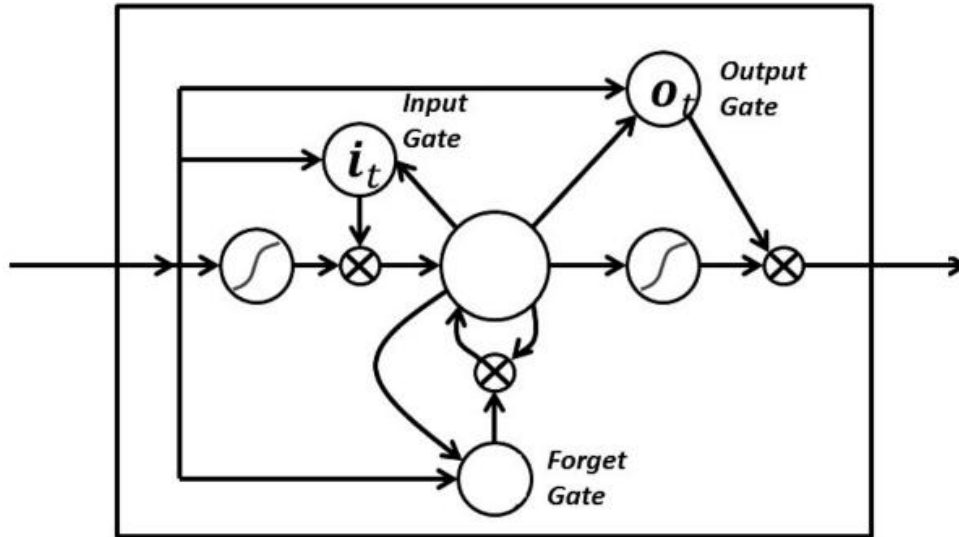
The gradient measures the output of a function by changing the inputs. The gradient can be also expressed as the example of slope of the function. The higher the gradient, the steeper the slope and the faster a model can learn. Meaning that the gradient simply measures all the changes in all weights about the change in error.

The first known obstacle for RNN's are the Exploding Gradients. In deep networks or recurrent neural networks, gradient explosion can occur when a value of the error gradient is updated by even larger number which in turn, results in larger updates to the network weights and as a result, we get an unstable network that is unable to learn from over long sequences of data [13]. One of the known approaches to fix exploding gradients is by using Long-Short Term Memory (LSTM) that we will discuss in the next paragraph.

The second issue is Vanishing Gradients. The Vanishing Gradients happen when the values of the gradient are too small. Thus, the RNN model stops learning or takes way too long. This issue was also resolved by the LSTM.

5.2.3 Long-Short Term Memory

Long-Short Term Memory (LSTM) networks are an extension of recurrent neural networks, which simply spans the memory, meaning that the LSTMs enable the RNNs to remember their inputs over a longer period of time which as we know is essential for RNN to get a good error rate. This is because compared to standard RNNs, LSTMs contain their information in a memory, meaning that during the loop when the RNN passes its current and previous inputs to upgrade the weights at the given time step, the LSTM performs more steps to fulfil the same task [14]. These steps can be expressed as the example of gates in the form of sigmoid which makes analog as it will be given a value that spans from 0 to 1. In an LSTM, we have 3 examples of gates throughout the loop. The 'input' 'forget' and 'output' gates. The input gate helps to determine whether to let the new input in or delete(forget) that information since it has no any crucial value or to let it impact the output at the current time step.



The two main issues with RNN's are then resolved by the LSTM because it enables us to keep the gradient steep enough. Thus, the training relatively short and has higher accuracy.

6.0 Building the Bot with Rasa Components

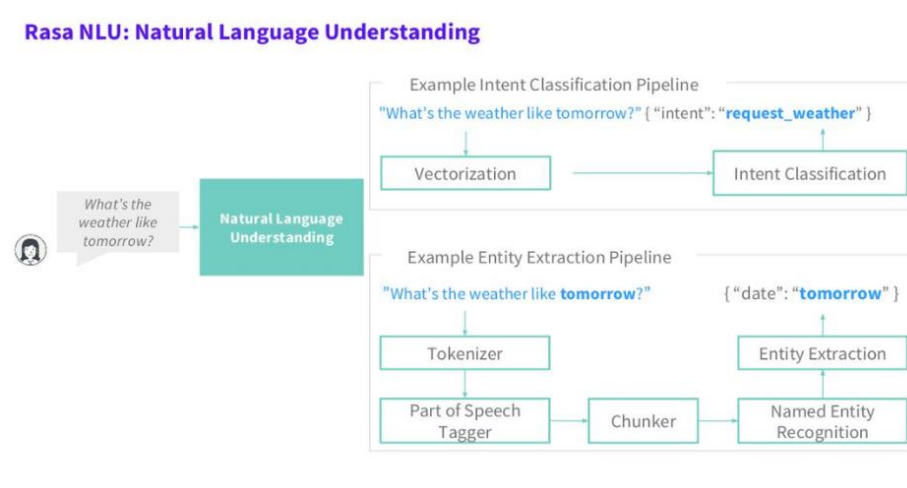
6.1 Rasa

Rasa is a leading open source Python implementation for NLP that integrates a machine learning toolkit which lets developers expand and enhance chatbots beyond answering simple queries. The open source Rasa Stack allows developers all over the world to build conversational AI that requires less computational power or data. Besides its broad abilities, the Rasa Stack allows developers to customize the source code so that it fits the most to the bot that they want to build [21]. Despite its large abilities in the NLP field, rasa stack implements number of sources for building a conversational AI chatbot. In this, Rasa's API uses ideas from scikit-learn and Keras, both libraries are open source and can be implemented independently to perform various machine learning tasks on texts. Rasa's stack has two components, Rasa NLU and Rasa Core which makes it not rule based framework for creating chatbots.

6.2 Rasa NLU

Rasa's NLU is an example of a natural language understanding model. Comprising of coupled modules and combining a number of NLP libraries in a single accessible API. It lets you use some predefined pipelines like spacy, sklearn, tensorflow_embedding, mitie and mitie_sklearn that run on sensible defaults (can be changed) and work well for most use cases [21]. For example, the recommended pipelined parts of speech annotated using the spaCy NLP library (section 5.1). Then the spaCy featuriser looks up a GloVe model (section 4.5) to check the vector for each token (section 4.1) and retrieves them to create a representation

of the whole structure of the sentence. Then, the scikit-learn classifier trains an estimator for the dataset, by default with a multiclass support vector classifier trained with five-fold cross-validation. Following that, the ner_crf trains a conditional random field to recognize the entities in the training data, using the previously created tokens and POS tags as base features. Since each of these components implements the same API, it is easy to swap the GloVe vectors for custom, domain-specific word embeddings, or to use a different machine learning library to train the classifier. There are further components for handling out-of-vocabulary words and many customization options for more advanced uses. The following figure below shows how the intent classification and entity extraction is performed through Rasa's NLU on the example sentence 'what is the weather like tomorrow'.



In the next figure you can see the config file that is passed to the Rasa NLU module to set default preferences by defining the libraries and the language on which they will be used.

```
language: "en"
pipeline: "spacy_sklearn"

we can use the component and configure it separately as below,

language: "en"

pipeline:
- name: "nlp_spacy"
- name: "tokenizer_spacy"
- name: "intent_entity_featurizer_regex"
- name: "intent_featurizer_spacy"
- name: "ner_crf"
- name: "ner_synonyms"
- name: "intent_classifier_sklearn"
```

The same way we can define a different library such as tensorflow_embedding, just like it is defined in the figure below.

```

language: "en"
pipeline: "tensorflow_embedding"

we can configure tensorflow_embedding as below,

language: "en"

pipeline:
- name: "tokenizer_whitespace"
- name: "ner_crf"
- name: "ner_synonyms"
- name: "intent_featurizer_count_vectors"
- name: "intent_classifier_tensorflow_embedding"

```

6.2.1 Building the NLU

Rasa NLU is used for intent classification and entity extraction. For example, taking a sentence like " what is the weather in Italy?" we would get structured data like.

```

{
  "text": "The weather condition in Italy",
  "intent": "inform",
  "entities": [
    {
      "start": 25,
      "end": 30,
      "value": "Italy",
      "entity": "location"
    }
  ]
}

```

To create the NLU model we need to have a config file, that we are going to pass to our Rasa NLU model. It contains the example of libraries which we want it to be using to perform intent classification and entity recognition in English language just like it was described in the previous paragraph for the config file. Then, we would need training data with all the intents listed which will make the bot categorize and perform entity recognition in order to understand the content of the input text, then classify and calculate the error rate of which intent it belongs to. The examples of the training data used can be found in the Image below. Note, most of the data was generated by me to create a enough intents for this chatbot.

```

{
  "text": "I am going to London today and I wonder what is the weather out there?",
  "intent": "inform",
  "entities": [
    {
      "start": 14,
      "end": 20,
      "value": "London",
      "entity": "location"
    }
  ]
},
{
  "text": "I am planning my trip to Amsterdam. What is the weather out there?",
  "intent": "inform",
  "entities": [
    {
      "start": 25,
      "end": 34,
      "value": "Amsterdam",
      "entity": "location"
    }
  ]
},
{
  "text": "Show me the weather in Dublin, please",
  "intent": "inform",
  "entities": [
    {
      "start": 23,
      "end": 29,
      "value": "Dublin",
      "entity": "location"
    }
  ]
},
]

```

Now we are ready to train our NLU model and save it for the later use, by running the example of code below.

```

6  from rasa_nlu import config
7  #from rasa_nlu.converters import load_data
8  from rasa_nlu.training_data import load_data
9  from rasa_nlu.model import Trainer, Interpreter, Metadata
10
11
12  def train(data, config_file, model_dir):
13      training_data = load_data(data)
14      configuration = config.load(config_file)
15      trainer = Trainer(configuration)
16      trainer.train(training_data)
17      model_directory = trainer.persist(model_dir, fixed_model_name='MyNLU')
18
19  def run():
20      interpreter = Interpreter.load('./models/nlu/default/MyNLU')
21      print(interpreter.parse('what is the weather like in rome?'))
22
23  if __name__ == '__main__':
24      train('./testData.json', './config.yml', './models/nlu')
25      run()

```

By running this code above, we make it train and save our NLU model in the directory of './models/nlu' (line 24) so that we can use it every time when we want to without retraining it again. Now let's see how

well our NLU model performs, by parsing it a sentence 'what is the weather like in Rome' (line 21) we would get.

```
{'intent': {'name': 'inform', 'confidence': 0.7335016765825505}, 'entities': [{'start': 28, 'end': 32, 'value': 'rome', 'entity': 'location', 'confidence': 0.6550054846580706, 'extractor': 'ner_crf}]}
```

It has successfully categorized it under intent 'inform' with confidence of 73%, which is the distance metric that indicates how closely the NLU could classify the result into the list of intents (this can be improved by giving more examples for the training data). However, in this specific project we only want the main intent to have higher confidence than the rest of the intents, which in this case is the 'inform' for looking up a weather with entity 'location: rome' to categorized under it. Now, let's say we have not passed a sentence with a name of a city or country. When giving the training data I also pass the entities that must be provided to fulfill the goal of the intent, which in our case is weather check. These entities are called slots, if we are training our bot to perform actions based on user input, we then would need to specify the example of slots we want it to look for. Now let's give it a sentence 'what is the weather like?'

```
'intent': {'name': 'inform', 'confidence': 0.7601223923051812}, 'entities': [],
```

It understands that the given input is of intent 'inform', however, the entities slot is empty which leaves it incomplete. Although, it has successfully finished the task, this is when it comes to the core module that will handle the dialogue discussed in the next paragraphs, where it will understand that the slot is missing and therefore will require the user to add more information.

The Named Entity Recognition in our NLU uses Heuristic approach (section 4.4). In the chatbot, there are several entities that must be identified for intent classification and each entity must be distinguished based on its type as a different entity for different detection logic. Although, it is possible to have many words that belong to the same exact entity. Now, let's presume a different scenario when the sentence is 'when is my meeting'. The word 'meeting' has many synonyms that can be used for the same intent. Such as, assembly, congress, conference, gathering, huddle etc. That is why when passing the training data, we must also pass the values of the entities. For example, in the figure below you can see that from the input text I am assigning the word 'conference' to the value of 'meeting'. After training, whenever the word 'conference' is extracted from the input-text, it will always be recognized as the entity 'meeting'. This way our bot learns how to handle synonyms.

```
{
  "text": "when is my conference",
  "intent": "search_for_meeting",
  "entities": [
    {
      "start": 11,
      "end": 21,
      "value": "meeting",
      "entity": "meetings"
    }
  ]
}
```

6.3 Rasa Core

Rasa core is used to guide the flow of conversation, while the Rasa NLU is used to understand and process the text to extract information e.g. entities. They both can work together but they can also be used independently as separate API's and be customized upon developer's preferences. Rasa Core is an open source chatbot framework to handle contextual conversations. Move beyond simple Q&A pairs and rigid decision trees. Its advanced dialogue management is based on ML and allows for smarter conversations and makes scaling easier [22]. The primary purpose is to help you build contextual, layered conversations with many of back-end-forth. To have real conversation with a bot, we would need to build on things that were discussed earlier to create an NLU model to extract entities. The core model will create a probability model which decides the set of actions to perform on the extracted intents.

6.3.1 Building the Core

To train the core model for our bot we need to use several files that are needed to make our model handle dialogues. These files are, 'stories.md', 'domain.yml' and 'actions.py'. Rasa Core works by learning from example of conversations, so we need to provide one, which is the 'stories.md' file. Suppose we have a simple conversation, the user says 'hello' this is considered as intent 'greet' now we want our bot to say 'hello' back which would be 'utter_greet' so our story file would look like as shown in the next image.

```
## story1
* greet
  - utter_greet
```

A story starts with '##' followed by a name (the name is optional). lines that start with '*' are messages sent by the user. Although, we don't need to write the actual message, but rather the intent (and the entities) that represent what the user means. Lines that start with '-' are actions taken by our bot. In this case all our actions are just messages sent back to the user, like 'utter_greet', but in general an action can do anything, including calling an API and interacting with the outside world. This is how one of the stories looks like created for our chatbot.

```
## story 01
*greet
  - utter_greet
*goodbye
  - utter_goodbye
*schedule_meeting
  - utter_ask_meeting_time
*search_for_meeting
  - actions.ActionShowMeeting
```

As for our 'domain.yml' file, domain file has essential role for our core model. Domain defines the world that our bot is living. Domain file connects all the intents along with the actions that our bot has to perform or call. It contains all the utterances that can be sent to a user when having a conversation, but whether which utterance or action has to be performed on the user's intent, our bot does not know, this is why we use the 'stories.md' file shown previously to train the bot on how to deal with different intents. At the moment our domain file looks like as shown in the next image.

```

3  actions:
4    - utter_ask_meeting_time
5    - utter_greet
6    - utter_ask_location
7    - action_weather
8    - actions.ActionShowMeeting
9    - utter_goodbye
10 entities:
11   - time
12   - location
13   - meeting
14 intents:
15   - greet
16   - goodbye
17   - schedule_meeting
18   - search_for_meeting
19   - inform
20 slots:
21   location:
22     type: text
23   meeting:
24     type: text
25   time:
26     type: text
27 templates:
28   utter_ask_location:
29     - text: In what location?
30   utter_ask_meeting_time:
31     - text: What time would you like me to schedule your meeting?
32   utter_goodbye:
33     - text: Talk to you later.
34     - text: Bye bye :(
35   utter_greet:
36     - text: Hello, how can I help you?
37     - text: Hi, I am here to help.

```

In the domain file we have 5 parts that are essential for our bot. First one is intents, these are the things that we expect users to say, extracted by NLU model. Actions are the example of things we would want our bot to perform. Templates are pattern default strings that our bot should dispatch regarding different intents. Entities are pieces of information that we want to extract from messages. Slots are examples of information to keep track of during the conversation. At the moment, the Slots and Entities are the same but if we cared about the user's name or age etc. We would have more example of slots.

Rasa Core's job is to choose the right action to execute at each step of the conversation. Simple actions are just sending a message to a user. These simple actions are the actions in the domain, which start with 'utter_'. They will just respond with a message based on a templates section. However, since we want our bot to be different, I have built custom actions specified in 'domain.yml' file to perform the actions we want which are 'ActionShowmeeting' and 'action_weather'. This are example of actions, that

I have made such that it connects to the online API's called 'apixu' to check the weather or the local database to retrieve meetings for a user. The code for 'actions.py' can be found at figure below.

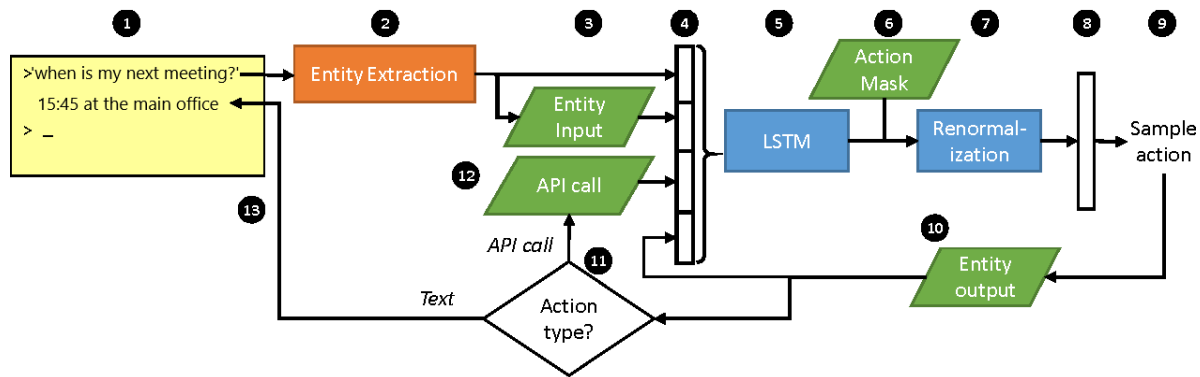
```
10 from rasa_core_sdk import Action
11 from rasa_core_sdk.events import SlotSet
12 #from rasa_core_sdk import Tracker
13 #from WeatherAPIXU.client
14
15 class ActionWeather(Action):
16     def name(self):
17         return 'action_weather'
18
19     def run(self, dispatcher, tracker, domain):
20         from apixu.client import ApixuClient
21         api_key = 'f7d8a7805fb74e22bffa182510192804' # your apixu key
22         client = ApixuClient(api_key)
23
24         loc = tracker.get_slot('location')
25         #current = client.getcurrent(q=loc)
26         current = client.current(q=loc)
27
28         country = current['location']['country']
29         city = current['location']['name']
30         condition = current['current']['condition']['text']
31         temperature_c = current['current']['temp_c']
32         humidity = current['current']['humidity']
33         wind_mph = current['current']['wind_mph']
34
35         response = "It is currently {} in {} at the moment. The temperature is {} degrees, the humidity is {}% and the wind speed is {} mph.".format(
36             condition, city, temperature_c, humidity, wind_mph)
37
38         dispatcher.utter_message(response)
39         #dispatcher.utter_message('test test')
40         return [SlotSet('location', loc)]
41         #return []
42
43 class ActionShowMeeting(Action):
44     def name(self):
45         return 'action_show_meeting'
46
47     def run(self, dispatcher, tracker, domain):
48         dispatcher.utter_message('Here is the meeting information!')
49         return []
```

In this code we can see each custom action is assigned to its separate class. This is done to avoid complexity since each action is for different intent and it would be better for those actions to not share the same exact values retrieved from the user. For example, I have imported 'rasa_core_sdk.events' that during the conversation, will save all the retrieved slots from the conversation with help of our pre-trained NLU model that later when the action called, will pass these slots to the actions. For example, when the ActionWeather is called the function 'run' creates client to connect to the online API-'apixu'(apixu is open source free weather API that provides real-time data about the weather). (line 22) and then derives all the slots (line 24), it then parses those slots e.g. location, to check the weather. All the derived information about the weather is then parsed into components (from line 28 to 33) which then are formed into a long string sentence (line 35) and dispatched to a user as a string (line 38).

Now so that we have all the files needed, we can train our dialogue model by the example of code below.

```
4  import logging
5  from rasa_core import training
6  from rasa_core.actions import Action
7  from rasa_core.agent import Agent
8  from rasa_core.domain import Domain
9  from rasa_core.policies.keras_policy import KerasPolicy
10 from rasa_core.policies.memoization import MemoizationPolicy
11 from rasa_core.featurizers import MaxHistoryTrackerFeaturizer, BinarySingleStateFeaturizer
12 from rasa_core.channels.console import ConsoleInputChannel
13 from rasa_core.interpreter import RegexInterpreter
14 from rasa_core.interpreter import RasaNLUInterpreter
15
16 # Function
17 #-----
18 def train_dialog(dialog_training_data_file, domain_file, path_to_model = 'models/dialogue'):
19     logging.basicConfig(level='INFO')
20     agent = Agent(domain_file,
21                  policies=[MemoizationPolicy(max_history=1)])
22     training_data = agent.load_data(dialog_training_data_file)
23     agent.train(
24         training_data,
25         augmentation_factor=50,
26         epochs=200,
27         batch_size=10,
28         validation_split=0.2)
29     agent.persist(path_to_model)
30
31 # Train
32 #-----
33 train_dialog('data/stories.md', 'domain.yml')
```

Here, we can change the training policy in which we can define our own LSTM or RNN (paragraph from 5.2 to 5.2.4) for dialog training (it is running on LSTM by default). One important point, 'max_history' is used to define how one action is dependent on previous questions. If 'max_history' is 1, then it just memorizes individual intent and its related actions, if we want the bot to remember and act more precisely on very large amount of intents the 'max_history' must be changed to a larger number. As a default it is set to 1 but during the trainings, I constantly keep it between 2 to 4. After the training is complete the core model will be saved in './models/dialogue' directory to be used later. As mentioned earlier, Rasa Core is a model that maps from dialog history to system actions (utterances of the bot). This decision is processed by LSTM, for example when we pass it a processed sentence(entities) 'when is my next meeting', it will make the decision to which action it is primarily relevant and perform based on its pretrained core model data that has all the example of long/short pre-trained sentences each with list of words(entities) 'when' followed by 'is' followed by 'my', 'next' etc. all under one action 'show_meeting' Check example in the following image.



6.4 Interactive Learning

The purpose of the LSTM is to take actions in the real world on behalf of the user. Although, it can be optimized using supervised learning (SL), with a domain dialogs which the LSTM should imitate or by using reinforcement learning RL, where the system improves by interacting directly with end users. The rasa core allows us to implement this kind of approach which is defined as Interactive learning. With interactive learning, our model can be constantly upgraded by changing the 'stories.md' file where every time the LSTM will re-learn to improve its performance.

In interactive learning mode, it is possible to provide step-by-step feedback on what our bot decided to do. It's similar to reinforcement learning, but with feedback on every single step (rather than just at the end of the conversation). Because we are providing *step-by-step* feedback, it is improved even more since it is trained much more directly by passing the correct answers to each step. Please check the image below to see how interactive learning is performed through cmd with implemented custom actions on NLU and core models. The output of the bot is described on the left side and user's input on the right side.

Chat History		
#	Bot	You
1	action_listen	
2		hi there intent: greet 0.83
3	utter_greet 1.00 Hi, I am here to help. action_listen 1.00	
4		bye intent: goodbye 0.79
5	utter_goodbye 1.00 Talk to you later.	
Current slots: location: None, meeting: None, time: None		

This was simple example of intents 'greet' and 'goodbye'. Now, let's try to make it work with multiple queries followed by one another. Please follow the image below.

#	Bot	You
1	action_listen	
2		hi there intent: greet 0.83
3	utter_greet 1.00 Hi, I am here to help. action_listen 1.00	
4		what is the weather like? intent: inform 0.92

Current slots:
location: None, meeting: None, time: None

The bot wants to run 'action_default_fallback', correct? (Y/n)

It has successfully classified it under intent inform with 92% accuracy. However, it has failed to understand that the slot 'location' is empty therefore it should have asked us for different action e.g. 'utter_ask_location'. It has asked my question whether the bot has to perform 'action_default_fallback' this is default action of Rasa usually run when the bot think that the conversation has ended. So, after telling it 'No' it has given me options of what action should it perform instead. Check example below.

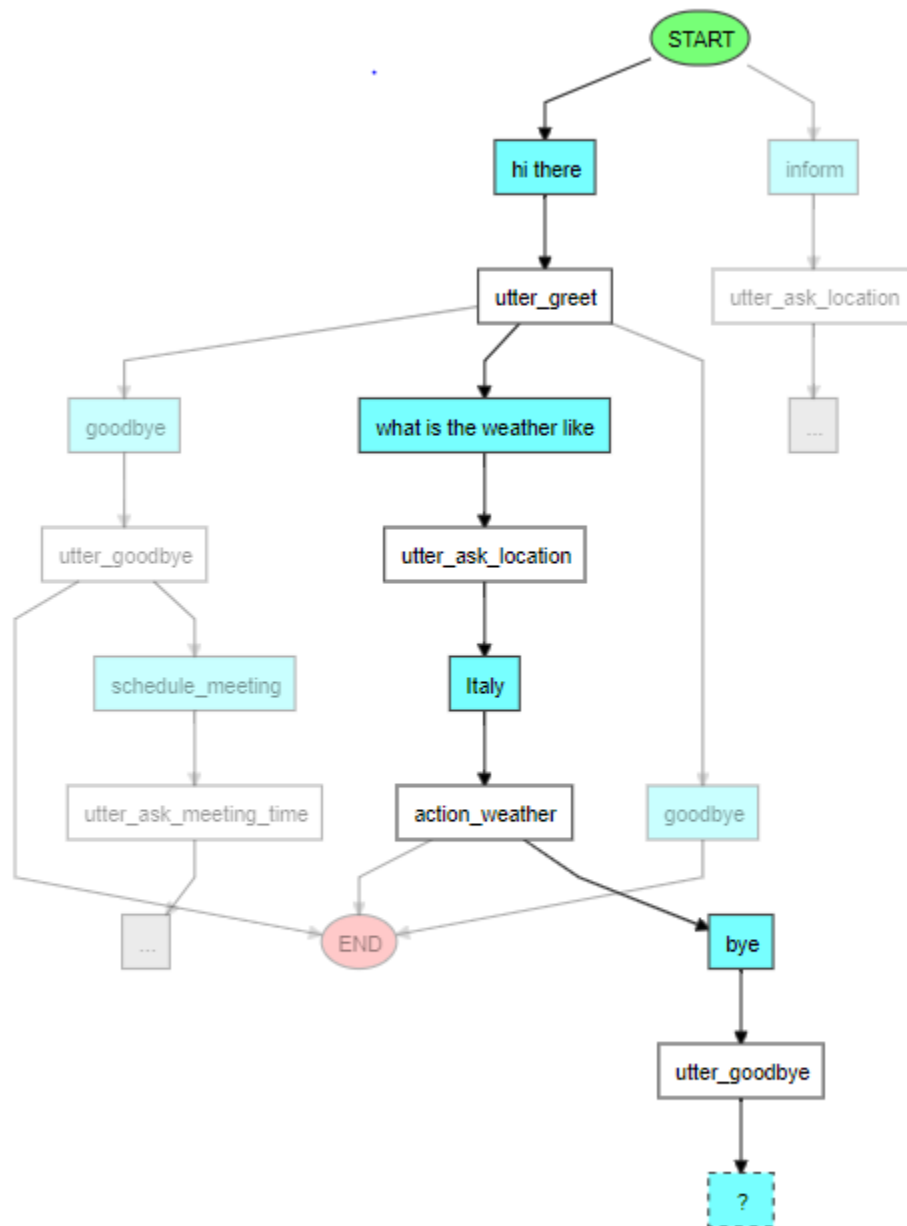
```
? What is the next action of the bot? (Use arrow keys)
<create new action>
1.00 action_default_fallback
0.00 action_deactivate_form
0.00 action_default_ask_affirmation
0.00 action_default_ask_rephrase
0.00 action_listen
0.00 action_restart
0.00 action_revert_fallback_events
0.00 action_weather
0.00 actions.ActionShowMeeting
» 0.00 utter_ask_location
0.00 utter_ask_meeting_time
0.00 utter_goodbye
0.00 utter_greet
```

After running 'utter_ask_location' I pass the location 'Italy' after which, it runs the 'action_weather'. Please refer to the next Image.

----- Chat History		
#	Bot	You
1	action_listen	
2		hi there intent: greet 0.83
3	utter_greet 1.00 Hi, I am here to help. action_listen 1.00	
4		what is the weather like? intent: inform 0.92
5	utter_ask_location In what location? action_listen 0.69	
6		[Italy](location) intent: inform 0.32
7	slot{"location": "Italy"} action_weather slot{"location": "Italy"} It is currently Clear in Rome at the moment. The temperature is 15.0 degrees, the humidity is 72% and the wind speed is 8.1 mph.	

From the image above you can see the outcome of the whole conversation. At this stage, we can either save or continue the conversation, the saved file will update the 'stories.md' with the outcome of this as a new story, which will be used within the bot to retrain in its core model. If during interactive training, we have had examples of wrong inputs e.g. typo like 'what is he weter like?' it will still be possible to run the action as soon as I direct it to. However, when saving the conversation, the NLU model will also be re-trained in order to update it with example of words that can also work even if they are typed wrong. This will increase the performance of the bot in human like conversations.

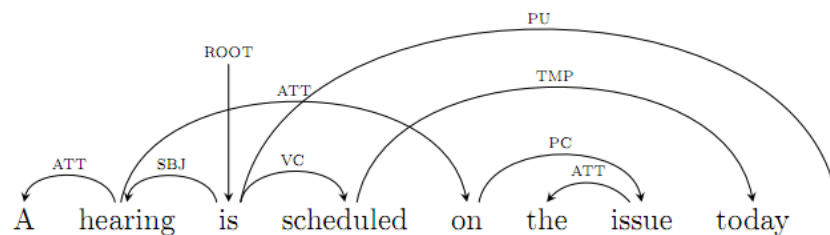
In the example below we can see the visualization created by Interactive-learning after the conversation has ended. It shows how chatbot labels all of the stories which are based on the steps it has taken to fulfill user's queries (the more we train through interactive Learning the larger it becomes).



7.0 Related Work

7.1 Dependency Parser

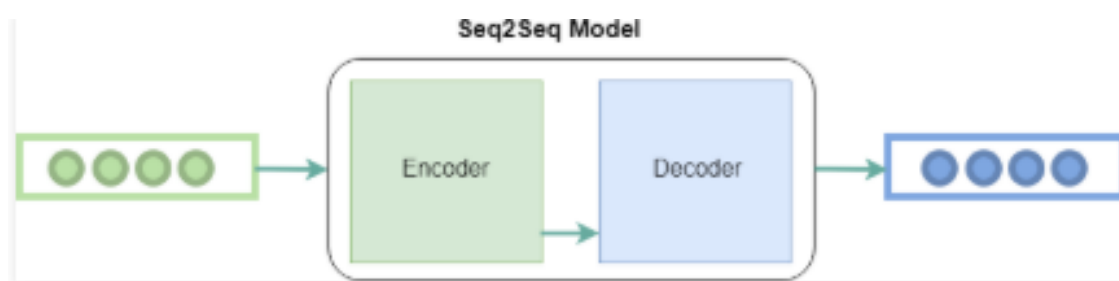
A dependency parser is another approach of parsing English sentence into structured phrase. In order to retrieve the inherent relations from corpus texts, it assigns grammatical relations between the words in a sentence - this can be represented as parsing trees or dependency graphs. For example, the sentence 'A hearing is scheduled on the issue today.' would be expressed in a dependency graph as shown in the figure below.



From the figure above, you can see how each verbal expression of a word can be dependent on the verbal expression of the rest of the words within the same sentence. When two words are connected by a dependency relationship, one is referred to as the 'head' word and the other as the 'dependent'. Dependency parsing uses a dependency grammar which assumes that syntactic structure consists of lexical elements linked by binary asymmetrical called dependencies [6]. For example, in English grammar words like 'by, came, up, down, etc.' are considered as prepositions and must always be followed by a noun or a pronoun determining the object/person in each sentence in relation to the other object/person for which the preposition is being used. For example, 'when is my next meeting at the Amex-stadium' is considered as correct sentence where the preposition 'at' is followed by the noun 'Amex-Stadium' which describes a place. However, because prepositions must be always followed by a noun, they usually should not be used at the end of the sentences. Even so, in certain circumstances it is acceptable to end the sentence with a preposition. For example, 'Amex-Stadium is where I will be at.' The preposition 'at' is used in the end of the sentence which could be understood by a Native-English speaker but would be ambiguous for the system. In order to classify and extract all the verbal meanings and their dependencies, the dependency parser analyzes the grammatical structure of a sentence. Thus, it can be used to solve different complex NLP problems such as recognizing named entities, relation extraction, or even translation between different languages.

7.2 Seq2Seq model

Seq2seq model is another machine text translation technique that was first introduced by Google [15]. General purpose of Seq2seq model is to perform machine translation over different languages, but it has been used for variety of tasks, such as, Summarization, Conversational Modeling and Image Captioning [15]. Seq2seq, takes input as sequence and returns output sequence of words. It does so by using LSTM. The seq2seq model is based on two procedures. The first procedure is Encoding, meaning it uses deep neural network layers and the input words to the corresponding hidden vectors. Such that, each vector represents the context of the word. The second procedure is Decoding which is similar to Encoding except, it takes hidden vector produced by the Encoder along with the current input to finally predict the next word. Please check example in the image below.



In the article about 'An empirical study of building chatbot using seq2seq model' [25]. It is explained how it is possible to create a chatbot based on many to many approach (LSTM/RNN) described earlier, to simply swap the input and output with the help of encoder and decoder. So, it can use neural machine translation (seq2seq mode) directly to build chatbot. For example, instead of using Mandarin characters as input and decoding output as English characters, it will decode the encoded English input-text with embedded output e.g. with sentence 'How is the weather' the output can be 'I don't know'.

Type	Train input	Train output
translation	你好	Hello
single turn QA	How's the weather	I don't know

8.0 Design

I have chosen to integrate conversational Interface that can be used to interconnect all the module files to form a bot, which will create an impression of a real conversation but with a chatbot. There are many conversational applications that allow to integrate your own chatbot through input channels, such as, Facebook-messenger, Twilio, Telegram, Webxteams and Slack. Rasa allows connection with external servers through public channels. These channels are formed between local computer and online server of the conversational application. For this Project I have chosen to implement my bot in Slack web service.

8.1 Slack API

To connect the bot to the Slack API the channel token and channel-name must be passed by 'run.py' python of the bot. This file combines both the NLU and core models to create input channel.

```
1 from rasa_core.channels.slack import SlackInput
2 from rasa_core.agent import Agent
3 from rasa_core.interpreter import RasaNLUInterpreter
4 import yaml
5 from rasa_core.utils import EndpointConfig
6
7
8 nlu_interpreter = RasaNLUInterpreter('./models/nlu/default/MyNLU')
9 action_endpoint = EndpointConfig(url="http://localhost:5055/webhooks/slack/webhook")
10 agent = Agent.load('./models/dialogue', interpreter = nlu_interpreter, action_endpoint = action_endpoint)
11
12 input_channel = SlackInput( slack_token='xoxb-607403756646-603559674128-oIUzV7UMm3g8lNl3HzQF5b')
13
14 agent.handle_channels([input_channel], 5004, serve_forever=True)
```

when connecting to the slack, it will pass this 'channel-name' with its 'token' and gain permission (line 12). After what the agent will create channel for the public internet with the port of 5004(line 14).

The endpoint of receiving messages from the slack to the bot must be defined as well. Check below.

```
http://localhost:5005/webhooks/slack/webhook
```

This will allow the bot to analyze the text input retrieved from the slack and dispatch the utterances along with the actions.

After all the configurations are complete, we can then follow to slack-URI to chat with our chatbot online. Check the next image.

personal ▾
Hovik Hakobyan

All Threads

Channels
chatbot
general
random
+ Add a channel

Direct Messages
Slackbot
Hovik Hakobyan (you)
+ Invite people

Apps
● Rayan
+ Install Google Drive

☆ Rayan
Messages About

Hovik Hakobyan

4:54 AM

Hey Rayan

Rayan

APP 4:54 AM

Hi, I am here to help.

Hovik Hakobyan

4:54 AM

what is the weather like?

Rayan

APP 4:54 AM

In what location?

+

Message Rayan

9.0 Testing and evaluation

9.1 Testing

While this chatbot is intended to serve as a personal assistant that demonstrates the capabilities of the NLP and ML, it is not without its flaws, limitations and drawbacks.

One such drawback is cost of computation, this project is interconnected to a relatively big number of libraries and each of these libraries has to be of a specific version in order for them to connect with Rasa in a single API. One other limitation of this project is that this chatbot can only understand a single intent at the time, from every passed input. For example, with the example of sentence 'Hey Rayan, when is my meeting' it will categorize it as intent 'ActionMeeting' and return the meeting time, but it would not be able to categorize it under two intents which in this case would also be 'greeting'. So, the output would look like '21:00 at the Chinese restaurant.' Where the correct reply should look like 'Hi there, your meeting is at 21:00 at the Chinese restaurant'. This bot is created based on a self-learning approach however it is limited when it comes to dealing with multiple queries at the time.

One another flaw of this project is that the way all the files (NLU and core models, along with the stories.md and 'config.js') are interconnected to establish a conversation. This Project can not be installed and used as an application of a chatbot on an operating system; it has to be run as a local server through 'cmd' or through an external server for conversational UI, such as Slack, so that the Interactive training can perform its classification while keeping track of the conversation online locally.

Throughout the backend development process, some unit tests were considered to ensure that each stage of improvement/change, the functionality worked as intended and the new changes did not break the existing code. This project is configured under many files that are linked with each other, which made the testing harder since it was not clear whether the issues/errors were external or internal (because of the libraries used, or structure/semantics issues in the training files). For example, to implement custom actions for the bot, I was required to create a separate rasa server that integrates my custom actions along with rasa's default actions into the core model, such that it would be possible to test whether my actions were recognized during the conversation or not. During the testing, the bot was running through a local server, 'cmd' was used as the testing environment. Once the backend was fully developed (local servers and rasa integration on python) I began to establish a connection between a front end and back end, after which the further tests were followed on the front end. The unit tests were example of custom 'test.py' files that would run the main code partially to detect internal bugs.

Had unit testing not have been done the front end would not be fully developed to test the code and this would have not only taken more time but would have also introduced the problem that if something was not working it would not be clear if the problem was from the front end or back end and debugging would have been more difficult.

9.2 Evaluation

As discussed earlier in this report, this chatbot is formed from the example of components that make it self-learning bot. Another approach to create a chatbot is known as 'rule-based'. This type of AI, simply takes an input from the user, checks the rules that it has been defined, and gives the pre- encoded response. The example of rule based chatbot is discussed in the paragraph 7.2, where the Seq2seq model is applied to encode inputs with decoded outputs. Just like it is used for machine translation where each word has its translated decoding indifferent language, Seq2se is used to apply the same approach to expected inputs from as user and with the utterances for them.

For the rule-based approaches the data sets used to train the bot are not biased – which they will be in the absence of a large number of real-world examples – they are unlikely to be exhaustive enough to cover every possible scenario the chatbot might encounter, while this self-learning chatbot will get better and better in its deliverance because it is re learning to improve while a rule based bot only understands a pre-defined set of options.

Throughout the development of this bot, there were number of tests performed to ensure the usability and the efficiency of this chatbot. As discussed earlier, different examples of training-data were used to train the bot to deal with ambiguous texts or synonyms that still can be used to derive the intents from the input text. For example, sentences with typing errors 'wht is the wether like today?' if not recognized, it would then be passed to Interactive training to resolve this issue when the next time similar or the same example of queries would be typed by a user, they then would be assigned to expected intents.

The possibilities of this chatbot could be expanded in dealing with more example of intents. However, because of the time constraints, it was decided that that making it work with more intents and proceed with different actions was out of scope for this project, but feasible if given more time.

10.0 Conclusion

Overall this bot has accomplished the objectives that were set for it. The bot is functional and serves as a proof of concept that demonstrates possibilities of chatbot.

Further improvements could be done to increase the capacity of the intents handled by this chatbot. Given more time, it would also be possible to make it work with multiple intents at the time by creating another core model, which would correlate both intents and dispatch different utterances that fits the most e.g. 'greeting' 'action-request' together.

Due to its complexity of running multiple servers and upgrading the pre-trained files on run time, it would be major improvement to host this project on Amazon Web Services (AWS) such that, all the servers would not need to run locally anymore and would be easier to contribute for any major changes in future.

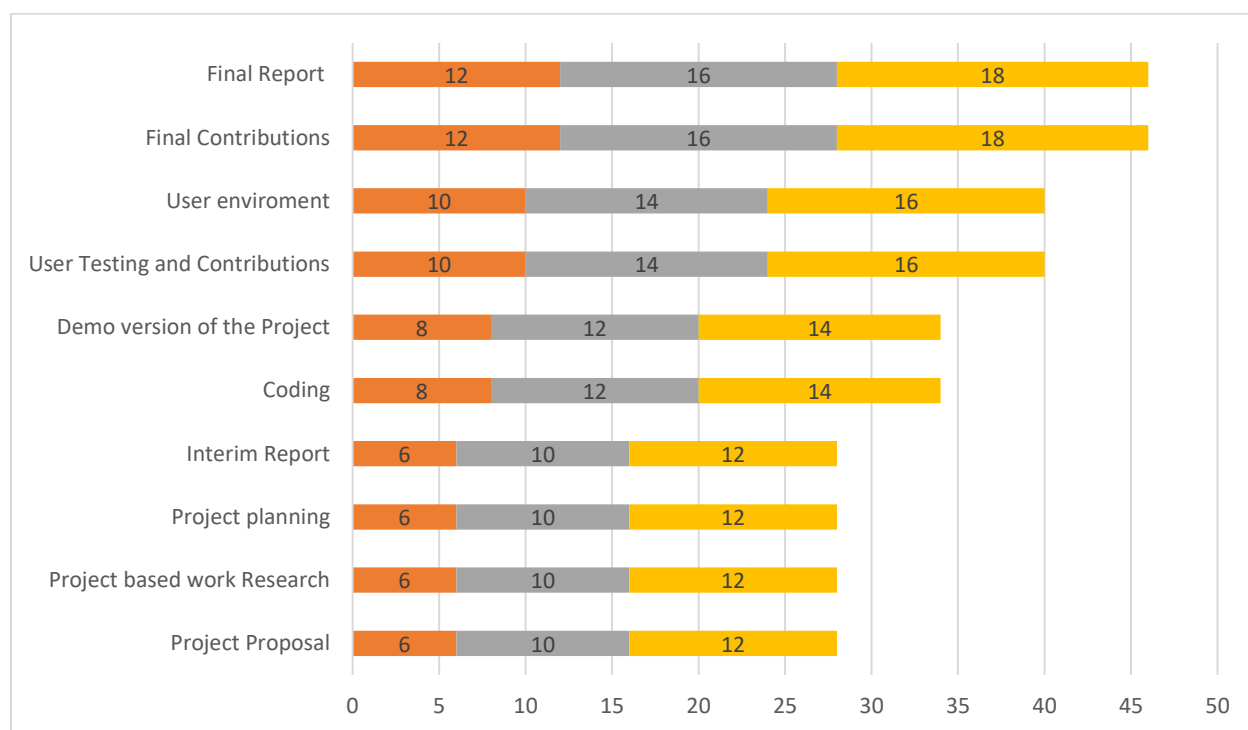
To summarize, this project has been a success as a fully functioning chatbot system along with implemented Interactive Learning, this chatbot can be classed as at is a usable system and can assist users. With regards to NLP and ML components, I have gained enough professional knowledge and skills in area of Natural Language Engineering to maintain awareness and technological developments, procedures, and standards in this field to create a chatbot.

11.0 Appendices

11.1 Project Plan

Attached below is a Gantt chart outlining the tasks that are already completed, as well as the time expected to spend on each task which is represented as 'hours per week'.

	Task Name	Start Date	Duration in hours per week			Expected End Date	days
			Least	Expected	Goal		
1	Project Proposal	10/1/2018	6	10	12	10/15/2018	7
2	Project based work R	10/15/2018	6	10	12	10/22/2018	7
3	Project planning	11/22/2018	6	10	12	11/29/2018	14
4	Interim Report	11/29/2018	6	10	12	11/15/2018	14
5	Coding	12/13/2018	8	12	14	12/20/2018	24
6	Demo version of the	12/20/2018	8	12	14	12/27/2018	7
7	User Testing and Cor	1/24/2019	10	14	16	1/31/2019	30
8	User enviroment	2/7/2019	10	14	16	2/14/2019	14
9	Final Contributions	2/14/2019	12	16	18	2/21/2019	14
10	Final Report	2/21/2019	12	16	18	3/21/2019	31



There were 3 main stages to this project: background reading, implementing and conducting different approaches for named entities, and coding and integrating online database for checking the weather.

The background reading involved me searching more examples of chatbots generated by different approaches to gain the necessary knowledge to understand which of those techniques would help me more to achieve the main objectives for this project. It has also helped me to gain knowledge for discussing further for this report, specifying how my approach would be different and why would it be more useful for the intentions of this project.

When implementing different approaches such as the ones explained in this project, I was required to use external resources in order to download and use labelled data. However, most of the data was generated by me to create sufficient intents for this chatbot.

11.2 Interim log

3/10/2018: Introductory meeting with supervisor in groups, where everyone got a chance to present their ideas of how they think they are going to structure their Final year projects while others could ask questions and discuss it among us by guidance of the supervisor.

6/11/2018: Discussed the project proposal that I have submitted. Discussed the main objective of this project and the difference between functional non-Functional requirements.

12/11/2018: reviewed the draft of interim report with my supervisor, discussed main bullet points that have to be represented in each paragraph.

06/02/2019: reviewed Discussed different implantations that I have researched which can be used to create this chatbot on python.

04/03/2019: reviewed Have implemented example of demo, discussed how to improve them and explored how to implement User Interface

03/04/2019: reviewed Catch up meeting with regards to implementation. Final discussion about code and any improvements. Also discussed about when to submit first draft of report.

25/04/2019: reviewed Feedback on first draft of report.

Attached below is the copy of my proposal that I have submitted for the final year project and have discussed with my supervisor.

Student number: 164848

Supervisor: Julie Weeds

Personal Assistant based on Human-Computer Interaction

Purpose

Personal assistant to help and guide employees/employers on their daily basis, based on different queries from a user, the Personal assistant will generate responses and/or Notify them about their scheduled meetings or just generate general responses if required. The responses however will be based on the Natural Language Processing approaches such as, Method 52 and Word to Vector embeddings that will be trained and tested upon many different corpuses.

Process

The process will involve all the steps to perform Natural Language Processing such as Tokenization, Lexical and semantic Analyses, after what the resulted outcome will be processed through different methods of classifications such as Word2vec embeddings where the words or phrases are mapped to vectors of real numbers, This can help to understand the word similarity in phrases or synonyms to prevent the ambiguity in human expressions.

Objectives

Project will be written in Python that will perform question answering based on its uniquely modified corpus. However, this Project will involve extensive use of different libraries(corpuses) to extract dates and time to be able to not only answer questions but also to assist. Libraries like Method 52 will be used to classify and test reliability of this Project in order to deliver certainty in its answers.

Primary Objectives:

- Working Project written in python run on windows
- Local MySQL DB of extracted Phases and sentences with their Entity Relationships
- Classified list of words (corpus) for it's primarily uses
- Final Report
- Citation of the libraries and other sources used

Expected difficulties:

- Part of Speech tagging issues due to amount of the data-sets used
- OS related problems based on different libraries used for Python
- Lack of understanding human expressions

Expected Optional adjustments:

- Texting environment
- Many more libraries used like word2vec
- Voice responses

11.3 How to run

To run this bot you have to follow 4 steps.

1) Rasa installation

You must install the latest version of Rasa Core. It is highly recommended that you create a virtual environment and then proceed with the installations

Creating virtual enviroment

```
conda create -Rayan python=3.6
```

Activating the new environment to use it

WINDOWS: activate bot

LINUX, macOS: source activate bot

Installing the latest Rasa stack and NLU

```
python -m pip install rasa_nlu[spacy]  
(https://rasa.com/docs/nlu/installation/)
```

Rasa Core

```
python -m pip install -U rasa_core  
(https://rasa.com/docs/core/installation/)
```

Language model

```
python -m spacy download en_core_web_md  
python -m spacy link en_core_web_md en --force;
```

Note: The Project file contains all the files needed and requirement.text with all the dependencies listed in them.

2) Setting Slack application

Let's us create a slack integration by creating a slack app. Let's go through the process of creating a Slack app.

- Create a Slack account and go to <https://api.slack.com/>. Now choose either an existing development Slack workplace(in case you have one) or create a new one. Name it DemoWorkplace or anything you like.
- Now create a Slack app in DemoWorkplace and give it a name 'Rayan'.
- Start adding features and functionalities to the app. We'll first create a 'bot user' under the 'Bots' Tab and integrate it into the app. This will make the app conversational. Save all the changes made.

- Start adding features and functionalities to the app. We'll first create a botuser under the Bots Tab and integrate it into the app. This will make the app conversational. Since this bot will be created as a user, we can choose to keep it constantly online. Save all the changes made.
- Lastly, we need to install this app into our workplace which we defined earlier. Authorize it and we have our app ready and integrated into the workplace. please follow the process in the example here: https://cdn-images-1.medium.com/max/600/1*nNCR8wmXVkJQAZJ7BTespq.gif

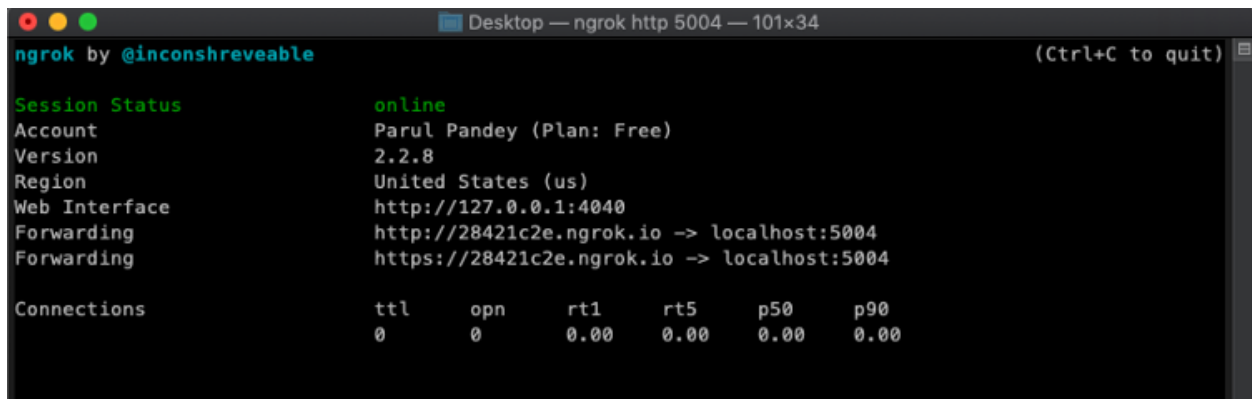
3) Setting ngrok

Ngrok is a multi-platform tunnelling, reverse proxy software that establishes secure tunnels from a public endpoint such as the internet to a locally running network service.

- 1- Download ngrok from here <https://dashboard.ngrok.com/user/login>
- 2- Unzip to install.
- 3- Run ngrok from unzipped file

Once run pass it `./ngrok http 5004`

If everything works you should see example of window below.



```

ngrok by @inconshreveable (Ctrl+C to quit)

Session Status      online
Account             Parul Pandey (Plan: Free)
Version             2.2.8
Region              United States (us)
Web Interface        http://127.0.0.1:4040
Forwarding            http://28421c2e.ngrok.io -> localhost:5004
Forwarding            https://28421c2e.ngrok.io -> localhost:5004

Connections
  ttl    opn    rt1    rt5    p50    p90
    0     0     0.00   0.00   0.00   0.00

```

You will later need to copy the generated ngrok URL to the slack in the following steps

4) Deploying the bot on Slack

In the project file you will find `run_app.py` which will integrate the chatbot with the slack app

But first you need to run the custom actions created for this bot, it can be done through running

```
python -m rasa_core_sdk.endpoint --actions actions
```

Now you need to run the `run_app.py` but first you Make sure to provide the slack token in the `run_app.py` it can be found under variable name Input channel.

```

from rasa_core.channels.slack import SlackInput
from rasa_core.agent import Agent
from rasa_core.interpreter import RasaNLUIInterpreter
import yaml
from rasa_core.utils import EndpointConfig

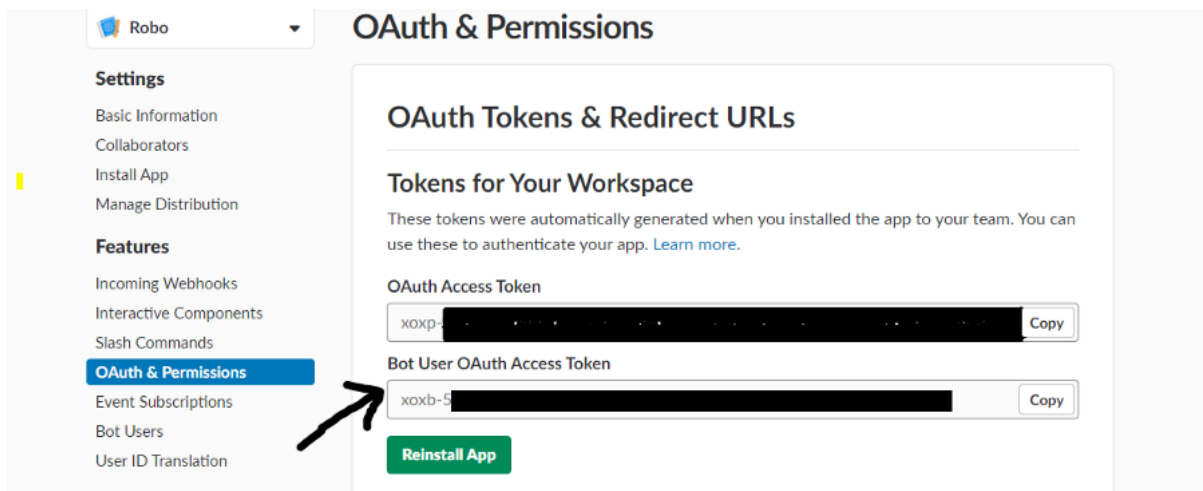
nlu_interpreter = RasaNLUIInterpreter('./models/nlu/default/MyNLU')
action_endpoint = EndpointConfig(url="http://localhost:5055/webhooks/slack/webhook")
agent = Agent.load('./models/dialogue', interpreter = nlu_interpreter, action_endpoint = action_endpoint)

input_channel = SlackInput( slack_token='xoxb-607403756646-603559674128-oIUzV7UMm3g81N13HrHzQF5b')

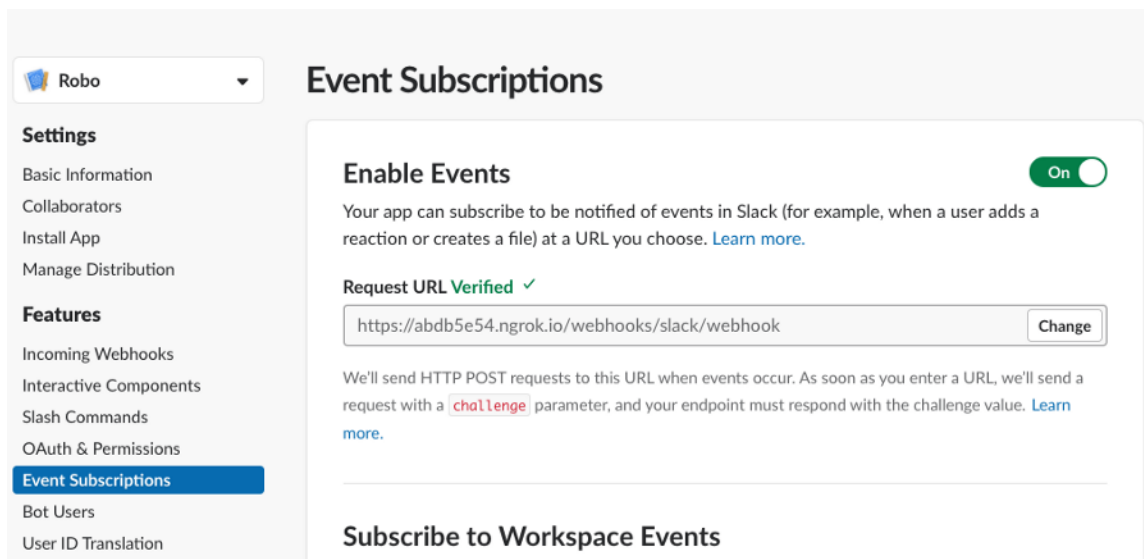
agent.handle_channels([input_channel], 5004, serve_forever=True)

```

The slack token can be obtained as followed.





- Now go to the event subscriptions page on Slack



- Provide the url gained from the ngrok and paste at Request URL to verify.
- Subscribe to workplace events by adding bot user event. Save all changes

Subscribe to Bot Events

Bot users can subscribe to events related to the channels and conversations they're part of.

Event Name	Description	
app_mention	Subscribe to only the message events that mention your app or bot	
message.im	A message was posted in a direct message channel	

Add Bot User Event

Ensure!

- That the custom actions server is running
- Ngrok is running on port 5004
- Run.py is running

Now navigate to the slack interface and chat with the bot 'Rayan'

12.0 References

- [1] The Chartered Institute for IT, "BCS Code of Conduct," [Online]. Available: <http://www.bcs.org/category/6030>
- [2] "Ethical Compliance Form for UG and PGT Projects", [Online]. Available: <https://www.sussex.ac.uk/webteam/gateway/file.php?name=ethical-review-guidance-for-ug-and-pgt-projects.pdf&site=356>.
- [3] Department of Electronic Engineering, "NLP-based Artificially Intelligent Chat-bot", 2007. [Online]. Available: <http://lbms03.cityu.edu.hk/studproj/ee/2007eepsb281.pdf>
<https://pdfs.semanticscholar.org/4119/324f5bdbbf6b620e90ea26f5e4d27e6b8de0.pdf>
- [4] Stanford University, Berant and Liang, "Semantic Parsing via Paraphrasing", 2009. [Online]. Available: <http://www.aclweb.org/anthology/P14-1133>
- [5] University of Nevada Las Vegas, Savitha, "Study of stemming algorithms", 2010. [Online]. Available: <https://digitalscholarship.unlv.edu/cgi/viewcontent.cgi?article=1755&context=thesesdissertations>
- [6] University of East Anglia, Marie-Claire, 2011. [Online]. Available: <https://ueaeprints.uea.ac.uk/47388/>
- [7] IJIRSET, A.alini and U.Jeyapriya, 2017. [Online]. Available: https://www.ijirset.com/upload/2017/icmtest/72_Named_Entity_501.pdf
-] "<https://towardsdatascience.com>," 26 Feb 2018. [Online]. Available: <https://towardsdatascience.com/recurrent-neural-networks-and-lstm-4b601dd822a5>
- [9] R. e. al., "Sequence Modeling: Recurrent and Recursive Nets," Sequence Modeling, 2009. Available: <http://www.deeplearningbook.org/contents/rnn.html>
- [10] N. Donges, "<https://towardsdatascience.com>," March 2018. [Online]. Available: <https://towardsdatascience.com/recurrent-neural-networks-and-lstm-4b601dd822a5>
- [11] J. Schmidhuber, "Deep learning in neural networks: An overview," August 2016. Available: <https://www.sciencedirect.com/science/article/pii/S0893608014002135>
- [12] J. Brownlee, "A Gentle Introduction to RNN Unrolling," September 2017. [Online]. Available: <https://machinelearningmastery.com/rnn-unrolling/>
- [13] J. Brownlee, "<https://machinelearningmastery.com>," December 2017. [Online]. Available: <https://machinelearningmastery.com/exploding-gradients-in-neural-networks/>
- [14] F. Gers, "Understanding LSTM Networks," August 2015.
- [15] Google, "seq2seq - Introduction," [Online]. Available: <https://google.github.io/seq2seq/>.

- [16] R. K. John Hewitt, "Sequence-to-sequence," 2018.
- [17] M. B. Nicole Radziwill, "Evaluating Quality of Chatbots".
- [18] C. J. A. M. Anik Raj C, "A SURVEY ON WEB BASED CONVERSATIONAL BOT DESIGN," October 2016. [Online]. Available: <https://pdfs.semanticscholar.org/a4e8/534437b8576705031522e8e6b22ab6f40439.pdf>. [Accessed May 2019].
- [19] spaCy, "Architecture," [Online]. Available: https://spacy.io/api#_title. [Accessed May 2019].
- [20] NLTK, "NLTK 3.4.1 documentation," [Online]. Available: <https://www.nltk.org/api/nltk.html>. [Accessed May 2019].
- [21] Rasa comuinty, "Build contextual chatbots and AI assistants with our open source conversational AI framework," Decmber 2018. [Online]. Available: <https://rasa.com/docs/>
- [22] Rasa community, "rasa core," October 2018. [Online]. Available: <https://rasa.com/products/rasa-core>
- [23] R. S. C. D. M. Jeffrey Pennington, "GloVe: Global Vectors for Word Representation," [Online]. Available: <https://nlp.stanford.edu/projects/glove/>. [Accessed May 2019].
- [24] R. S. Jeffrey Pennington, "Global Vectors for Word Representation," vol. 2014. <https://www.aclweb.org/anthology/D14-1162>
- [25] P. Guo, Y. Xiang, Y. Zhang and W. Zhan, "Snowbot: An empirical study of building chatbot using seq2seq model with different machine learning framework," [Online]. Available: <https://pdfs.semanticscholar.org/7299/e85664420faacf1911c78c4e9ea3674909b9.pdf>. [Accessed May 2019].