

Intelligent Social Chatbot – Classifying Intents and Identifying Entities for Generating Response

Thesis Submitted in partial fulfillment of the requirements for the degree of

**Master of Engineering
In
Computer Science & Engineering**

Submitted By

Aritra Nayak

Examination Roll number:	M4CSE19012
Class Roll No.:	001710502004
Registration number:	140744 of 2017-2018
Session:	2017 - 2019

Under the esteemed guidance of

Dr. Dipankar Das

Dept. of Computer Science & Engineering

Jadavpur University, Kolkata-700 032

Faculty Council of Engineering and Technology
JADAVPUR UNIVERSITY, KOLKATA – 700032

Certificate of Recommendation

This is to certify that Aritra Nayak (University Registration No.: 140744 of 2017-2018, Examination Roll No.: M4CSE19012) has completed his master's thesis entitled **“Intelligent Social Chatbot – Classifying Intents and Identifying Entities for Generating Response”**, under the supervision and guidance of Asst. Professor Dr. Dipankar Das, Jadavpur University, Kolkata. We are satisfied with his work, which is being presented for the partial fulfillment of the degree of Master of Engineering in Computer Science & Engineering, Jadavpur University, Kolkata - 700032.

Prof. (Dr.) Dipankar Das

Faculty in Charge of Thesis

Prof. (Dr.) Mahantapas Kundu

*HOD, Dept. of Computer Science &
Engineering, Jadavpur University,*

Kolkata – 700 032

Prof. (Dr.) Chiranjib Bhattacharjee

*Dean, Faculty Council of Engineering and
Technology, Jadavpur University,*

Kolkata – 700 032

Faculty Council of Engineering and Technology
JADAVPUR UNIVERSITY, KOLKATA – 700032

*Certificate of Approval**

This is to certify that the thesis entitled “**Intelligent Social Chatbot – Classifying Intents and Identifying Entities for Generating Response**” is a bona fide record of work carried out by Aritra Nayak (University Registration No.: 140744 of 2017-2018, Examination Roll No.: M4CSE19012) in partial fulfillment of the requirements for the award of the degree of Master Of Engineering in Computer Science and Engineering in the Department of Computer Science and Engineering, Jadavpur University and presented in a manner satisfactory to warrant its acceptance as a prerequisite to the degree for which it has been submitted. It is understood that by this approval the undersigned do not necessarily endorse or approve any statement made, the opinion expressed or conclusion drawn therein but approve the thesis only for the purpose for which it has been submitted.

Final Examination for Evaluation of the Thesis

Signature of Examiners

* *Only in case the thesis is approved.*

Declaration of Originality and Compliance of Academic Ethics

I hereby declare that this thesis contains literature survey and original research work by the undersigned candidate, as part of her Master of Engineering in Computer Science & Engineering.

All information in this document has been obtained and presented in accordance with academic rules and ethical conduct.

I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name (Block Letters): **ARITRA NAYAK**

Exam Roll Number: **M4CSE19012**

Thesis Title: **Intelligent Social Chatbot – Classifying
Intents and Identifying Entities for Generating
Response**

Signature with date:

Acknowledgement

I would like to express my deepest and sincerest gratitude to my thesis guide cum advisor, (Prof.) Dr. Dipankar Das, for his solicitous guidance towards my research study. His knowledge and commitment to the project have been an endless source of inspiration to me. Also I would like to thank several Open Source Courses which helped me a lot in gaining knowledge on various related topics which played an important part throughout my thesis work.

Finally, I would like to thank my family members, friends for their unconditional support.

Abstract

The focus of the present thesis is to classify user input based on their intents for the development of an intelligent chatbot. In addition, it also aims to understand the entities of such intents so that they can be further employed for generating response to the user. In order to develop a chatbot especially for railway domain, a proper labeled dataset (user query versus intent class) is needed. Thus, a chatbot framework is prepared using one of the most widely used global chat systems (like Facebook messenger, Google assistant, telegram etc. along with a global server like heroku, dialogflow system of Google and a database management system (e.g., MySQL Workbench). The dataset is developed to train the intent classification model and can be used later developing dialogue management tracker system. Two frameworks have been used to develop the intent classification model, Convolution Neural Network (CNN) and Long and Short Term Memory (LSTM). A comparative study of both is discussed here. After intent classification model, an Entity Recognition model has been proposed with the help of spaCy libraries to detect all the entities from user inputs. Later, those intent values have been used to provide a response to the user by utilizing a railway based API.

Contents

Chapters			Page No.
Chapter 1: Introduction			2-11
1.1	NLP – Dialogue and Discourse Processing		2-3
1.2	Motivation		3
1.3	Chatbot		4
	1.3.1	What is a chatbot ?	4
	1.3.2	Types of a chatbot	4
	1.3.3	How NLP helps to build a chatbot	5
	1.3.4	Applications of chatbot	6
1.4	Challenges in chatbot		6-8
	1.4.1	Context in chatbots	6-7
	1.4.2	User’s limited attention	7
	1.4.3	Chatbot Testing	7-8
	1.4.4	Viability of Data	8
1.5	Objectives		8
1.6	Contributions		9
1.7	Thesis Overview		9-11
	1.7.1	Overall Architecture	9-11
	1.7.2	Dataset Collection	11
	1.7.3	Intent Classification	11
	1.7.4	Entity Recognition and Response Generation	11
Chapter 2: Related Works			12-16
2.1	Previous Contributions		12
	2.1.1	What is Turing Test?	13
	2.1.2	What Is Loebner Prize ?	13

Chapters			Page No.
	2.1.3	Official List of Winners of Loebner Prize	14-16
	2.2	Winning chatbots and Comparative Analysis	16
Chapter 3: Dataset Collection			17-35
	3.1	Existing Datasets	17-19
	3.2	Taxonomy of Intents and Entities	20
	3.2.1	Retrieval-based models	20
	3.2.2	Generative models	20
	3.3	Closed Domain vs. Open Domain	21
	3.4	Restrictions of Previous Datasets	21
	3.5	Data Collection Framework	22-35
	3.4	Corpus Statistics	35
Chapter 4: Intent Classification			36-60
	4.1	Pre-Requisite	36-37
	4.1.1	Data Preparation	36-37
	4.1.2	Defining Model	37
	4.1.3	Making Predictions	37
	4.2	CNN based Intent Classification	38-49
	4.2.1	Experiments & Results(CNN)	46-49
	4.3	LSTM based Intent Classification	49-59
	4.3.1	Experiments & Results(LSTM)	57-59
	4.4	Comparative Analysis	59
Chapter 5: Entity Recognition and Response Generation			60-69
	5.1	Entity Recognition	60-61
	5.2	spaCy framework and its application	62-65
	5.3	Experiments and Results on Entity Recognition on Test Data	65-66

Chapters			Page No.
	5.4	Response Generation	67-68
Chapter 6: Conclusion and Future Work			69-73
	6.1	Conclusion	69
	6.2	Future Work	70
	6.3	References	71-73

Chapter 1:

Introduction

Instant messaging applications, which were started in early 90's, became so popular as they great ways to communicate one to one basis with few taps of our fingers. Soon this informal way of communication was known as chats. From ICQ to AOL, Yahoo Messenger to GTalk and WeChat to WhatsApp, Facebook messenger to Telegram, Hike to Viber – instant messaging has toured a great journey on its own. Since last 25 years, chat services have become better and efficient with allowing us to send not just text messages but sharing images, videos, locations, doc files, etc. Chatbot Development is definitely going to help businesses to prosper as the potential customers are going to get much relevant information on their hands in no time. Repetitive questions, important information about any organization, business can be communicated to end users very easily with the help of chatbot development. Using chatbot, we can also reduce the manual intervention thus easing any business process in an organization.

1.1 NLP – Dialogue and Discourse Processing

Every time user types something to a chatbot, the chatbot needs to get the intent of the input strings, then after the identification of the intent, chatbot needs to respond back. Before responding back there are several steps that occur in the backend. For example, Natural Language Processing is what allows chatbots to understand your messages and respond appropriately. When you send a message with “Hello”, it is the NLP that lets the chatbot know that you’ve posted a standard greeting, which in turn allows the chatbot to leverage its AI capabilities to come up with a fitting response. In this case, the chatbot will likely respond with a return greeting. Without Natural Language Processing, a chatbot can’t meaningfully differentiate between the responses “Hello” and “Goodbye”. To a chatbot without NLP, “Hello” and “Goodbye” will both be nothing more than text-based user inputs. Natural Language Processing (NLP) helps provide context and meaning to text-based user inputs so that AI can come up

with the best response. During the conversation, sometimes it happens so, that chatbot needs to refer to a previous input of the user to provide a correct response to the current input of the user. Dialogue processing comes into play in this regards. If we can assign a specific session to each intent, and if we can remember the parameters or entities associated with that intent, we can use it later for dialogue tracker system development.

1.2 Motivation

Chatbots are the future of our today's messaging apps. Now-a-days, a robotic conversational agent is able to communicate with people helping them in a number of ways easing their lifestyle, is what chatbot development was started for. With the help of AI, human computer interaction is now in a different level and chatbots are leading its way to near future where we will be talking to bots, which will solve our day to day issues with few taps of our fingers. In our modern days. Users exhibit different ranges of motivations and patterns of use. An enormous variation in chatbot alternatives can also be seen by exploring Facebook Messenger, which now has over 200,000 chatbots. One type of chatbots help user accomplishing quick and specific tasks like checking weather outside, organizing scheduled meetings, ordering food, flight booking. Another stream of chatbots help organizing web contents and then providing user those contents in a group wise manner according to user specific need.

For example, Microsoft launched Heston Bot[36], which focuses on food and cooking opportunities as well as fashion. The global clothing company H&M launched a chatbot to provide shopping suggestions based on photos from users' personal wardrobes. Another stream of chatbots supports more long-term relationships and activities such as charitable giving, civic engagement, work, fitness, and personal health. So, in these circumstances, we can expect that the ways in which people will interact with conversational user interfaces in the future will change, resulting in new user behaviors as well as new social norms and user expectations. Here, as the work done, is a part of building a bot which will be able to provide information on trains, it will be very helpful in current condition of our country where many people carry smartphones in their pocket, but lack proper knowledge on trains. The main important lesson learned up to current scenario is that chatbots are not a one-solution-fits-all technology. People have multiple motivations, and the purposes for using a chatbot can vary enormously. As such, there is a need for an appropriate range of use cases in the chatbot context.

1.3 Chat-bot

A computer program designed to simulate interactive human-like conversation with human users, especially over the Internet.

1.3.1 What is a Chat-bot?

A chatbot is a conversational agent that exploits NLP/ML technologies to engage users in a natural language conversation to:

- **Accomplish a task:** Like information seeking (when is the next train to Vizag?) or servicing a request (book me a ticket to a movie).
- **Engage and entertain the user in some creative ways:** Like “tell me a joke” or “say something interesting about an actor.”

1.3.2 Types of Chat-bot (based on their goal)

- **Normal chitchat:** Social “hi hello” conversation. Ex: Natasha of hike, though Natasha provides many relevant day to day information regarding sports, politics, tech news etc.
- **Task Completion:** Ex – Play a song, adding some items to cart, paying monthly bills etc.
- **Question-Answering:** Like IBM Watson.
- **Recommendation system:** Ex – Any bot working in a food joints, any shopping malls etc.

No one had ever thought to use chat for the business purpose preciously, just because they were supposed to be informal communications. But in developing countries like India, people started using WhatsApp chats to order products and for getting services locally. This inspired Facebook to introduce chatbots in Messenger, which meant that to talk to the customer representative of the company you need not hold the line for minutes. This is the most favorable advantage of the chatbot Development.

The chatbot is an AI based chat option, by which businesses can now make their own chat representative for the Messenger to tackle the most reasonable queries by the customers. Chatbot development creates chatbots which is nothing but an austere software that interprets anything you type or say and accordingly respond by answering or executing the command. Most popular example of a bot in chatbot

development currently is Apple's Siri. But Facebook has taken a leap from these personal bots by amalgamating two most popular technologies – instant messaging and artificial-intelligence.

1.3.3 How NLP helps to build a chatbot

Natural Language Processing (NLP) is concerned with how technology can meaningfully interpret and act on human language inputs. NLP allows technology such as Amazon's Alexa to understand what you're saying and how to react to it. Without NLP, AI that requires language inputs is relatively useless.

Natural Language Processing is what allows chatbots to understand your messages and respond appropriately. When you send a message with "Hello", it is the NLP that lets the chatbot know that you've posted a standard greeting, which in turn allows the chatbot to leverage its AI capabilities to come up with a fitting response. In this case, the chatbot will likely respond with a return greeting.

Without Natural Language Processing, a chatbot can't meaningfully differentiate between the responses "Hello" and "Goodbye". To a chatbot without NLP, "Hello" and "Goodbye" will both be nothing more than text-based user inputs. Natural Language Processing (NLP) helps provide context and meaning to text-based user inputs so that AI can come up with the best response.

Advanced NLP can even understand the intent of your messages. For example, are you asking a question or making a statement. While this may seem trivial, it can have a profound impact on a chatbot's ability to carry on a successful conversation with a user.

One of the most significant challenges when it comes to chatbots is the fact that users have a blank palette regarding what they can say to the chatbot. While you can try to predict what users will and will not say, there are bound to be conversations that you would never imagine in your wildest dreams.

While Natural Language Processing (NLP) certainly can't work miracles and ensure a chatbot appropriately responds to every message, it is powerful enough to make-or-break a chatbot's success. Don't underestimate this critical and often overlooked aspect of chatbots.

1.3.4 Applications of chatbot

In recent days, chatbots are being implemented in many sectors be it in technical websites, be it any customer service centric companies' websites, or be it any android apps where one can book a train, do recharges, bill payments etc. It is ideal to have a chatbot if your business falls into any of the following categories:

- Serving many requests that are similar to each other
- Sells products that need low involvement
- Is in a highly competitive niche
- for Better Customer Experience
- for Conversational Commerce
- for Personal Assistance
- for Data Gathering and Analysis

1.4 Challenges in Chatbot

We are seeing revolution in chatbot development in the way customers interact with business now-a-days. According to recent research, “47% of consumers are open to buying items through a chatbot”. Thus, most of the customer-centric organisations have implemented their business process in augmenting or building these virtual agents and are attaching it to their websites. But the path towards this much spread adoption of chatbots in every field of our lives is not at all picture perfect. There are many pitfalls and roadblocks to overcome. So, they tend to overlook few critical aspects during chatbot development. Here are 4 big challenges that companies face while building their bots:

1.4.1 Context in chatbots:

The key to an intelligent chatbot is its proper association with its context of user input and meaningful responses to the user, as conversation without any context would be useless. The challenge is for companies is to develop and maintain the memory of bots that can offer user specific responses. That's when NLP comes into the picture and overcome the challenge of understanding the depth of conversation; up-to an extent. NLU, an underlying concept of NLP, try to read the databases and data sets when

bots are structured, in predefined sequential order and then converts it into a language that users understand.

However, in real life, humans do not interact in a defined pre-specified order, as a result intelligent slot filling, which stores the preferences of the regular users is the alternative to maintain the memory of a bot effectively. This can ensure that our virtual agents are not interacting in the same old predefined order but in a more personalized fashion. Example:

User: Can you please tell me which flights from London are expected this evening after six?

Bot: Sure, let me check

Bot: Here are the flights:
EK 201, EK 522

User: and which one's tomorrow morning before twelve?

Bot: Sorry, can you please say your query again?

1.4.2 User's limited attention:

Users, now, have very short time for their queries to be answered and expect fast replies. It's quite challenging for tech companies to develop chatbots, which holds user's attention till the end of conversation.

Building conversational UI is therefore important in exhibiting human like conversations and better customer experiences. It initiates interactions to be more social than being monotonous as well as technological in nature. The conversations as a result, should be natural, creative and emotional in order for your chatbot to be successful. In some cases, however a machine wouldn't always render the same empathy that a human could and this is when a human replacement should take care of the user's request.

1.4.3 Chatbot Testing:

Chatbot testing is another main challenge where most of the hardles lie. Chatbots are continuously evolving to more intelligent ones due to its upgradation in natural language models. Thus, it is more important to test chatbot's accuracy. It will depend on what type of method you want to examine.

- First method involves automated testing of chatbots. There are many automation testing platforms like Zypnos, TestyourBot, Bot Testing, Dimon etc. These platforms allow detailed reports of the results and coding of test scripts, which could be run for all the test cases.
- The other method involves testing of conversational logic i.e. manual testing executed by a closed group of testers. They act as users and check the bot for all the unexpected slots possible. This method can be time consuming and partially accurate. However, it has its own benefits that outrange automation, by checking the logic against human conversations.

1.4.4 Viability of Data

There is no point of having lots of data, intelligent slot filling or technologically advanced chatbot- if it actually doesn't deliver the USP of your organisation. It is vital for a chatbot to not only be enriched with meaningful data that is matching our work that needed to be done by our bot, It is also to be equipped to deliver the brand identity to our target audience. In order to check the viability of our virtual agents we should consider asking ourselves the following:

- Are our virtual agents delivering to the right audience?
- Does it offer business goals uniquely?
- How is it different from other chatbots?

1.5 Objectives

The main objective here is to build some models which will help in partial development of the intelligent chatbot that will help commuters of India who travels in trains. The chatbot will be able to provide all train related live informations to users. For that, we aim for developing a dataset for training the stepping stone of the project which is intent classification model, then using the parameter values of every user input we will develop the dialogue manager. With the help of a entity recognizer we will also be able to generate proper response to the user. So, at first, taking user's input into consideration, the chatbot will try to understand its intent and will be providing response depending on user's need.

1.6 Contributions

Our contributions towards building an information providing intelligent chatbot are building the below models:

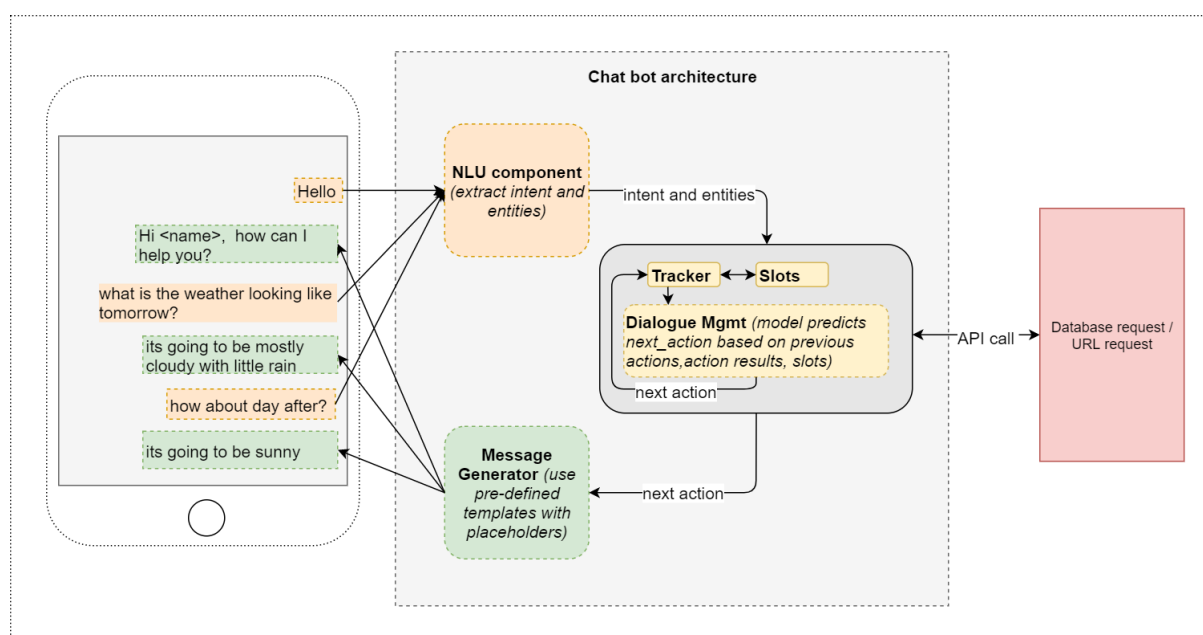
- Data Collection Framework
- Intent Classification Model
- Named Entity recognition model for slot filling
- Response Generation using web API

1.7 Thesis Overview

Dataset collection framework, intent classification model based on CNN and LSTM, Entity Recognizer model and response generation utilizing railway based API are the overview of the entire work done here.

1.7.1 Overall Architecture

The ML driven model learns the actions based on the training data provided (unlike a traditional state machine based architecture that is based on coding all the possible if-else conditions for each possible state of the conversation.) Here is a high level overview of such an architecture for a chat-bot.



First, let's see what all things it is needed to determine an appropriate response at any given moment of the conversational flow?

We need to know the user's intent—these are called as intents. Few examples of intents are—‘request weather’, ‘request restaurant’ etc., The intent in the above example is ‘request weather’.

Here we need to know the specific intents of the request, for example—the information we get asking questions like when? where? how many? to the user input etc., that correspond to extracting the information from the user request about date, time, location, number respectively. Here date, time, location, number are the entities. Quoting the above weather example, the entities can be ‘date-time’ (user provided information) and location (note—location need not be an explicit input provided by the user and will be determined from the user location as default, if nothing is specified). The intent and the entities together will help to make a corresponding API call to a weather service and retrieve the results, as we will see later.

Now refer to the above figure, and the box that represents the NLU component (Natural Language Understanding) helps in extracting the intent and entities from the user request.

NLU component constitutes —

A supervised intent classification model that is trained on varieties of sentences as input and intents as target. Typically, a linear SVM will be enough as an intent classification model.

Entity extraction model—This can be a pre-trained model like Stanford NLP library (OR) it can be trained using some probabilistic models like CRF (conditional random fields).

3. Now, since this is a conversational AI bot, we need to keep track of the conversations happened thus far, to predict an appropriate response. For this purpose, I need a dictionary object that can be persisted with information about the current intent, current entities, persisted information that user would have provided to bot's previous questions, bot's previous action, results of the API call (if any). This information will constitute our input X , the feature vector. The target y , that the dialogue model is going to be trained upon will be ‘next action’ (The next action can simply be a one-hot encoded vector corresponding to each actions that we define in our training data).

Then, that brings me to the next question—how do we get the training values for my feature vector, input X ?

Getting the information regarding the intent and entities is straightforward as I have seen from the NLU component.

Getting the remaining values (information that user would have provided to bot's previous questions, bot's previous action, results of the API call etc.,) is little bit tricky and here is where the dialogue manager component takes over. These feature values will need to be extracted from the training data that the user will define in the form of sample conversations between the user and the bot. These sample conversations should be prepared in such a fashion that they capture most of the possible conversational flows while pretending to be both a user and a bot.

1.7.2 Dataset Collection

For this, a data collection framework has been generated using Facebook Messenger, Dialogflow, Heroku server and my SQL Workbench which helps to build a labeled dataset.

1.7.3 Intent Classification

Using the labeled data that we have got from the data collection framework, we trained two intent classification models, based on CNN and LSTM respectively. Using basic CNN and LSTM model, two intent classification models have been introduced which will be able to classify user input based on their intent, thus embarking the first step of the chatbot development.

1.7.4 Entity Recognition Model and Response Generation

After getting the right intent of a user input, chatbot does need the entity parameter values (sometimes called 'slot' values) of that intent. What are intents and its entity values are already discussed above. To get the proper values of each entity of the intent, we need a proper entity recognition model. Here we have used an entity recognition model provided by SPACY and using it, we are able to get out the entity parameters values. Then, using the entity values, chatbot will be able to generate the appropriate response to the user. The entity values are also very important for developing the dialogue tracker system. As discussed above, context monitoring is a major challenge in modern day's chatbot systems. So, entity recognition model plays an important developing a personalized chatbot.

Chapter 2:

Here we will discuss some research works that have been done over recent years.

Related Work

2.1 Previous Contributions

Starting in 1966 with the introduction of the ELIZA chatbot, a great deal of effort has been devoted towards the goal of developing a chatbot system that would be able to pass the Turing Test. These efforts have resulted in the creation of a variety of technologies and have taken a variety of approaches. Now I will discuss and compare the different technologies used in the chatbots which have won the Loebner Prize Competition, the first formal instantiation of the Turing Test.

The very first known chatbot was Eliza, which was developed in 1966. Its goal was to behave as a Rogerian psychologist. It used simple pattern matching and mostly returned users sentences in a form of questions. Its conversational ability was not very good, but it was enough to confuse people at a time when they were not used to interact with computers and to start the development of other chatbot systems. The very first online implementation of Eliza was done by the researches at Jozef Stefan Institute in Ljubljana, Slovenia and is still available¹ for testing.

The first such a system that was actually evaluated using some sort of Turing Test was PARRY (Colby, 1975). Parry was designed to talk as a paranoid person. Its transcripts were given to psychiatrists together with transcripts from real paranoia patients for comparison. The psychiatrists were able to make the correct identification only 48% of the time.

Here in most of the bots, pre-specified responses were taken into consideration. Most of them were able to give a specified set of responses stored in a database. But in today's AI world, we want a chatbot that will try to be more human like.

2.1.1 What is Turing Test?

- Alan Turing proposed the Turing Test as a replacement for the question “Can machines think?”
- Turing's aim is to provide a method to assess whether or not a machine can think.
- Also known as the **Imitation Game**.
- Conversation with a chatbot should be indistinguishable compare to a conversation with human.
- The test: A man (A), a woman (B) and an interrogator (C) chat. The objective of the interrogator is to determine which of the other two is the woman. If a machine (bot) chats instead of A or B and fools the interrogator, it has passed the Turing test.

2.1.2 What Is Loebner Prize?

- The Loebner Prize is an annual competition in artificial intelligence that awards prizes to the computer programs considered by the judges to be the most human-like.
- Based on standard Turing Test.

In each round, a human judge simultaneously holds textual conversations with a computer program and a human being via computer.

2.1.3 Official List of Winners of Loebner Prize

Year	Chatbot	Technologies	Language Trick
1991	PC Therapist III (Weintraub, 1986)	parsing, pattern matching, word vocabulary, remembers sentences	non sequitur, canned responses
1992	PC Professor (Weintraub, 1986)		
1993	PC Politician (Weintraub, 1986)		
1994	TIPS (Whalen, 1994; Hutchens, 1997)	Pattern matching, database like system	Model of personal history
1995	PC Therapist (Weintraub, 1986)	Same as in 1991	Same as in 1991
1996	HeX (Hutchens, 1997)	Pattern matching, Markov chain models to construct some replies	database of trick sentences, Model of personal history, not repeating itself
1997	CONVERSE (Levy, 1997)	Statistical parser, pattern matching, modular with weighted modules, WordNet synonyms, list of proper names, ontology, database for storing facts	Proactivity
1998	Albert One (Garner, 1995)	Pattern matching, hierarchical composition of other chat bots (Eliza, Fred,Sextalk)	Proactive monologues

Year	Chatbot	Technologies	Language Trick
1999	Albert One (Garner, 1995)	Pattern matching, hierarchical composition of other chat bots (Eliza, Fred, Sextalk)	Proactive monologues
2000	A.L.I.C.E (Wallace, 2003)	AIML (Artificial Intelligence Mark-up Language – advanced pattern matching)	
2001	A.L.I.C.E (Wallace, 2003)		
2002	Ella (Copple, 2009)	Pattern matching, phrase normalization, abbreviation expansion, WordNet	Monologues, not repeating itself, identify gibberish, play knock-knock jokes
2003	Jabberwock (Pirner, 2003)	Parser, pattern matching – simpler than AIML, Markov Chains, Context free grammar (CFG)	
2004	A.L.I.C.E (Wallace, 2003)	Same as in 2000	
2005	George (Carpenter, 2006)	Based on Jabberwocky chatbot. No pattern matching or scripts. Huge database of people’s responses.	
2006	Joan (Carpenter, 2006)		
2007	UltraHAL by Robert Medeksza*	Combination of VB code and pattern matching scripts	
2008	Elbot (Roberts, 2007)*	Commercial NLI system	
2009	Do-Much-More (Levy, 2009)*	Commercial property of Intelligent Toys Ltd.	

Year	Chatbot	Technologies	Language Trick
2010	Suzette (Wilcox, 2011)	ChatScript (AIML successor. Concepts, triples, variables)	
2011	Rosette (Wilcox, 2011)		
2012	Chip Vivant (Embar, 2011)	Not publicly disclosed. Common ontology and AI, responses taken from ChatScript (but not in ChatScript format or engine).	
2013	Mitsuku(Steve Worswick)	Based On AIML technology	
2014	Rose(Bruce Wilcox)	Same as Rosette(2011), but an improved version.	
2015	Rose(Bruce Wilcox)	---Same as above---	
2016	Mitsuku(Steve Worswick)	Based On AIML technology	
2017	Mitsuku(Steve Worswick)	---Same as above---	
Note: The winners marked with asterisk (*) are commercial programs and thus their technologies and internal structure is not publicly available.			

2.2 Winning chatbots and Comparative Analysis

Regardless of the fact that none of the existing chatbots were able to pass the Turing Test, each year there is a winner of the Loebner Prize Competition that appears most human from all the competing chatbots. The list of each year's winners together with the used technologies can be seen on the Table 1. We tried to separate the technology part into the technical approaches and algorithms and the language and approach tricks used to confuse judges on the Turing Test. The technologies are explained in the following chapters in more detail. The winners marked with asterisk (*) are commercial programs and thus their technologies and internal structure is not publicly available.

Chapter 3: Dataset Collection

Here we have built a framework using facebook messenger, dialogflow,heroku server/ngrok server,dbms system by which we have collected the dataset required for training our intent classification model and entity recognition model for generating proper responses.

3.1 Existing Datasets

3.1.1 Reddit Dataset:

https://www.reddit.com/r/datasets/comments/3bxlg7/i_have_every_publicly_available_reddit_comment/?sh=69e4fee7&st=j9udbxta

The dataset is ~1.7 billion JSON objects complete with the comment, score, author, subreddit, position in comment tree and other fields that are available through Reddit's API. **It's now a torrent. This can give me an opportunity to examine the data. The file is structured with JSON blocks delimited by new lines (\n).**

Example JSON Block:

```
{ "gilded":0, "author_flair_text": "Male", "author_flair_css_class": "male", "retrieved_on": 1425124228, "ups": 3, "subreddit_id": "t5_2s30g", "edited": false, "controversiality": 0, "parent_id": "t1_cnapn0k", "subreddit": "AskMen", "body": "I can't agree with passing the blame, but I'm glad to hear it's at least helping you with the anxiety. I went the other direction and started taking responsibility for everything. I had to realize that people make mistakes including myself and it's gonna be alright. I don't have to be shackled to my mistakes and I don't have to be afraid of making them.", "created_utc": "1420070668", "downs": 0, "score": 3, "author": "TheDukeofEtown", "archived": false, "distinguished": null, "id": "cnasd6x", "score_hidden": false, "name": "t1_cnasd6x", "link_id": "t3_2qyhmp" }
```

3.1.2 rDany Chat Dataset :

<https://www.kaggle.com/eibriell/rdany-conversations>

Context: It is a virtual companion conversation dataset, capable of holding long and interesting conversations. But the lack of a good technique, and good datasets apparently is holding the advances of AI in that sense. Chatbots don't have personality, nor context awareness, and datasets used to train them are just pair of question/answers, or IT conversations. This dataset is being built using rDany bot for Telegram, Kik and Messenger.

This bot have a personality: Candid, True, Fun, Optimistic, Empathic, Gender neutral, Likes art

And knows a very limited word, its room, Wikipedia, and a schematic view of the world. And speaks Spanish (native), English (with some errors), and other languages (using automatic translation). You can learn more about it on rDany's Telegram channel: <https://t.me/rDany>

3.1.3 Ubuntu Dialogue Corpus – (26 million turns from natural two person dialogues)

<https://www.kaggle.com/rtatman/ubuntu-dialogue-corpus>

This corpus of naturally-occurring dialogues can be helpful for building and evaluating dialogue systems.

The new Ubuntu Dialogue Corpus consists of almost one million two-person conversations extracted from the Ubuntu chat logs, used to receive technical support for various Ubuntu-related problems. The conversations have an average of 8 turns each, with a minimum of 3 turns. All conversations are carried out in text form (not audio).

The full dataset contains 930,000 dialogues and over 100,000,000 words and is available [here](#). This dataset contains a sample of this dataset spread across .csv files. This dataset contains more than 269 million words of text, spread out over 26 million turns.

- Folder : The folder that a dialogue comes from. Each file contains dialogues from one folder .
- dialogueID : An ID number for a specific dialogue. Dialogue ID's are reused across folders.
- Date : A timestamp of the time this line of dialogue was sent.

- From : The user who sent that line of dialogue.
- To : The user to whom they were replying. On the first turn of a dialogue, this field is blank.
- Text : The text of that turn of dialogue, separated by double quotes (“). Line breaks (\n) have been removed.

3.1.4 Deep NLP dataset in NLP

<https://www.kaggle.com/samdeephlearning/deepnlp>

Contains:

- Sheet_1.csv contains 80 user responses, in the response_text column, to a therapy chatbot. Bot said: 'Describe a time when you have acted as a resource for someone else'. User responded. If a response is 'not flagged', the user can continue talking to the bot. If it is 'flagged', the user is referred to help.
- Sheet_2.csv contains 125 resumes, in the resume_text column. Resumes were queried from Indeed.com with keyword 'data scientist', location 'Vermont'. If a resume is 'not flagged', the applicant can submit a modified resume version at a later date. If it is 'flagged', the applicant is invited to interview.

We can do:

- Classify new resumes/responses as flagged or not flagged.
- There are two sets of data here - resumes and responses. Split the data into a train set and a test set to test the accuracy of your classifier. Bonus points for using the same classifier for both problems.

3.2 Taxonomy of Models in Chatbot

3.2.1 Retrieval-based models (easier)

It uses a repository of predefined responses and some kind of heuristic to pick an appropriate response based on the input and context. The heuristic could be as simple as a rule-based expression match, or as complex as an ensemble of Machine Learning classifiers. These systems don't generate any new text, they just pick a response from a fixed set.

3.2.2 Generative models (harder)

It doesn't rely on pre-defined responses. They generate new responses from scratch. Generative models are typically based on Machine Translation techniques, but instead of translating from one language to another, we "translate" from an input to an output (response).

Both approaches have some obvious pros and cons. Due to the repository of handcrafted responses, retrieval-based methods don't make grammatical mistakes. However, they may be unable to handle unseen cases for which no appropriate predefined response exists. For the same reasons, these models can't refer back to contextual entity information like names mentioned earlier in the conversation. Generative models are "smarter". They can refer back to entities in the input and give the impression that you're talking to a human. However, these models are hard to train, are quite likely to make grammatical mistakes (especially on longer sentences), and typically require huge amounts of training data.

Deep Learning techniques can be used for both retrieval-based or generative models, but research seems to be moving into the generative direction. Deep Learning architectures like Sequence to Sequence are uniquely suited for generating text and researchers are hoping to make rapid progress in this area. However, we're still at the early stages of building generative models that work reasonably well. Production systems are more likely to be retrieval-based for now.

3.3 Closed Domain vs. Open Domain

In an **open domain (harder)** settings, the user can take the conversation with chatbot anywhere, any time in any context. There isn't necessarily a well-defined goal or intention of the chatbot to accomplish. Conversations on social media sites like Twitter and Reddit are typically open domain – they can go into all kinds of directions. The infinite number of topics and the fact that a certain amount of world knowledge is required to create reasonable responses makes this a hard problem.

In a **closed domain (easier)** settings, the space of possible inputs and outputs is somewhat limited because the system is trying to achieve a very specific goal. Technical Customer Support or Shopping Assistants are examples of closed domain problems. These systems don't need to be able to talk about politics, they just need to fulfill their specific task as efficiently as possible. Sure, users can still take the conversation anywhere they want, but the system isn't required to handle all these cases – and the users don't expect it to.

3.4 Restrictions of Previous Datasets

As we have seen, all the previously discussed labeled datasets and all the available open source datasets are not suitable for our specific chatbot that are destined to provide information on trains to daily commuters. For intent classification model, a proper labeled dataset is needed for training the model. From a train related query of user, we are only able to get the slot values that will be helpful while building the entity recognition model, dialogue tracker model as well as for response generation part.

3.5 Data Collection Framework

For this We have developed a framework of Chabot using Facebook messenger application(we can use any popular messaging platform), Google's Dialogflow System, Heroku Server, MySQL Workbench and node.js for the coding part. As we have needed the labeled data from train information perspective, a Chabot was needed. Many chatbots leverage Natural Language Processing (NLP) to interpret the intent of a customer's input, allowing the bot to give an accurate response. Implementing NLP in our bot can be pretty difficult, but there are several platforms that make it much easier. The steps that I followed to develop this Chabot are as follows :

Note : For this process, we need to have, facebook application(in mobile or PC), Node.js, MySQL Workbench pre-installed in PC, a properly running server.

- we installed Node.js and npm in our system.
- We created a server that will be provided as a webhook when setting up the bot. For server I used heroku. One can use the free ngrok server, but it has limitation on time of use. The webhook provides the core of your chatbot experience as it is how your bot gets notified of the various interactions and events that happen, including when someone sends a message.
- We created a new directory for the bot we will build, call it *sample-bot* or whatever you like. Inside the newly created directory, run the following command in the terminal to initialize the project with a *package.json* file.

```
npm init
```

- We filled out the necessary information in the prompt that follows, then create an *src* folder at the root of your project directory.
- We then, setup *Express* as our web server and *body-parser* to parse incoming request bodies. We installed them as dependencies first:

```
npm install body-parser express
```

- We created an `index.js` file in the `src` directory and paste in the following code to setup Express on port 5000. We can also choose another port if 5000 is already in use on our machine.

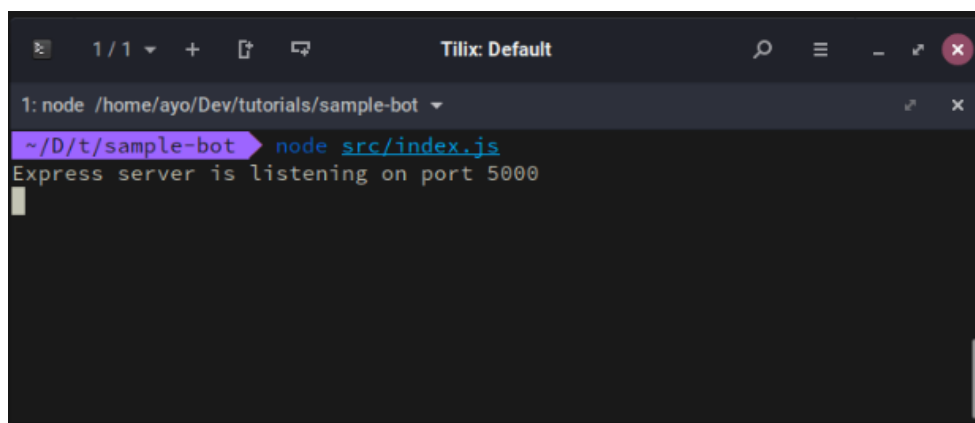
```
const express = require('express');
const bodyParser = require('body-parser');

const app = express();

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));

app.listen(5000, () => console.log('Express server is listening on port 5000'));
```

- We saved the file and run `node src/index.js` from the root of your project directory to start the server.
- If everything goes correct and the dependencies are installed properly, we can see the following message printed out in our terminal.



Set up Facebook verification endpoint

During setting up a webhook for our chatbot, Facebook requires a verification step to ensure that the webhook is working genuinely. This is how it works:

1. We create a token which is simply a random string of our choice and hardcode it to our webhook.
2. We provide this token when subscribing our webhook to receive events for our bot.
3. Facebook sends a GET request to our webhook with the token we have provided in the previous step in the `hub.verify` parameter of the request.

4. Our webhook will verify that the token is correct and send the appropriate response back to Facebook.
5. Finally, Messenger will subscribe the webhook to the bot.

- We create a file called `verify-webhook.js` in the `src` directory and paste in the following code:

```
const verifyWebhook = (req, res) => {
  let VERIFY_TOKEN = 'pusher-bot';

  let mode = req.query['hub.mode'];
  let token = req.query['hub.verify_token'];
  let challenge = req.query['hub.challenge'];

  if (mode && token === VERIFY_TOKEN) {
    res.status(200).send(challenge);
  } else {
    res.sendStatus(403);
  }
};

module.exports = verifyWebhook;
```

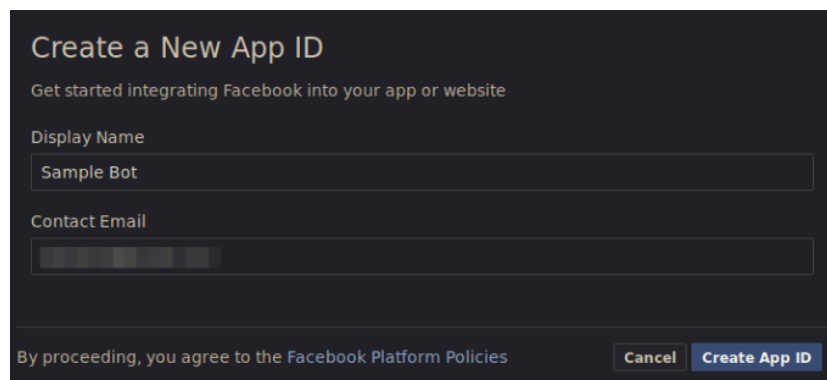
- Here the token is set to `pusher-bot` in this example. We can set it to anything we like, but we need to make sure to take note of it as we will need it later when setting up our app on Facebook Messenger. Next, set up the endpoint that will receive events from the Facebook messenger. Add the following code to your `index.js` file:

```
const verifyWebhook = require('./verify-webhook');

app.get('/', verifyWebhook);
```

Set up a Facebook application

- We set up a facebook page and a particular facebook app that uses messenger services, then connect between them.



Create a New App ID
Get started integrating Facebook into your app or website

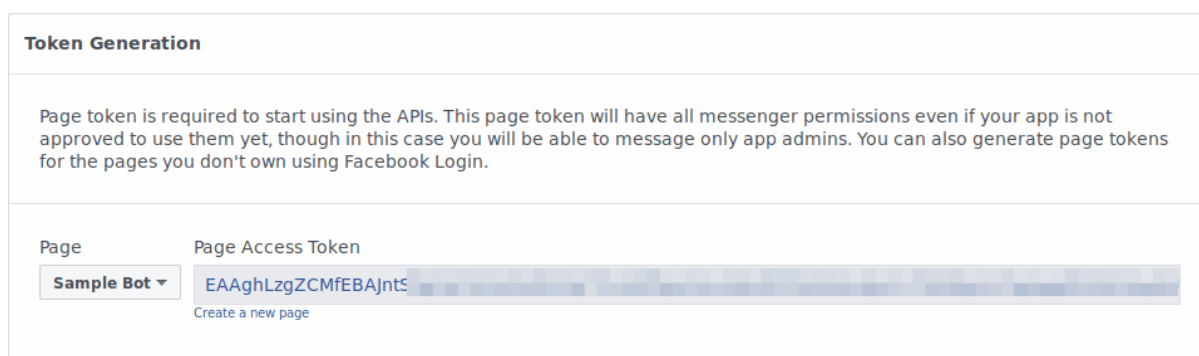
Display Name
Sample Bot

Contact Email

By proceeding, you agree to the Facebook Platform Policies

Once done, we go to the dashboard of the app, just created. Next, we need to add a product to our app. Scroll down the page a bit and find the **Messenger** option then click the **Set Up** button. This will redirect you to the Messenger Platform.

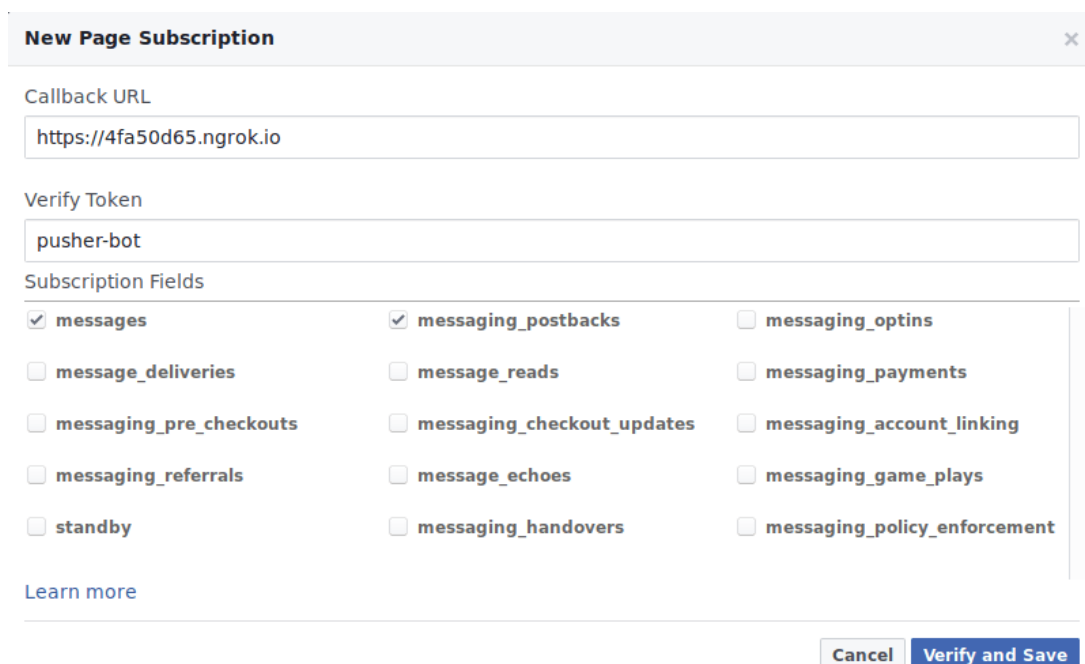
Once there, in the “Token Generation” section and connect your app to a Facebook Page. This will generate a Page Access token that we will make use of later.



The screenshot shows the 'Token Generation' section of the Facebook developer dashboard. It contains a text block explaining that a page token is required for API use and that it will have all messenger permissions. Below this, there is a 'Page' dropdown menu set to 'Sample Bot' and a 'Page Access Token' input field containing a long alphanumeric string. A 'Create a new page' link is visible below the token field.

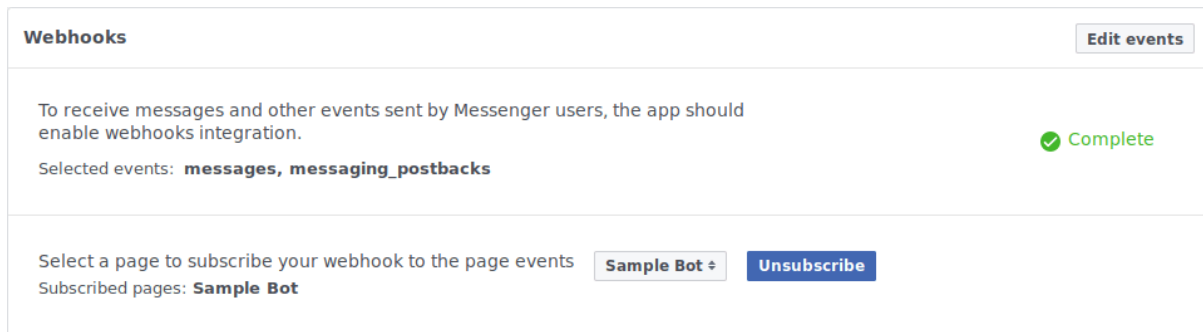
Next, in the **Webhooks** section, we click on **Setup Webhooks**. A popup window appears and we fill that as below.

In the Callback URL field, add the URL that was provided by ngrok in the previous step (the HTTPS version). Add the token we have created in *verify-webhook.js* in the verify token field (‘pusher-bot’ in my case) then select **messages** and **messaging_postbacks** under Subscription Fields.



The screenshot shows a 'New Page Subscription' popup window. It has a title bar with a close button. The 'Callback URL' field contains 'https://4fa50d65.ngrok.io'. The 'Verify Token' field contains 'pusher-bot'. Under 'Subscription Fields', there are three columns of checkboxes. The first column has 'messages' checked. The second column has 'messaging_postbacks' checked. The third column has 'messaging_optins' checked. At the bottom, there are 'Cancel' and 'Verify and Save' buttons.

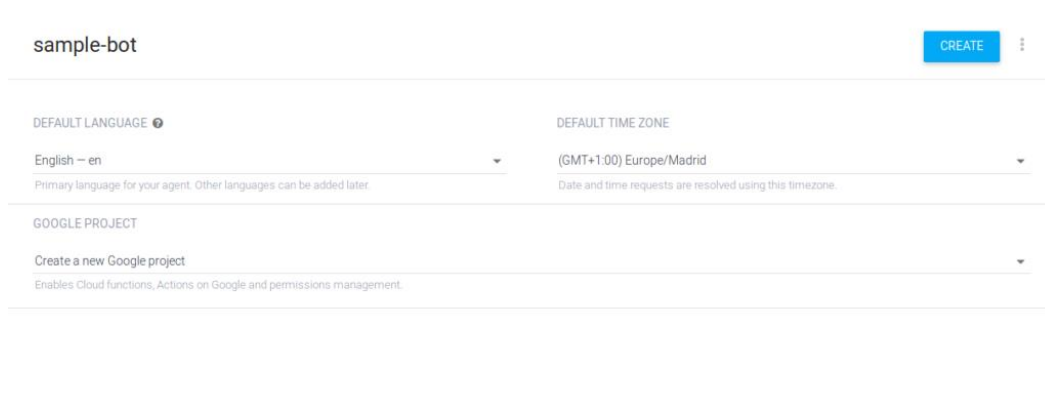
We hit **Verify and Save**. This will verify that the provided token matches the one in our code and save our webhook. Next, we need to subscribe our page to receive webhook events as shown below.



The screenshot shows the 'Webhooks' configuration page. At the top left is the title 'Webhooks' and at the top right is an 'Edit events' button. The main content area contains the following text: 'To receive messages and other events sent by Messenger users, the app should enable webhooks integration.' To the right of this text is a green checkmark and the word 'Complete'. Below this, it says 'Selected events: **messages, messaging_postbacks**'. At the bottom, there is a section for selecting a page to subscribe to, with the text 'Select a page to subscribe your webhook to the page events' and a dropdown menu currently showing 'Sample Bot'. To the right of the dropdown is an 'Unsubscribe' button. Below the dropdown, it says 'Subscribed pages: **Sample Bot**'.

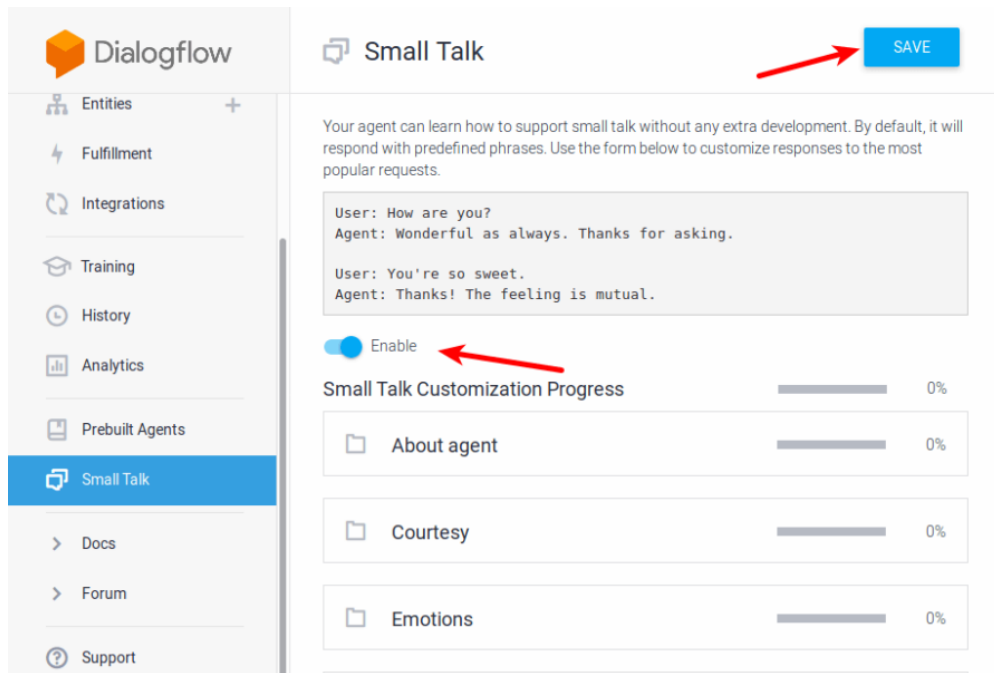
Set up Dialogflow integration :

- We go to dialogflow website and open up a free account. After exposing our server to the web, we start the dialogflow integration. In our bot, all the NLP tasks, that are required, are done by dialogflow.
- Click on the **Create Agent** button. Give your agent a name and fill in the remaining fields, then hit the **CREATE** button. Dialogflow should redirect you to the main page of the agent.



The screenshot shows the 'Create Agent' form in Dialogflow. The form title is 'sample-bot' and there is a blue 'CREATE' button on the right. The form has three main sections: 'DEFAULT LANGUAGE', 'DEFAULT TIME ZONE', and 'GOOGLE PROJECT'. The 'DEFAULT LANGUAGE' section shows 'English - en' with a dropdown arrow and a note: 'Primary language for your agent. Other languages can be added later'. The 'DEFAULT TIME ZONE' section shows '(GMT+1:00) Europe/Madrid' with a dropdown arrow and a note: 'Date and time requests are resolved using this timezone.'. The 'GOOGLE PROJECT' section shows 'Create a new Google project' with a dropdown arrow and a note: 'Enables Cloud functions, Actions on Google and permissions management.'.

- Dialogflow has a feature that gives our bot the ability to have simple conversations with users without writing any code. On the sidebar of the page, we click the **Small Talk** option and enable it for our bot.



- Before we integrate Dialogflow with our bot, we need to set up authentication first.

To do this, go to agent's settings. Under **Google Project**, click on the service account name. This will open your Google Cloud Platform service account's page.



- We see a **Dialogflow Integration** service account in the list of accounts. We click on the three dots menu on the right and select **Create Key**. Leave the file format as JSON and click **Create**. This will download a JSON file to your computer. After opening the JSON file with our favorite text editor, we only need two of the values: `private_key` and `client_email`. Store these values as environmental variables before proceeding.
- We can use the `dotenv` package to load environmental variables from a `.env` file into `process.env`.

```
npm install --save dotenv
```

- And add the following line at the top of `index.js`


```
require('dotenv').config({ path: 'variables.env' });
```

- Then create a `variables.env` file in the root of our project directory. We should add this file to our `.gitignore` so that we do not commit it to our repository by accident. Here's how our `variables.env` file should look like:

```
FACEBOOK_ACCESS_TOKEN=EAAghLzgZCMfEBAJntS...  
DIALOGFLOW_PRIVATE_KEY="-----BEGIN PRIVATE KEY-----\n<KEY>\n-----  
END PRIVATE KEY-----\n"  
DIALOGFLOW_CLIENT_EMAIL=foo@<PROJECT_ID>.iam.gserviceaccount.com
```

- Remember the Page Access Token received from Facebook earlier? Don't forget to add it to our `variables.env` file as shown above.

Token Generation

Page token is required to start using the APIs. This page token will have all messenger permissions even if your app is not approved to use them yet, though in this case you will be able to message only app admins. You can also generate page tokens for the pages you don't own using Facebook Login.

Page	Page Access Token
Sample Bot ▾	EAAghLzgZCMfEBAJntS <small>Create a new page</small>

- The next step is to use this feature in our messenger bot. Dialogflow provides an SDK for Node.js which we can add to our project by running the following command:

```
npm install --save dialogflow
```

- Also, grab `node-fetch` so that we can use that to send requests to the Facebook API:

```
npm install --save node-fetch
```

- Next, we create a file called `process-message.js` in our `src` directory and paste in the following contents:

```

const fetch = require('node-fetch');

// You can find your project ID in your Dialogflow agent settings
const projectId = ''; //https://dialogflow.com/docs/agents#settings
const sessionId = '123456';
const languageCode = 'en-US';

const dialogflow = require('dialogflow');

const config = {
  credentials: {
    private_key: process.env.DIALOGFLOW_PRIVATE_KEY,
    client_email: process.env.DIALOGFLOW_CLIENT_EMAIL
  }
};

const sessionClient = new dialogflow.SessionsClient(config);

const sessionPath = sessionClient.sessionPath(projectId, sessionId);

// Remember the Page Access Token you got from Facebook earlier?
// Don't forget to add it to your `variables.env` file.
const { FACEBOOK_ACCESS_TOKEN } = process.env;

const sendTextMessage = (userId, text) => {
  return fetch(
    `https://graph.facebook.com/v2.6/me/messages?access_token=${FACEBOOK_ACCESS_TOKEN}`,
    {
      headers: {
        'Content-Type': 'application/json',
      },
      method: 'POST',
      body: JSON.stringify({
        messaging_type: 'RESPONSE',
        recipient: {
          id: userId,
        },
        message: {
          text,
        },
      }),
    },
  );
};

```

```

    }),
  }
);
}

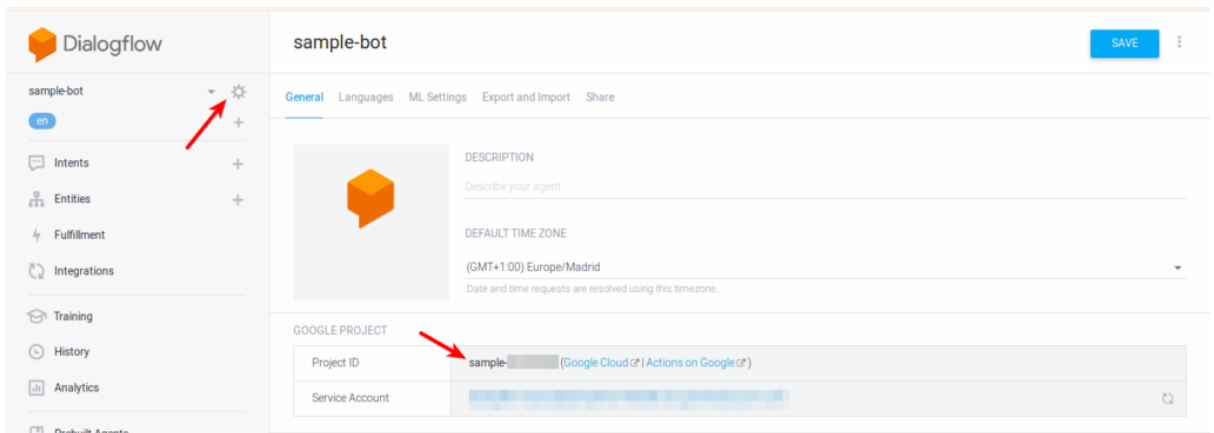
module.exports = (event) => {
  const userId = event.sender.id;
  const message = event.message.text;

  const request = {
    session: sessionPath,
    queryInput: {
      text: {
        text: message,
        languageCode: languageCode,
      },
    },
  };

  sessionClient
    .detectIntent(request)
    .then(responses => {
      const result = responses[0].queryResult;
      return sendTextMessage(userId, result.fulfillmentText);
    })
    .catch(err => {
      console.error('ERROR:', err);
    });
}

```

- Note: find our agent's project ID in the [Dialogflow settings](#) and enter it as the value of projectId above.



- If we look at the code in `process-message.js`, we will see that we passed the text message received from Facebook Messenger to Dialogflow. We can then extract the matching response from Dialogflow and send the result back to Messenger.
- To receive message events from Facebook, we need to add the following to our `index.js` file:

```
const messageWebhook = require('./message-webhook');
```

```
app.post('/', messageWebhook);
```

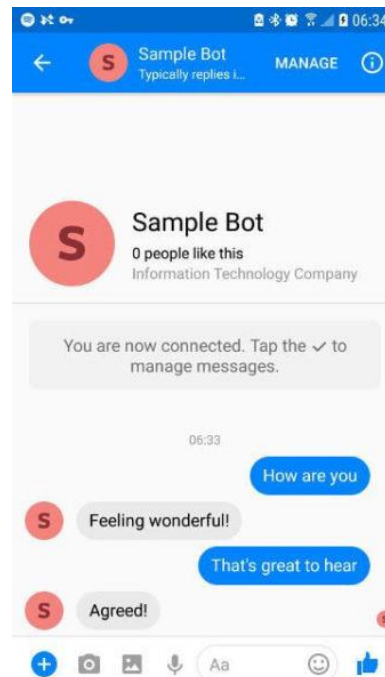
- We create the `message-webhook.js` file in our `src` folder then paste in the following code:

```
const processMessage = require('./process-message');
```

```
module.exports = (req, res) => {
  if (req.body.object === 'page') {
    req.body.entry.forEach(entry => {
      entry.messaging.forEach(event => {
        if (event.message && event.message.text) {
          processMessage(event);
        }
      });
    });
  }
};
```

```
res.status(200).end();
}
};
```

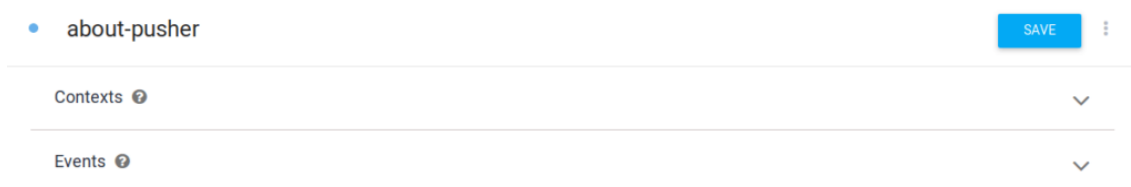
- Now we can go ahead and test our bot! We can send simple messages like “What’s up?”, or “How are you” and we will get replies from the bot. These responses are from the Small Talk feature we enabled above.



Create our own chatbot intents

Our bot’s ability to make small talk is all well and good, but that’s probably not what our customers want. To add functionality to our bot, We need to create intents. We can read more about intents on the Dialogflow website, but the gist of it is that it helps we map a user’s request with the appropriate response.

To start, locate the ‘Intents’ option on the left and click ‘Create Intent’. We need to give our intent a name. We’ll call mine ‘about-pusher’.



- Now under Training phrases, we need to provide examples of how users may express their request in natural language. The more examples you provide, the more accurate our bot becomes at matching the request with the correct intent.

Training phrases

Search training phrases

” Add user expression

” How does Pusher work

” What does Pusher do

” Talk to me about pusher

” What is Pusher

- Once we have populated the training phrases, we need some responses. These are the replies that will be sent to our bot when this intent is matched. We can hit the Save button at the top of the page after adding a few responses.

Responses

DEFAULT GOOGLE ASSISTANT +

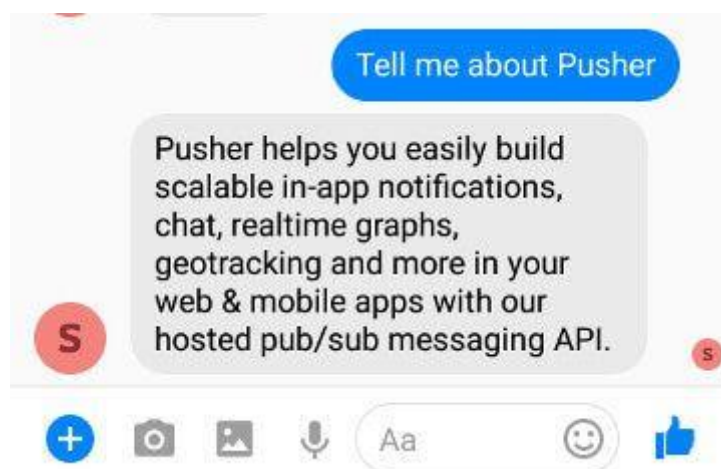
Text response

- 1 Pusher helps you easily build scalable in-app notifications, chat, realtime graphs, geotracking and more in your web & mobile apps with our hosted pub/sub messaging API.
- 2 Pusher is the category leader in delightful APIs for app developers building communication and collaboration features.
- 3 Enter a text response variant

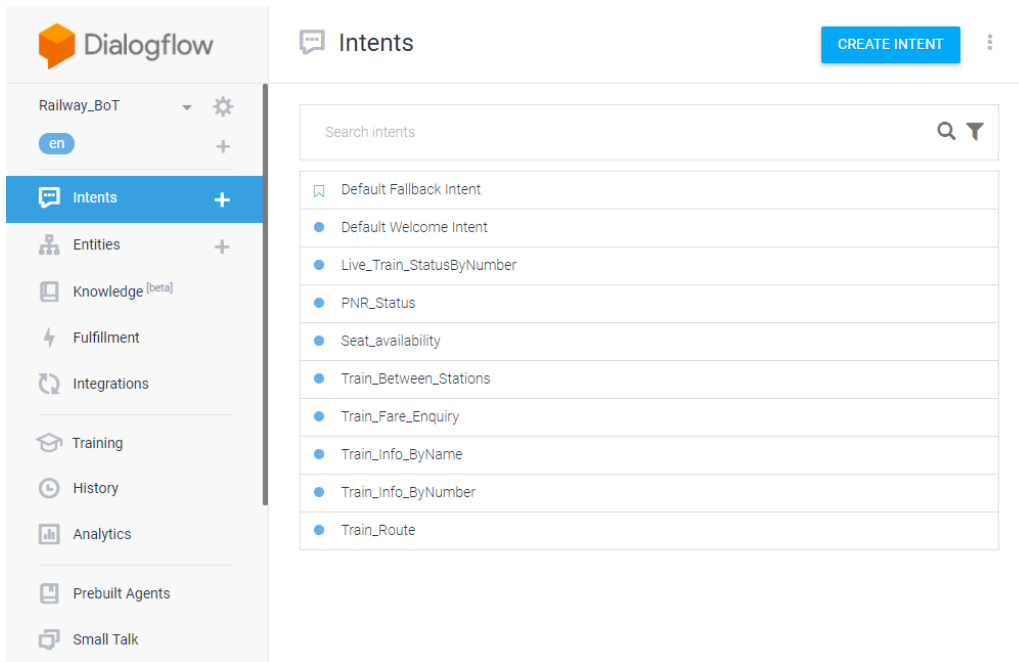
ADD RESPONSES

Set this intent as end of conversation

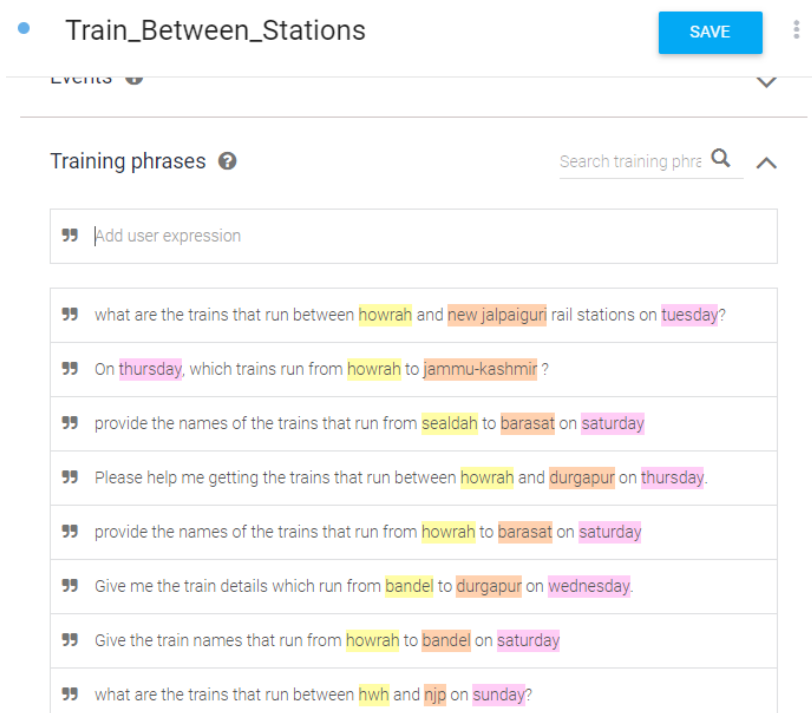
- Finally, try it out in Messenger. It should work just fine.



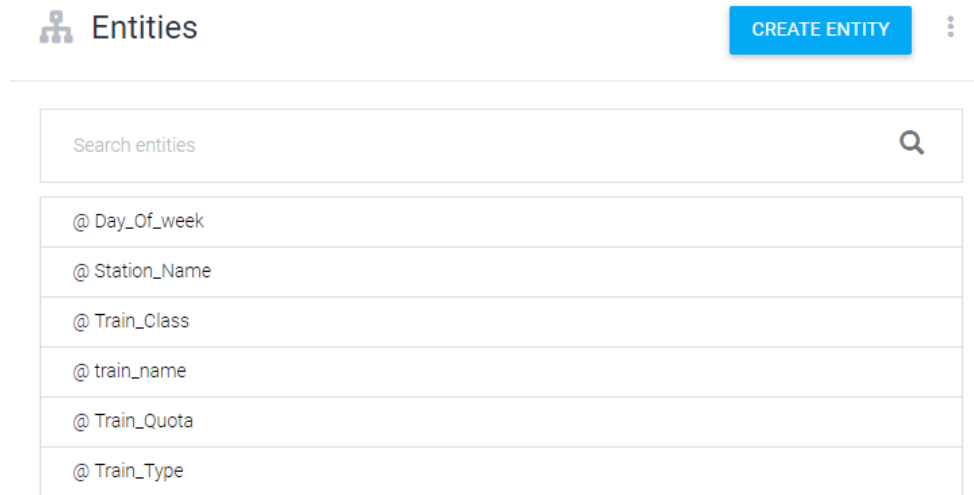
- In dialogflow we have created all the intents that are required for our actual project. And we have also developed the entities for those intents in a proper manner so that we can get the slot values accordingly.
- Under every intent, training phrases can be added, so that messenger bot can be well equipped for handling user responses.
- Our created intents specific to our task are as below:



- How training phrases have been added for an intent manually is shown below:



- Entities of every intents need not be same, We have maintained all the possible entities like below in dialogflow :



Intent vs entities details :

- Live_Train_StatusByNumber < trainno >
- PNR_Status < pnrno >
- Seat_availability < trainno, stnfrom, stnto, quota, class, date >
- Train_Between_Stations < source, dest, day >
- Train_Fare_Enquiry < trainno, source, dest, age, preff, quota, date >
- Train_Info_ByName < train_name >
- Train_Info_ByNumber < trainno >
- Train_Route < trainno >

We have also maintained possible entity values under every entity name in dialogflow as shown above.

3.6 Corpus Statistics

We have collected atleast 100 sentences/user inputs per intent class. We have also taken account of the parameter values or we may call the entity values of each user input under their intent class.

Chapter 4: Intent Classification Model

An important component of chatbot is intent classifier model. We all know that chatbots are day to day helper for us in our lives. Suppose we have an assistant bot and we tell the bot to ‘*book a veg restaurant*’. Now, here our assistant know how to respond for that query because it has the brain that is trained for identifying the sole intent of the query. In a similar manner, we trained our chatbot to respond for a particular query. So in case of chatbots, in order to make them respond according to the users query, we need an “intent classification” model and the categories in which a chatbot responds, are known as “intents”.

In other words, for example, if we have asked “*for booking a cab*” then it will respond under that intent category and if we would have asked for “*booking a flight*” then it will respond under a different intent category and so on.

4.1 Pre-Requisite

4.1.1 Data Preparation

Here we need a labeled dataset where, in the first column we have “User input”, in the second column we have the “intent_name” that is associated with that particular “User input”, and in the third column we have the entity(s) values that will help in building Entity Recognition model later. For now, we need the first two columns of the dataset.

In the field of NLP, data preparation is the most deciding factor. In case of our present work, we have simplified the data as much as possible, which in turn, gives the intent classification model a help, to be trained easily and in a faster manner.

- **Data Cleaning** - “*Data is like crude. It’s valuable, but if unrefined it cannot really be used.*”(by **Clive Humby**, UK Mathematician and architect of Tesco’s

Clubcard, 2006)[1] As we are using raw data, it has been cleaned beforehand. Thus, we removed every punctuation, special characters and all the stop words using vastly used NLTK toolkit's standard functions[2].

- **Tokenization** is done afterwards, so that all the sentences are segmented into words. After this words were converted into lowercased.
- **Lemmatization** is followed after that. "*Lemmatization (or lemmatization) in linguistics, is the process of grouping together the different inflected forms of a word so they can be analyzed as a single item.*" For example :

```
lemmatizer.lemmatize("cats") ==> cat
lemmatizer.lemmatize("churches") ==> church
lemmatizer.lemmatize("abaci") ==> abacus
```

- **Encoding** – After data cleaning, we prepared the lists of words of sentences. To convert these words into indexes, we use them as input to the Tokenizer. This is called **Input Encoding**. After this, we used padding to make them of equal length so that they can be used in the model. For outputs, same procedures have been followed. Indexing of the intents was followed by tokenization. After indexing all the intents, one hot encoding was carried out, so that they can be fed to the intent classification model.
- **Training and Validation Set** – When all the data is ready for the model, the final step is to split the dataset into training and validation sets. Here we divided the dataset into 80 % of training and 20 % of validation sets respectively. Now all we have to do is to develop a model framework and fed this data into it.

4.1.2 Defining Model

We have used CNN and LSTM model both, so that we can do a comparative analysis later. Detailed descriptions on CNN and LSTM are discussed later part of the thesis. The framework that we have implemented here, are standard CNN and LSTM model.

4.1.3 Making Predictions

By giving the input text into above function, we got the confidence values of each intent classes associated with each of them. The highest confidence value associated intent class is the resultant intent class of that input sentence.

4.2 CNN based Intent Classification

Deep neural networks (DNN) have revolutionized the field of natural language processing (NLP). Convolutional neural network (CNN) and Recurrent neural network (RNN) are the two main types of deep neural architectures that are widely explored to handle various NLP tasks. CNN is supposed to be good at extracting position-invariant features and RNN at modeling units in sequence. The state of the art on many NLP tasks often switches due to the battle between CNNs and RNNs. This work is the first systematic comparison of CNN and RNN on a wide range of representative NLP tasks, aiming to give basic guidance for the selection of a network.

The idea of using a CNN to classify text was first presented in the paper “Convolutional Neural Networks for Sentence Classification” by Yoon Kim, 2014. [3]

Representation :

One of the central intuitions about this idea is to **see** our documents as images. But let’s discuss it in a detailed manner. Let us say we have a sentence and we have $\text{max-length} = 70$ and $\text{embedding size} = 300$. We can prepare a matrix of numbers with the shape 70×300 to represent this sentence. For images, we also have a matrix where individual elements are pixel values. Instead of image pixels, the input to the tasks is sentences or documents represented as a matrix. Each row of the matrix corresponds to one-word vector.

Convolution Idea: While for an image we move our convolution filter horizontally as well as vertically, for text we fix kernel size to $[\text{filter size} \times \text{embed size}]$, i.e. $(3, 300)$ we are just going to move vertically down for the convolution taking look at three words at once since our filter size is 3 in this case. This idea seems right since our convolution filter is not splitting word embedding. It gets to look at the full embedding of each word. Also, one can think of filter sizes as unigrams, bigrams, trigrams, etc. Since we are looking at a context window of 1, 2, 3, and 5 words respectively.

A Convolutional Neural Network applied to NLP task look like the network depicted in Figure below.

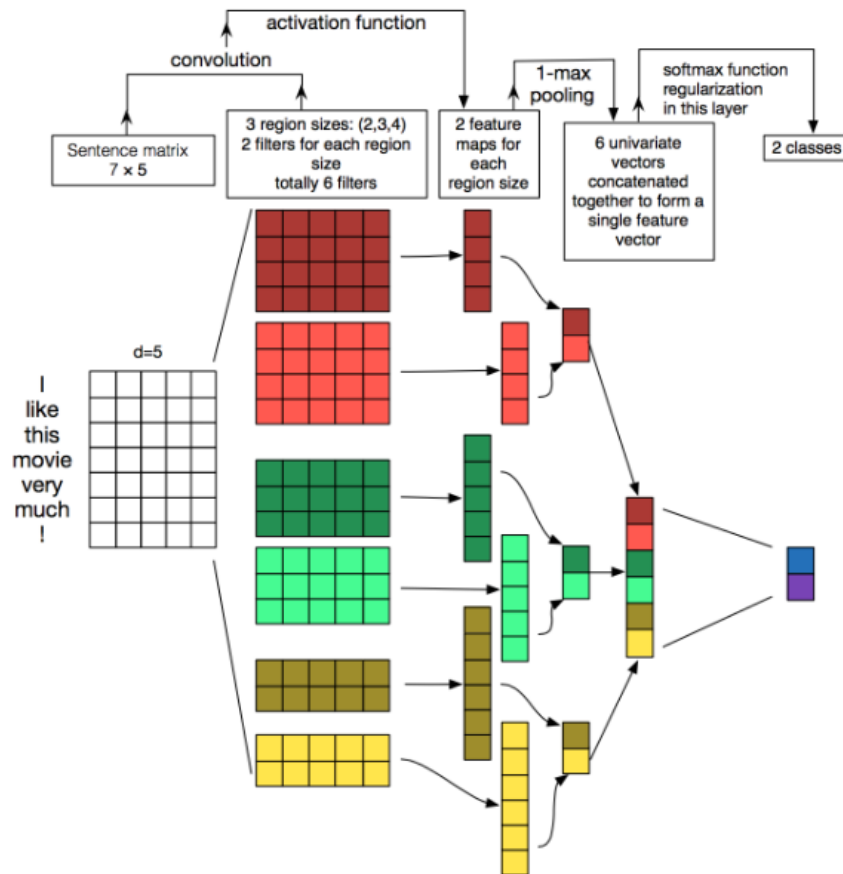


Figure: Illustration of a Convolutional Neural Network (CNN) architecture for sentence classification. Here we depict three filter region sizes: 2, 3 and 4, each of which has 2 filters. Every filter performs convolution on the sentence matrix and generates (variable-length) feature maps. Then 1-max pooling is performed over each map, i.e., the largest number from each feature map is recorded. Thus a univariate feature vector is generated from all six maps, and these 6 features are concatenated to form a feature vector for the penultimate layer. The final softmax layer then receives this feature vector as input and uses it to classify the sentence; here we assume binary classification and hence depict two possible output states. Source: Zhang, Y., & Wallace, B. (2015). A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification.

In most NLP tasks, deep learning models operate on word embedding, which are continuous representations of the input. Convolutions of kernel size n on word embedding will learn to emphasize or disregard n -grams in the input. For example, in sentence classification, a kernel of size 2 learns which bigrams (2-gram) are important for the ultimate classification decision. This concept is similar to attention mechanisms that are commonly used in encoder-decoder architectures.

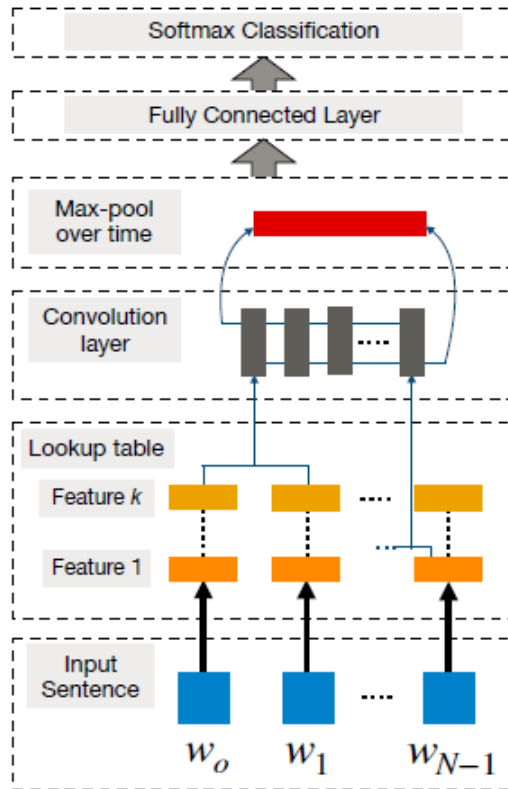


Figure: Collobert and Weston were among the first researchers to apply CNN-based frameworks to NLP tasks[5]. The goal of their method was to transform words into a vector representation via a look-up table, which resulted in a primitive word embedding approach that learn weights during the training of the network.

In order to perform sentence modeling with a basic CNN, sentences are first tokenized into words, which are further transformed into a word embedding matrix (i.e., input embedding layer) of d dimension. Then, convolutional filters are applied on this input embedding layer that consists of applying a filter of all possible window sizes to produce what's called a *feature map*. This is then followed by a *max-pooling* operation that is applied on a max operation on each filter to obtain a fixed length output and reduce the dimensionality of the output. It is observed that the procedure produces the final sentence representation.

By increasing the complexity of the aforementioned basic CNN and adapting it to perform word-based predictions, other NLP tasks such as NER, aspect detection, and POS tagging. This requires a window-based approach, where for each word a fixed size window of neighboring words (sub-sentence) is considered. Then a standalone CNN is applied to the sub-sentence and the training objective is to predict the word in the center of the window, also referred to as word-level classification.

It is noticed that, CNNs were also used for more complex tasks where varying lengths of texts are used such as aspect detection, sentiment analysis, short text categorization,

and sarcasm detection. However, some of these studies reported that external knowledge was necessary when applying CNN-based methods to microtexts such as Twitter texts. Other tasks where CNN proved useful are query-document matching, speech recognition, machine translation (to some degree), and question-answer representations, among others. On the other hand, a DCNN was used to hierarchically learn to capture and compose low-level lexical features into high-level semantic concepts for the automatic summarization of texts.

Overall, CNNs are effective because they can mine semantic clues in contextual windows, but they struggle to preserve sequential order and model long-distance contextual information. Recurrent models are better suited for such type of learning and they are discussed in next section.

Now we have discussed all the underlying layers of CNN one by one.

Pooling Layer

The second important building block of CNNs is the pooling layer. This layer is used to make the outputs less sensitive to the local variation in the inputs. This invariance to small local translation and can decrease the spatial resolution and lead to underfitting in some applications. It is observed that when accurate spatial features are not required, pooling can improve the performance of CNNs in extracting the features of interest. Further, pooling can reduce over-fitting since it decreases the number of dimensions and parameters. In a sense, pooling takes subsamples from the outputs. Similar to convolutional layers, pooling layers use a kernel (a rectangular receptive field) to apply an aggregation function such as maximum, average, L2-norm, or weighted average to summarize the values of the neurons within the pooling kernel. To have a pooling layer in CNNs, we need to determine the size of pooling kernels, the step of shifting, and the number of padding. Below figure depicts max pooling over a 4×4 matrix where the size of pooling kernel is 2×2 and the kernel shifts two pixels over the matrix (i.e., stride of 2).

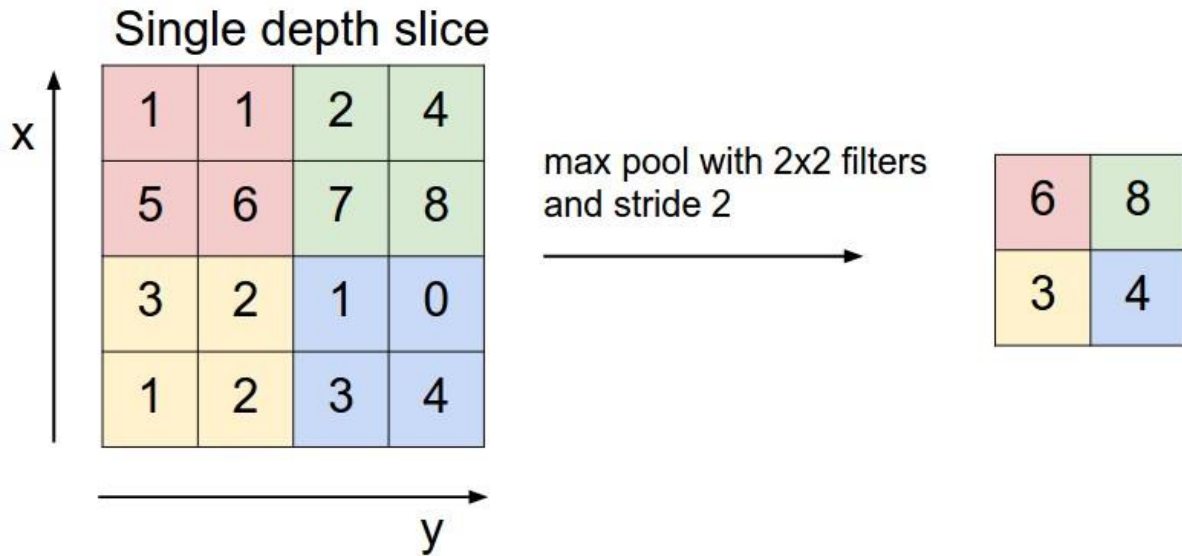


Figure: Pooling of a 4×4 image using a 2×2 kernel with a stride of 2

Fully Connected Layer

A typical CNN consists of several convolutional layers where each convolutional layer is followed by a pooling layer. The last building block of CNNs is the fully connected layer, which is basically a traditional MLP. This component is used to either make a more abstract representation of the inputs by further processing of the features or classify the inputs based on the features extracted by preceding layers.

Activation Functions

Activation functions are important for an Artificial Neural Network to learn and understand the complex patterns. The main function of it is to introduce non-linear properties into the network. What it does is, it calculates the ‘weighted sum’ and adds direction and decides whether to ‘fire’ a particular neuron or not. There are several kinds of non-linear activation functions, like *Sigmoid*, *Tanh*, *ReLU* and *leaky ReLU*. The non linear activation function will help the model to understand the complexity and give accurate results.

Main Architecture of the CNN framework used here :

As discussed earlier, we have applied Convolutional Neural Networks algorithm on text data and as the data is of one-dimensional we cannot use the conventional CNN architecture used for image data. As a result we used 1-D convolutional layers instead of the most popular 2-D convolutional layers. All other layers like max-pooling and dense layers are used as it is. The convolutional neural network (CNN) architecture that has been implemented in the current study constitutes two convolutional layers and two fully connected layer, also known as dense layers. Among the two dense layers, the first one has 128 hidden neurons and the second one has 64 hidden neurons. For the current study, we tried to classify each text input to a particular intent class among 4 intent classes. Thus we have 4 output classes. As a result we added 4 *softmax* units in our last (output) layer to estimate the probability distribution of the classes. In our architecture, every convolutional layer is followed by a max-pooling layer. Each of the first and second convolutional layer is followed by a 1-D max-pooling layer of max-pooling window size of 5 and 3 respectively. The number of kernels (filters) is set to 256 and 512 for the first and second convolutional layer respectively. The sizes of the kernels that have been applied to the first and second convolutional layers are 5 and 3 respectively. Batch size of 16 is applied throughout the training process. Rectified Linear Units (ReLU) were used in convolutional layers and fully connected layers, except in the last dense layer, as activation functions to introduce non-linearity to the model. As the problem is a classification problem, we have used the Categorical Cross-entropy loss. Adam optimizer is also employed to minimize the loss function across the training data. The number of epochs is set to 100. The training procedure for this study was performed entirely on a CPU-based system, no GPU has been used for conducting any part of the training process.

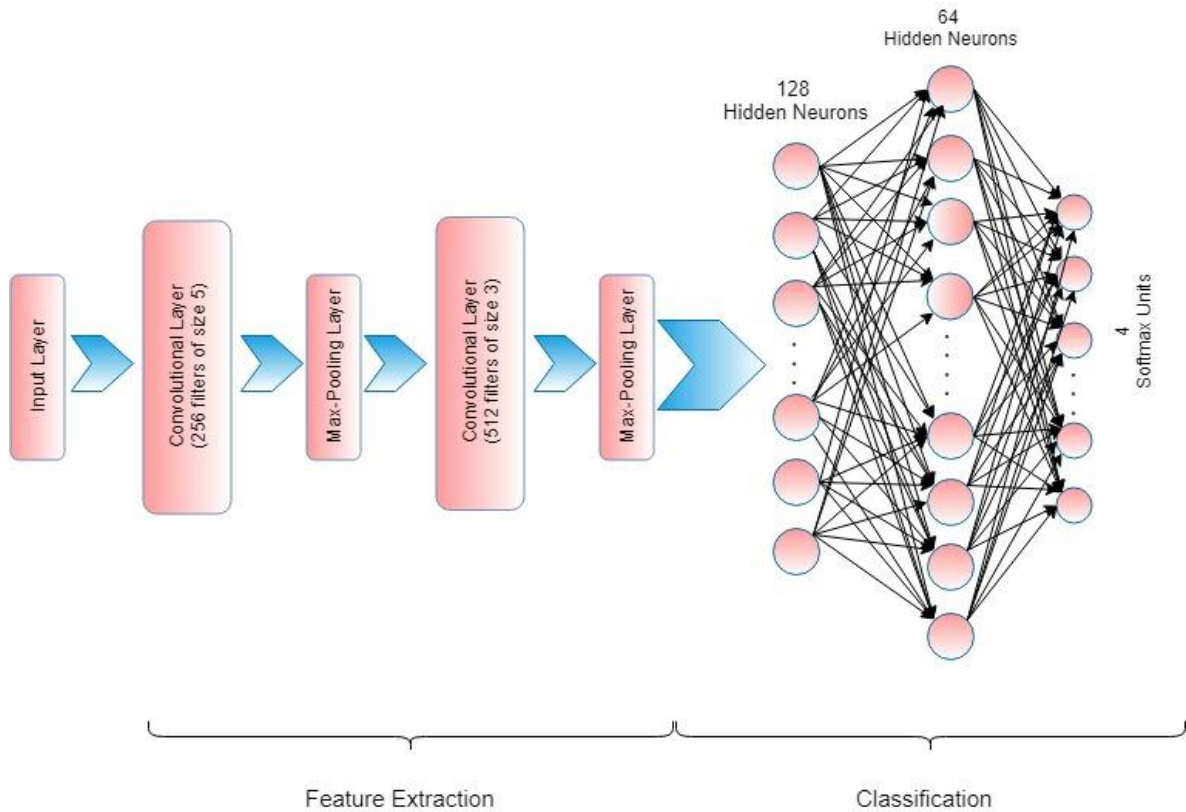
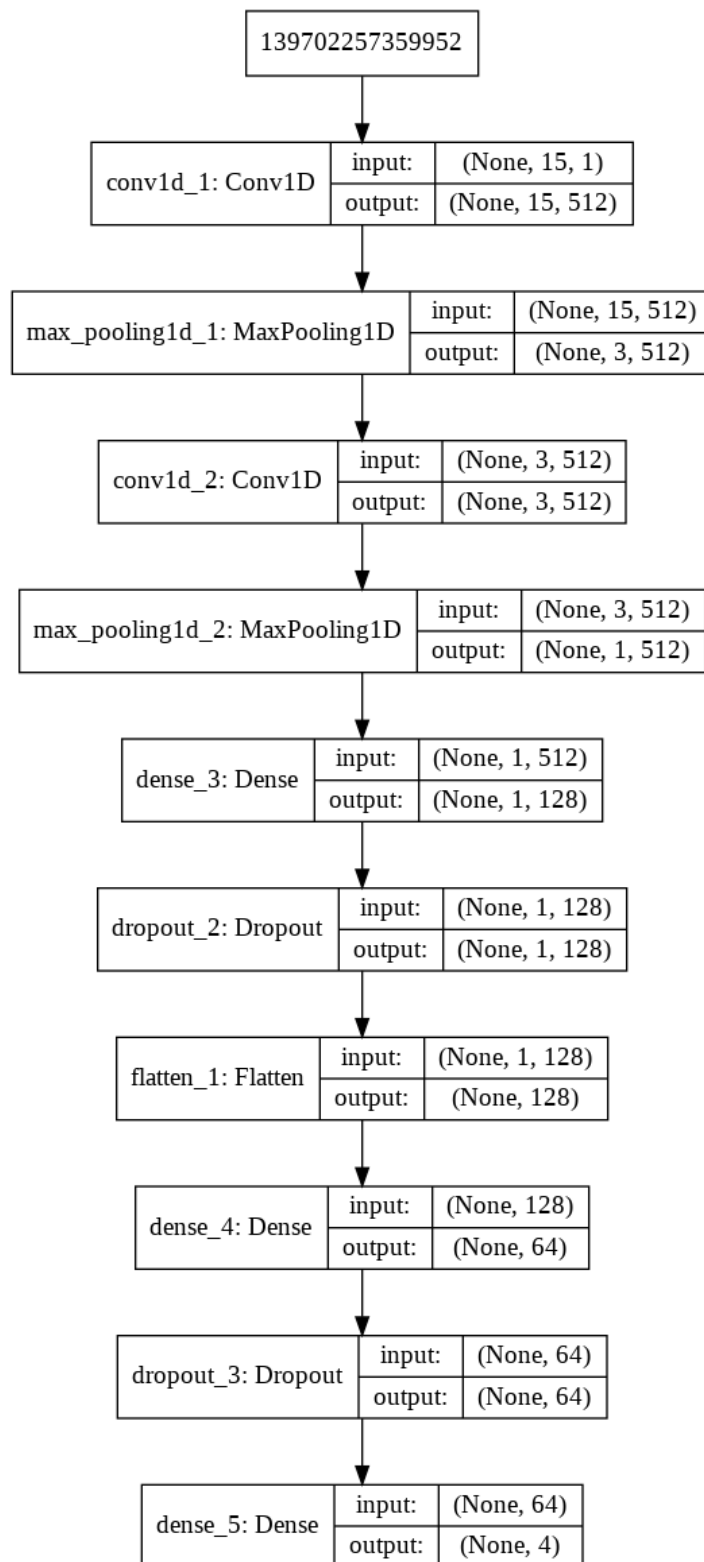


Figure: The baseline architecture of the CNN used in the current study to classify speech utterances based on their emotional states.

We have also used Dropout and Flatten function. Flatten function is used whenever we needed to reduce the dimension of the data which was output by a layer in the network and Dropout layer is used to reduce over-fitting during the training process. Dropout layers reduces over-fitting by dropping out or ignoring some of the neurons. We have used two Dropout layers in our network architecture with each of them residing right after each of the two Dense layers. A dropout rate of 20% has been used in both of the two cases.

Our model of the CNN can be summarized as follows:



Experiments & Results(CNN) :

First let us discuss the frameworks that we have used as CNN based in a more detailed manner. The below figure talks about it:

Layer (type)	Output Shape	Param #
conv1d_5 (Conv1D)	(None, 28, 256)	1536
max_pooling1d_5 (MaxPooling1D)	(None, 5, 256)	0
conv1d_6 (Conv1D)	(None, 5, 512)	393728
max_pooling1d_6 (MaxPooling1D)	(None, 1, 512)	0
dense_13 (Dense)	(None, 1, 128)	65664
dropout_8 (Dropout)	(None, 1, 128)	0
flatten_3 (Flatten)	(None, 128)	0
dense_14 (Dense)	(None, 64)	8256
dropout_9 (Dropout)	(None, 64)	0
dense_15 (Dense)	(None, 11)	715
Total params: 469,899		
Trainable params: 469,899		
Non-trainable params: 0		

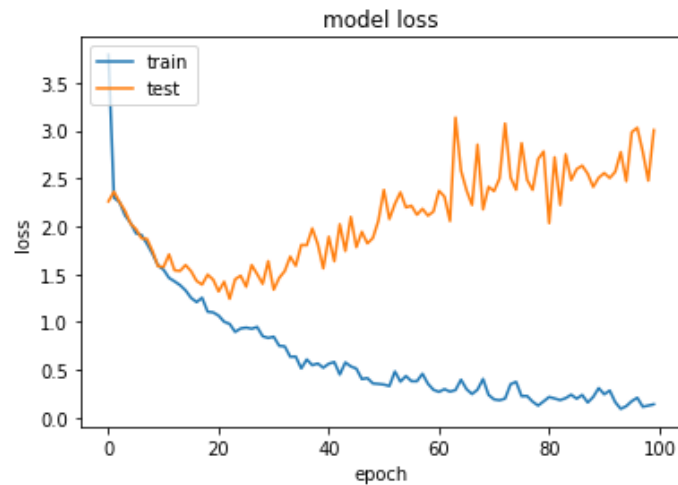
In this section we report the performance of the CNN model we implemented. We have Training vs. Test Accuracy graph, Training vs. Test Loss graph.

For the loss plot we have used categorical cross-entropy formula. For a binary classification problem, the formula can be written as:

$$\mathcal{L}(\theta) = -\frac{1}{n} \sum_{i=1}^n [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)] = -\frac{1}{n} \sum_{i=1}^n \sum_{j=1}^m y_{ij} \log(p_{ij})$$

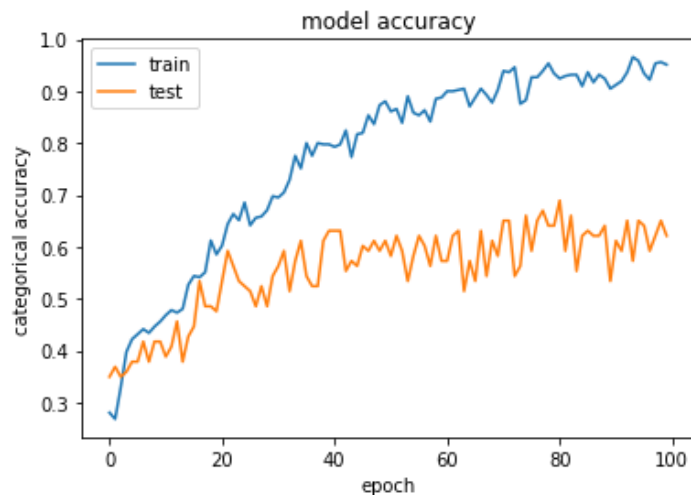
- $y[i]$ are the true labels (0 or 1)
- $p[i]$ are the predictions (real numbers in $[0,1]$), usually interpreted as probabilities
- $\text{output}[i]$ (not shown in the equation) is the *rounding* of $p[i]$, in order to convert them also to 0 or 1; it is this quantity that enters the calculation of accuracy, implicitly involving a threshold (normally at 0.5 for binary classification), so that if $p[i] > 0.5$, then $\text{output}[i] = 1$, otherwise if $p[i] \leq 0.5$, $\text{output}[i] = 0$.

Training vs. Test Loss graph:



So, we can clearly see that, validation/test set loss is more compare to the training set loss.

Training vs. Test Accuracy graph:



Training accuracy is more than validation accuracy, but the gap between them is not big enough to conclude this as overfitting.

Here is the sample input text that we have provided our CNN based intent classifier as shown below:

prediction after training on CNN

```
▶ text = "Give me live status of train number 12345"  
pred = predictions(text, model_cnn, "cnn")  
get_final_output(pred, unique_intent)
```

The confidence score of each intent classification class is shown as below:

```

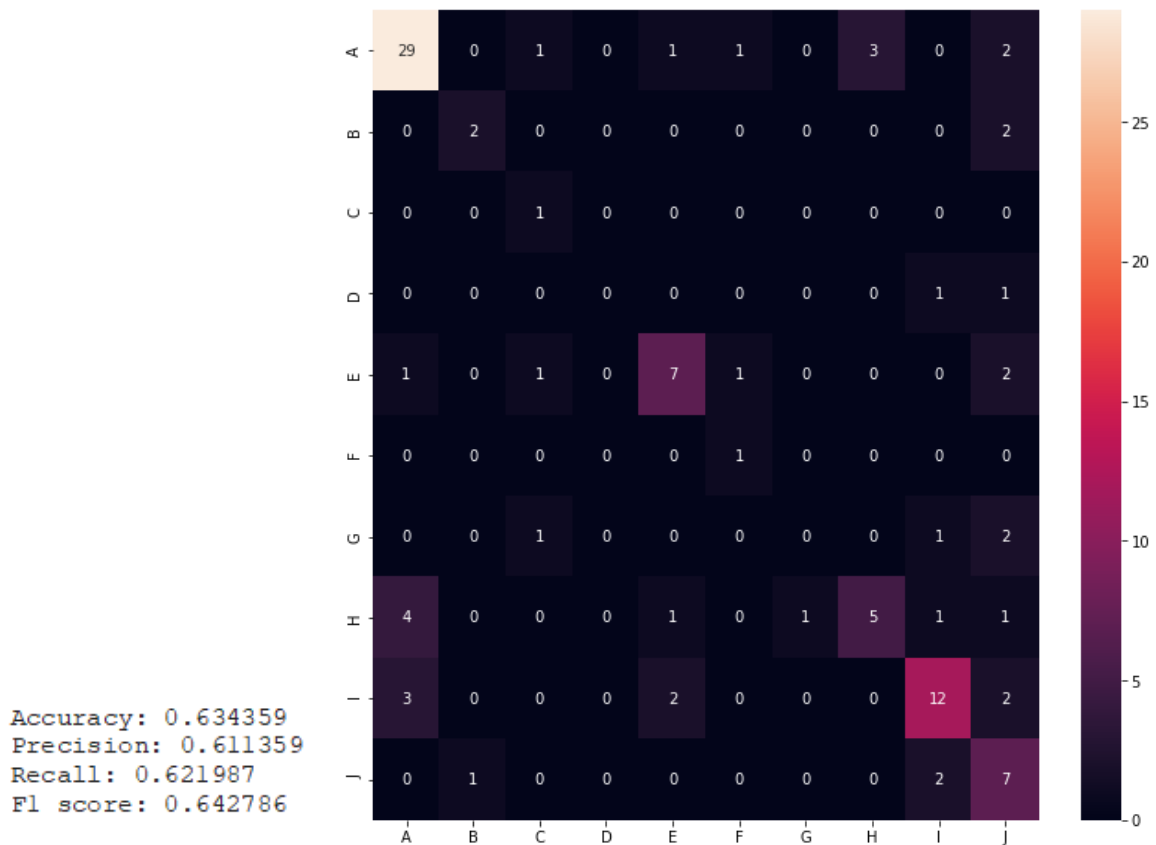
['give', 'me', 'live', 'status', 'of', 'train', 'number',
'12345']
Prediction by CNN
Live_Train_StatusByNumber has confidence = 0.19253011
Default Fallback Intent has confidence = 0.114902504
PNR_Status has confidence = 0.098868944
Seat_availability has confidence = 0.09356083
Train_Fare_Enquiry has confidence = 0.09186842
Train_Info_ByName has confidence = 0.09156009
Train_Info_ByNumber has confidence = 0.08778542
Train_Route confidence = 0.081357285
Train_Between_Stations has confidence = 0.052512027
Default_Welcome_Intent has confidence = 0.015155939

```

Figure : “Live_Train_statusByNumber” intent has the highest confidence value,So it is the predicted intent class of the user input.

The below figures depicts more about our CNN based classification model :

We have calculated accuracy, precision, recall, f-score and cohen’s kappa value for the CNN based classifier.



Confusion Matrix

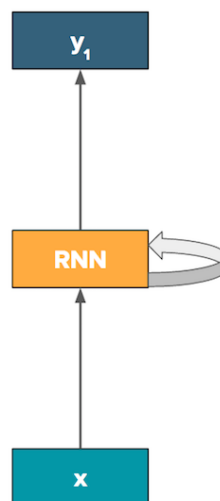
Now we will discuss the LSTM framework, so that we can compare the two frameworks later and be sure whose usability is better in terms of intent classification model.

4.3 LSTM based Intent Classification

Recurrent Neural Networks

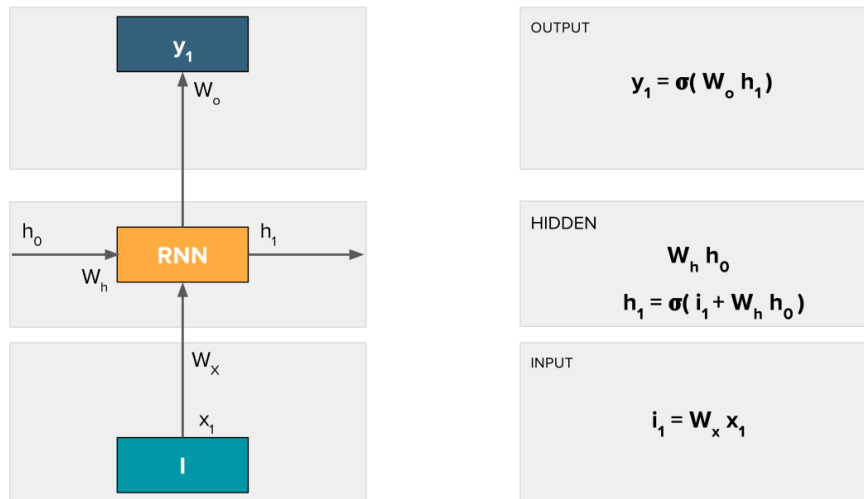
There are two parts to understanding this diagram:

- (a) Understanding what happens at each step i.e., how the current word is processed
- (b) Understanding how that gets together with preceding words



UNDERSTANDING WHAT HAPPENS AT EACH STEP

h_t is called as the *hidden state* at time t , it can be thought of as a way by which the RNN represents phrases it has seen up to time t internally. At time $t=1$, i.e. when we encounter the first word x_0 , the *initial hidden state* h_0 can be thought of as a zero-vector representing no prior knowledge.



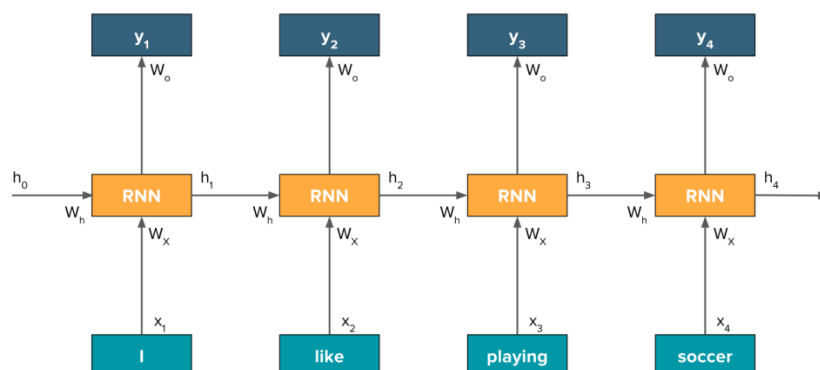
There are three stages at each time step t :

- **Input:** takes in the word embedding of the current word and creates another representation for it using the input weight w_x
- **Hidden:** gets the representation from the previous words with the current one using w_h . h_{t-1} represents the hidden state from all previous words
- **Output:** converts the internal hidden representation to the actual desired output using w_o .

Here, all the weights(w) are matrices, hidden states(h) is a vector and in practice we have a bias(b) vector at each stage which is added (not shown in the figure). “ σ ” here represents the sigmoid function that results in any number between 0 and 1 given an input.

UNDERSTANDING HOW THAT GELS TOGETHER WITH PRECEDING WORDS

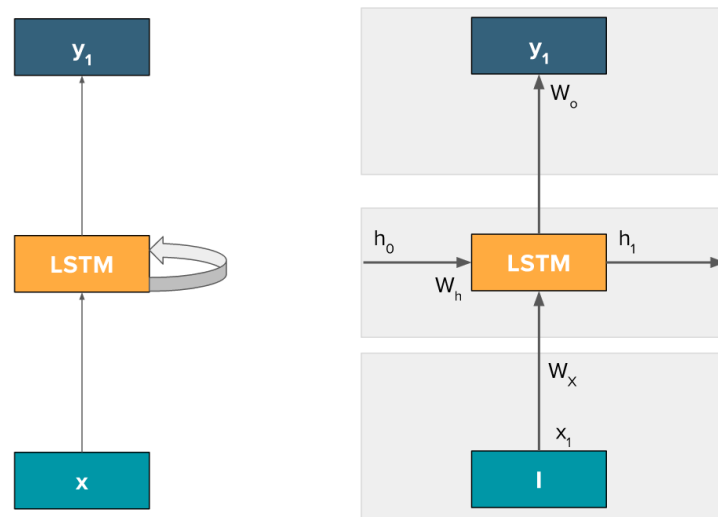
Unrolling the loop in the diagram above on a phrase looks as the figure below:



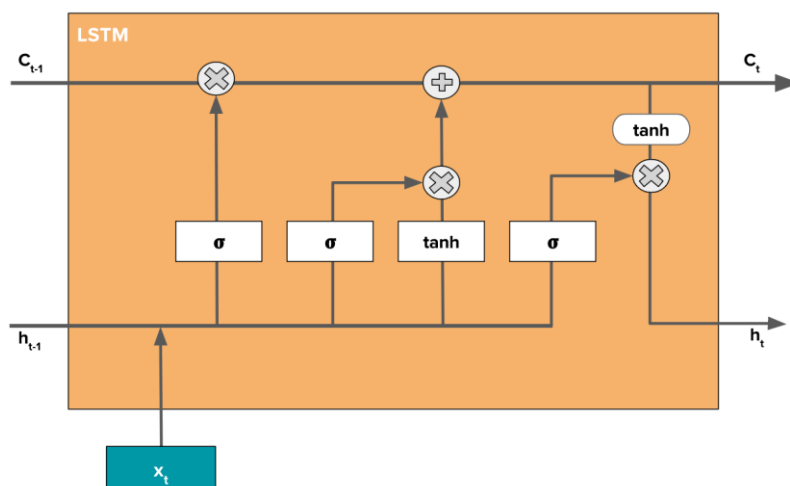
The idea is to from left to right and at each word we create an understanding of what we have read until now. RNNs are great but there is a problem, since at each step we create an understanding of what we read till now, so it's always a mix of the past and the present. Due to this reason sometimes, we may have to process a long piece of text but the only vital information that we may need was the first word itself. Vanilla RNNs suffer from what can be termed as the long-term dependency problem.

LONG SHORT TERM MEMORY (LSTMS)

The abstract representation is exactly the same as for *RNNs*, the difference lies in that *orange* cell.



Inside the orange cell we have the following layout



The first major difference is that we have a C_t now, which is called the **cell state**. Similar to the RNN structure, we still have h_t which derives its value as a *curated version* of C_t . The *cell state* is a very important component of the LSTM as it acts like a *conveyor belt* through which the information flows within and among cells. The small boxes with the three “ σ ” are known as “**gates**” because “ σ ” results a number between 0 and 1 which in a way represents the degree to which information can flow.

eg:

- Information * 0 = No information flows through
- Information * 1 = All the information passes through
- Information * (0, 1) = Some fraction of the information passed through

The following are some new labels in the figure:

tanh: a function that gives out a number between -1 and 1



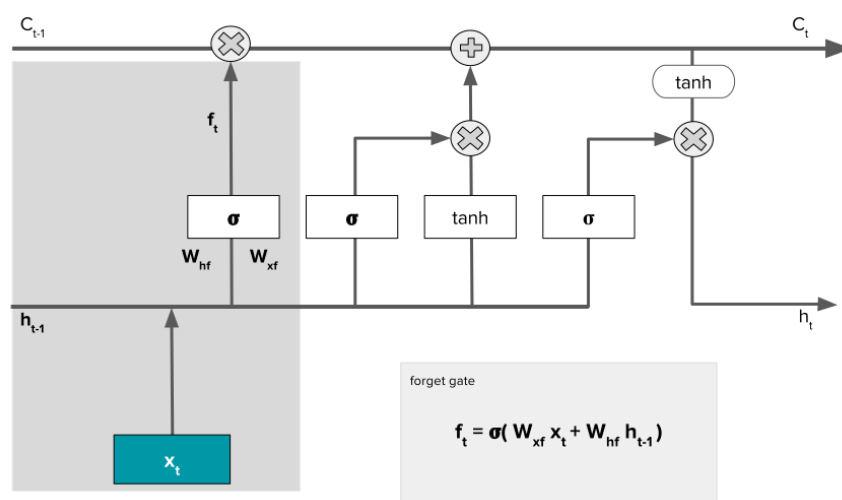
: element wise matrix multiplication



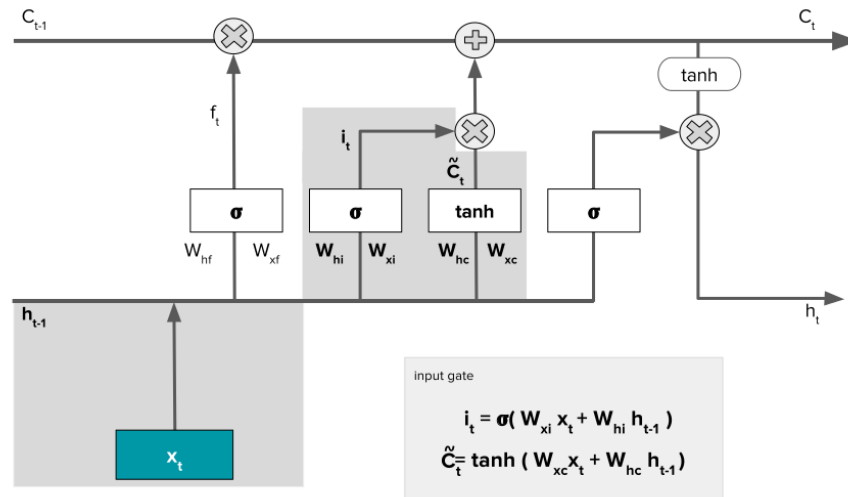
: element wise matrix addition

There are 3 gates and 4 overall steps.

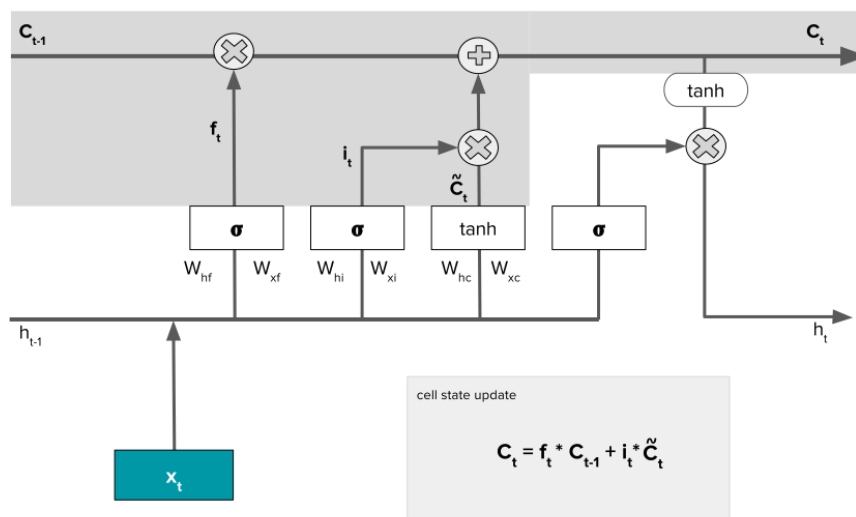
- **Forget Gate (f_t)**: as the name suggests, it decides what information do we want to forget. Takes as input h_{t-1} and the current input x_t and applies a “ σ ” resulting in a vector of numbers between 0 and 1.
-



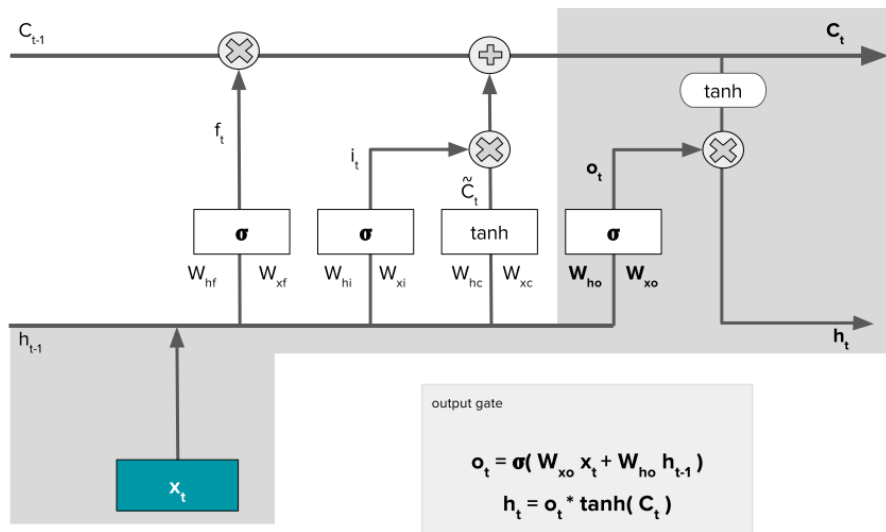
- **Input Gate (i_t):** with the current input, we need to make changes in the cell state to form C_t . For this we may have two outcomes
 - The input gate decides which values to update and by how much using the “ σ ” function*
 - New candidate cell values \hat{C}_t are generated using tanh
 -



Now we know what to forget, what to update and the candidate new values \hat{C}_t . Using this knowledge, we have the next figure

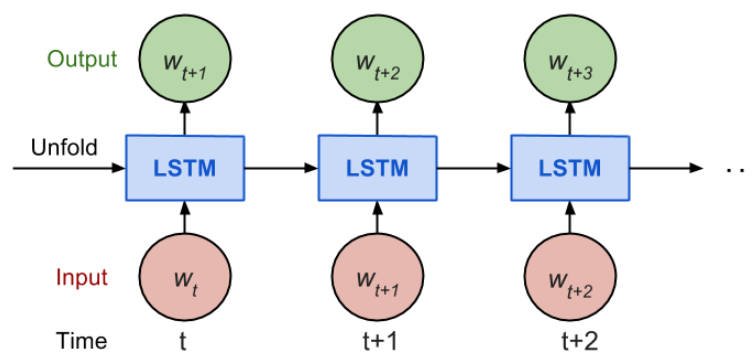


Output Gate (o_t): decides what part of the cell state C_t should we sent as the new hidden state h_t . The “ σ ” takes care of deciding what parts to output and multiplied by C_t gives the new hidden state h_t .



In case of Recurrent Neural Networks, the vital contrasting factor is that unlike normal feed-forward networks for which the activation outputs are propagated only in one direction, here the activation outputs from neurons propagate in both the directions i.e. from inputs to outputs and from outputs to inputs. For this reason, a loop is formed within the neural network that acts as a ‘memory state’ of the neurons thereby providing the neurons an ability to remember what have been learned so far.

The memory state in RNNs provides a plus over ancient neural networks however a dilemma is also included known as the vanishing gradient problem. In this downside, while learning with an oversized variety of layers, it becomes very tedious for the network to learn and tune the parameters of the previous layers. In order to address this problem, a modified version of RNNs termed as LSTMs (Long Short Term Memory) have been developed.



LSTMs have an additional state called ‘cell state’ through which the network makes adjustments in the information flow. The advantage of this state is that the model can remember or forget the leanings more selectively.

1. Input Layer: Takes the sequence of words as input
2. LSTM Layer: Computes the output using LSTM units. Here we have added 100 units in the layer, but this number can be fine-tuned later.
3. Dropout Layer: A regularization layer which randomly turns-off the activations of some neurons in the LSTM layer that helps in preventing over fitting.
4. Output Layer: Computes the probability of the best possible next word as output

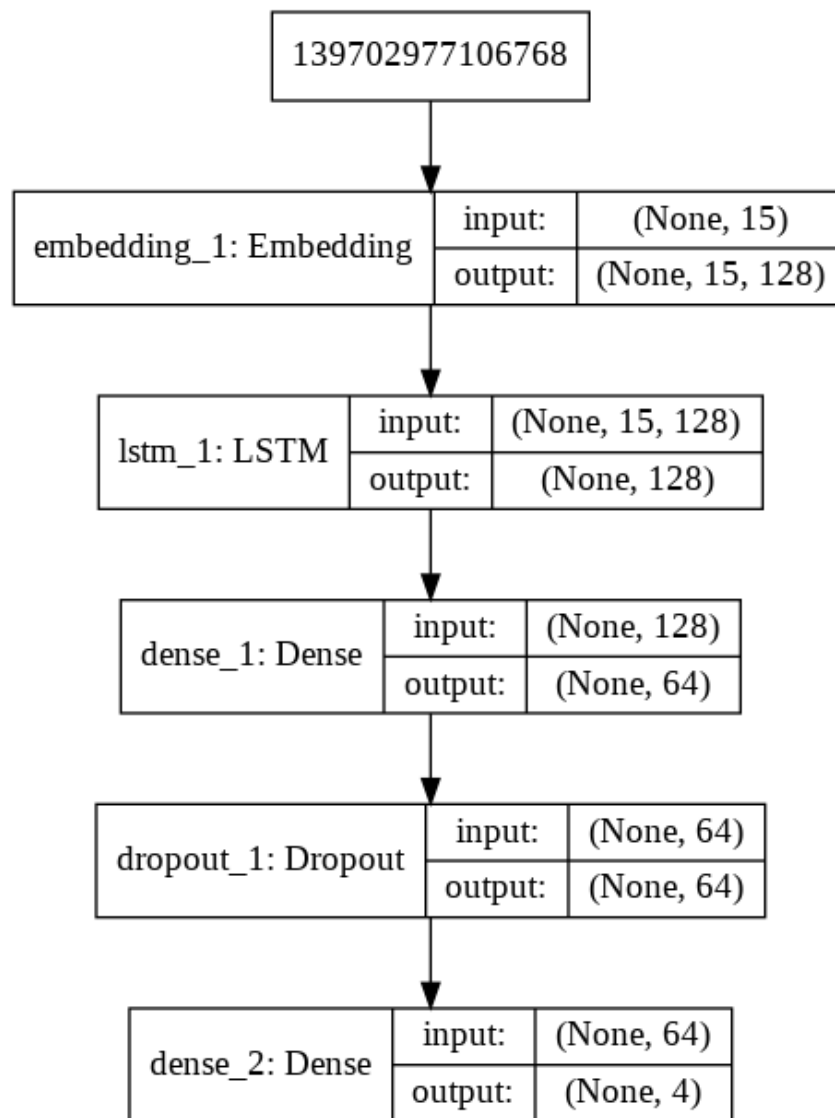
LSTM Hyper parameter tuning

A few ideas should be kept in mind when manually optimizing hyper parameters for RNNs:

- Watch out for over-fitting, which happens when a neural network essentially “memorizes” the training data. Overfitting means you get great performance on training data, but the network’s model is useless for out-of-sample prediction.
- Regularization helps: regularization methods include l_1 , l_2 , and dropout among others.
- So have a separate test set on which the network doesn’t train.
- The larger the network, the more powerful, but it’s also easier to over-fit. Don’t want to try to learn a million parameters from 10,000 examples
- More data is almost always better, because it helps fight over-fitting.
- Train over multiple epochs (complete passes through the dataset).
- Evaluate test set performance at each epoch to know when to stop (early stopping).
- The learning rate is the single most important hyper parameter.
- In general, stacking layers can help.
- For LSTMs, use the softsign (not softmax) activation function over tanh (it’s faster and less prone to saturation (~ 0 gradients)).
- Updaters: RMSProp, AdaGrad or momentum (Nesterovs) are usually good choices. AdaGrad also decays the learning rate, which can help sometimes.

Finally, remember data normalization, MSE loss function + identity activation function for regression, Xavier weight initialization

Now our model of LSTM model looks like this :



Experiments & Results(LSTM):

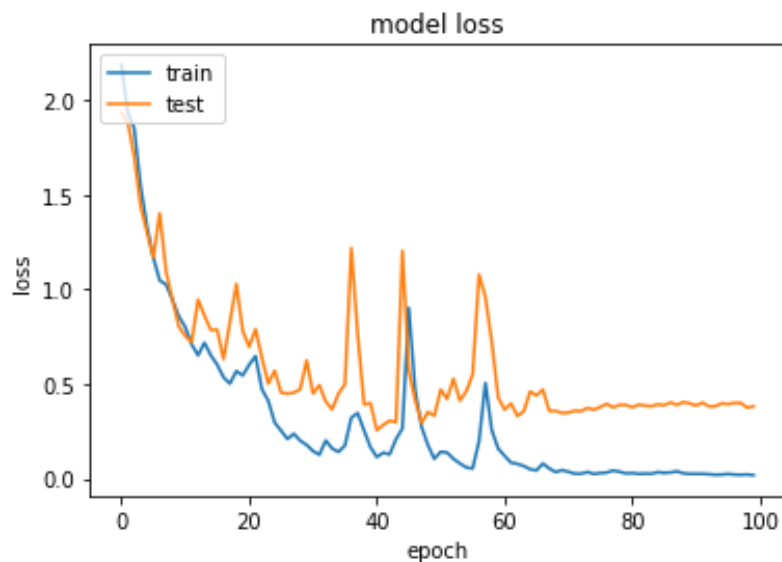
First let us discuss the frameworks that we have used as LSTM based in a more detailed manner. The below figure talks about it:

Layer (type)	Output Shape	Param #
embedding_5 (Embedding)	(None, 28, 128)	38272
lstm_5 (LSTM)	(None, 200)	263200
dense_18 (Dense)	(None, 64)	12864
dropout_11 (Dropout)	(None, 64)	0
dense_19 (Dense)	(None, 11)	715

Total params: 315,051
Trainable params: 315,051
Non-trainable params: 0

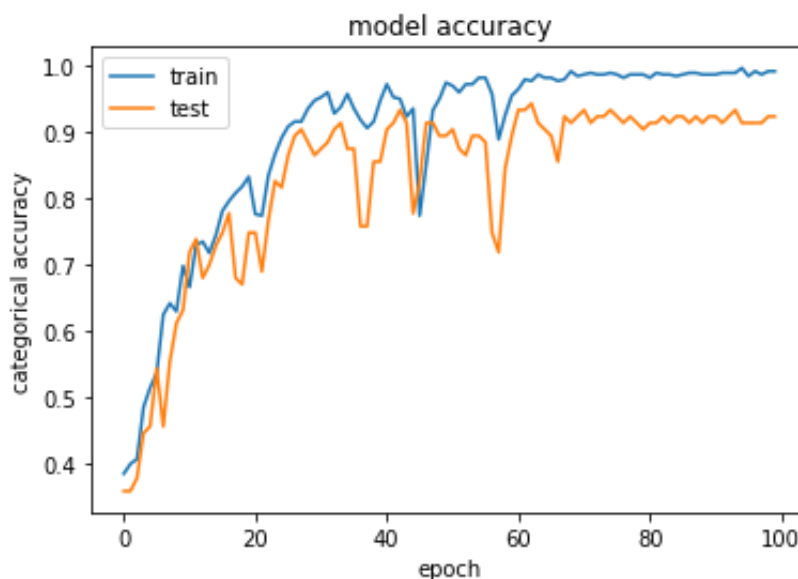
Here also we have used the categorical_crossentropy loss, and we have already discussed how to calculate the loss in CNN (Experiments & Results) section.

Training vs. Test Loss graph:



So, here we can see clearly that, validation/test loss is almost identical to the training set loss which is expected.

Training vs. Test Accuracy graph:



Similarly, validation set accuracy is almost identical to training set and also beneath the training set curve. So we can conclude LSTM based intent classification is a better choice for this model.

Here is the sample input text that we have provided our LSTM based intent classifier as shown below:

prediction after training on LSTM

```
[ ] text = "Give me live status of train number 12345"  
    pred = predictions(text, model_lstm, "lstm")  
    get_final_output(pred, unique_intent)
```

The confidence score of each intent classification class is shown as below:

```
['give', 'me', 'live', 'status', 'of', 'train', 'number',  
'12345']
```

Prediction by LSTM

```
Live_Train_StatusByNumber has confidence = 0.99006534  
Train_Between_Stations has confidence = 0.0055835717  
PNR_Status has confidence = 0.004171253  
Default_Fallback_Intent has confidence = 0.00016836118  
Seat_availability has confidence = 8.072023e-06  
Train_Fare_Enquiry has confidence = 3.436412e-06  
Default_Welcome_Intent has confidence = 4.2496833e-08  
Train_Info_ByName has confidence = 3.7332825e-08  
Train_Route has confidence = 1.6617346e-09  
Train_Info_ByNumber has confidence = 2.337104e-10
```

Here are the result value of accuracy, precision value, recall value, f-score value and cohen's kappa value and the confusion matrix for the LSTM based classifier, shown below in figure 6 and 7 respectively:

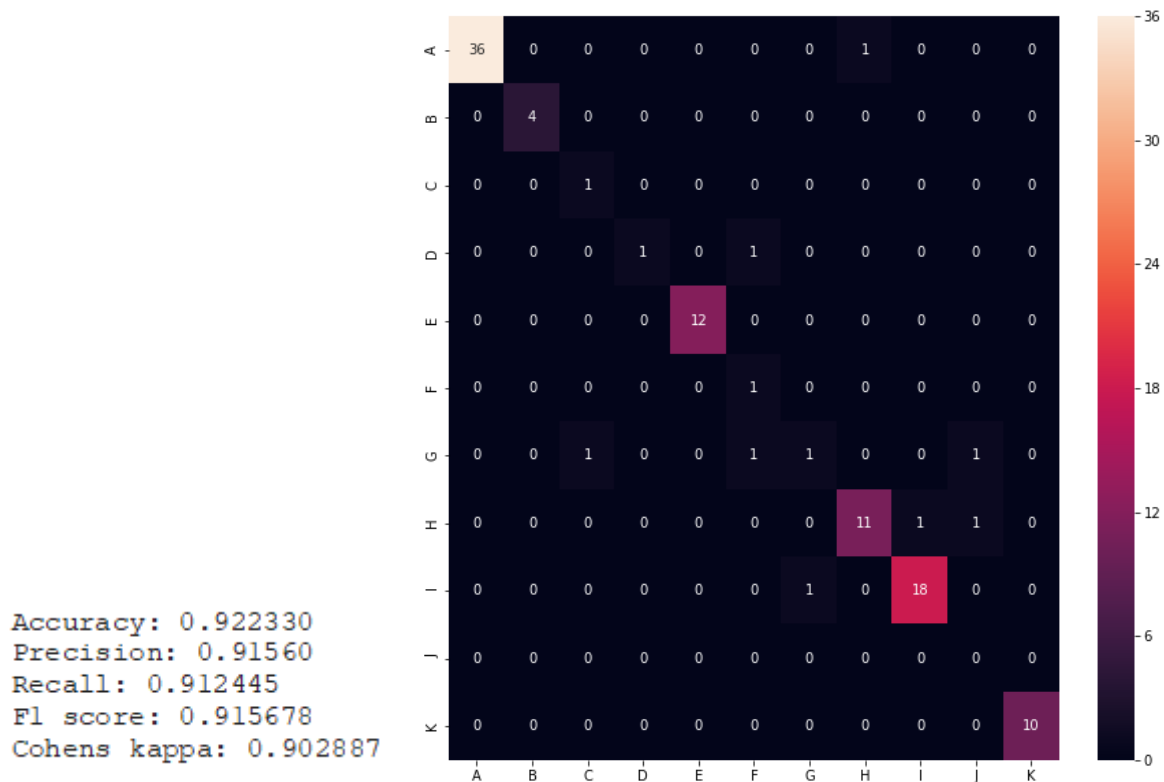


Figure 6

Figure 7

4.4 Comparative Analysis:

So, from comparative analysis of these two frameworks, we can come to the conclusion that, LSTM provides better accuracy in classifying intents. LSTM has been used successfully to over-shadow the effect of noise prevalent in the dataset better compare to CNN.

Chapter 5: Entity Recognition and Response Generation

5.1 Entity Recognition

After classification of intents, we need proper entity values or parameter values, which are useful in generating quality responses to the User. NER helps us in this section. Named-entity recognition (NER) (also known as entity identification, entity chunking and entity extraction) is a subtask of information extraction of NLP, that seeks to locate and classify named entity mentions in unstructured text into pre-defined categories such as the **person names, organizations, locations, medical codes, time expressions, quantities, monetary values, percentages** etc.

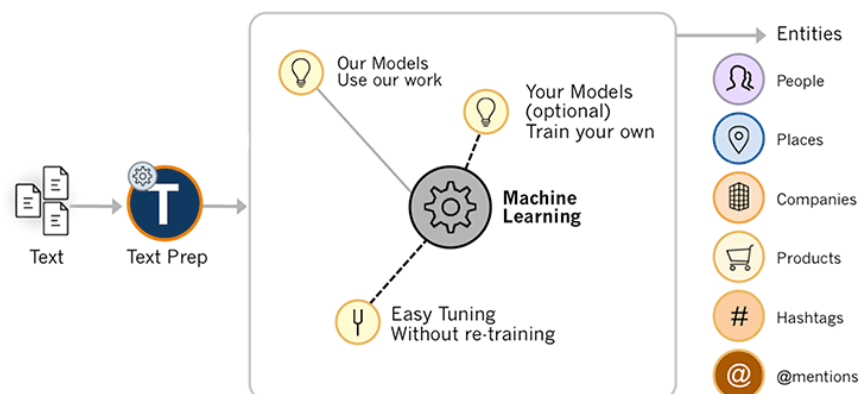


Figure : Here we can see, how an input text can be tagged as various entity set using NER model.

For Example,

Enter A Text

Michael Jordan of the Chicago Bulls is getting a 10-hour Netflix documentary in 2019

Named Entities

Name: Michael Jordan

Group: Netflix

Group: Chicago Bulls

NER helps in:

- to know the relevant tags for each paragraph in an article, thus helping in automatically categorizing the articles in defined hierarchies and enable smooth content discovery.
- to associate relevant tags to every article. Suppose, over an online publishing platform, user can search his/her required article using a fast efficient search algorithm. With this approach, a search term will have to be matched with only the small list of entities discussed in each article leading to faster search execution.
- in providing customer in a fast and efficient manner. For example,



For this above customer query, NER can help customer service provider locating the location as bandra and product as Fitbit.

DEMO- ENTER A TEXT

@cromaretail please train your staff in croma bandra to provide correct details of customer support for Fitbit. The number given doesnt work

KEYWORDS

Group: croma

Group: Fitbit

Place: bandra

Standard Libraries to use Named Entity Recognition

There are three standard libraries which are most popular. They are as follows:

1. Stanford NER[7]
2. spaCy[6]
3. NLTK[2]

In order to accomplish our goals, we have used spaCy.

5.2 spaCy framework and its application

It is an open-source software library for advanced Natural Language Processing, written in the programming languages Python and Cython. The library is published under the MIT license and currently offers statistical neural network models. The spaCy pre-trained model has list of entity classes. Those are as follows :

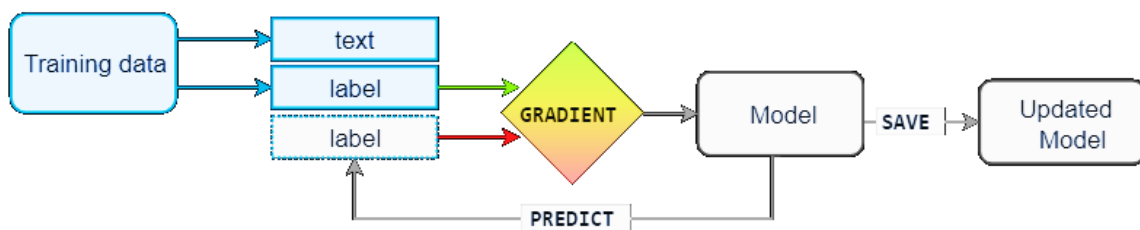
TYPE	DESCRIPTION
PERSON	People, including fictional.
NORP	Nationalities or religious or political groups.
FAC	Buildings, airports, highways, bridges, etc.
ORG	Companies, agencies, institutions, etc.
GPE	Countries, cities, states.
LOC	Non-GPE locations, mountain ranges, bodies of water.
PRODUCT	Objects, vehicles, foods, etc. (Not services.)
EVENT	Named hurricanes, battles, wars, sports events, etc.
WORK_OF_ART	Titles of books, songs, etc.
LAW	Named documents made into laws.
LANGUAGE	Any named language.
DATE	Absolute or relative dates or periods.
TIME	Times smaller than a day.
PERCENT	Percentage, including "%".
MONEY	Monetary values, including unit.
QUANTITY	Measurements, as of weight or distance.
ORDINAL	"first", "second", etc.
CARDINAL	Numerals that do not fall under another type.

spaCy's models are statistical and every "decision" they make – for example, which part-of-speech tag to assign, or whether a word is a named entity, is a prediction. This prediction is based on the examples the model has seen during training. To train a

model, we first need training data i.e the examples of text, and the labels you want the model to predict. This could be a part-of-speech tag, a named entity or any other information.

The model, then shows the unlabeled text and how to make a prediction. Because we know the correct answer, we can give the model feedback on its prediction in the form of an *error gradient* of the *loss function* that calculates the difference between the training example and the expected output. The greater the difference, the more significant the gradient and the updates to our model.

- **Training data:** Examples and their annotations.
- **Text:** The input text the model should predict a label for.
- **Label:** The label the model should predict.
- **Gradient:** Gradient of the loss function calculating the difference between input and expected output.



When training a model, we don't just want it to memorize our examples – we want it to come up with theory that can be generalized across other examples. After all, we don't just want the model to learn that this one instance of “Amazon” right here is a company – we want it to learn that “Amazon”, in contexts *like this*, is most likely a company. That's why the training data should always be representative of the data we want to process. A model trained on Wikipedia, where sentences in the first person are extremely rare, will likely perform badly on Twitter. Similarly, a model trained on romantic novels will likely perform badly on legal text.

This also means that in order to know how the model is performing, and whether it's learning the right things or not, we don't only need training data and we will also need evaluation data. If we only test the model with the data it was trained on, we will have no idea how well it's generalizing. If we want to train a model from scratch, we usually need at least a few hundred examples for both training and evaluation. To update an existing model, we can already achieve decent results with very few examples – as long as they are representative.

TOKENIZATION METRICS

We have to note that if the development data has raw text, some of the gold-standard entities might not align to the predicted tokenization. These tokenization errors are excluded from the NER evaluation. If the tokenization makes it impossible for the model to predict 50% of entities, F-score of the corresponding NER might still look good.

NAME	DESCRIPTION
Dep Loss	Training loss for dependency parser. Should decrease, but usually not to 0.
NER Loss	Training loss for named entity recognizer. Should decrease, but usually not to 0.
UAS	Unlabeled attachment score for parser. The percentage of unlabeled correct arcs. Should increase.
NER P.	NER precision on development data. Should increase.
NER R.	NER recall on development data. Should increase.
NER F.	NER F-score on development data. Should increase.
Tag %	Fine-grained part-of-speech tag accuracy on development data. Should increase.
Token %	Tokenization accuracy on development data.
CPU WPS	Prediction speed on CPU in words per second, if available. Should stay stable.
GPU WPS	Prediction speed on GPU in words per second, if available. Should stay stable.

Improving accuracy with transfer learning V2.1

In most projects, we usually have a small amount of labeled data, and access to a much bigger sample of raw text. The raw text contains a lot of information about the language in general. Learning this general information from the raw text can help the model to make use the smaller labeled data more efficiently.

There are two main ways to use raw text in spaCy models, i) **word vectors** and ii) **language model pre-training**. Word vectors provide information about the definitions of words. The vectors are a look-up table, so each word only has one representation, regardless of its context. On the other hand, Language model pretraining lets us learn contextualized word representations. Instead of initializing spaCy's convolutional neural network layers with random weights, the `spacy pretrain` command trains a language model to predict each word's word vector based on the surrounding words. The information used to predict this task is a good starting point for other tasks such as named entity recognition, text classification or dependency parsing.

5.3 Experiments and Results on Entity Recognition on Test Data

Now, we have modified standard spaCy NER in the following ways. To discuss it first, first we had to check if the standard NER of spaCy model is working fine with recognizing all the entities in our user input or not. What we came to realize is as below:

Testing on trained standard model:

```

Entities [(['howrah', 'LOC')]
Tokens [(['On', '', 2), ('thursday', '', 2), (',', '', 2), ('which', '', 2), ('train', '', 2), ('runs', '', 2), ('between', '', 2), ('howrah', 'LOC', 3), ('and', '', 2), ('jammu', '', 2), ('-', '', 2),
Entities [(['23456', 'PNR')]
Tokens [(['Provide', '', 2), ('me', '', 2), ('live', '', 2), ('status', '', 2), ('of', '', 2), ('train', '', 2), ('number', '', 2), ('23456', 'PNR', 3)]
Entities [(['34567', 'PNR')]
Tokens [(['Can', '', 2), ('you', '', 2), ('please', '', 2), ('provide', '', 2), ('the', '', 2), ('live', '', 2), ('train', '', 2), ('location', '', 2), ('of', '', 2), ('train', '', 2), ('whose', '', 2),
Entities [(['howrah', 'LOC'), ('bandel', 'LOC')]
Tokens [(['Give', '', 2), ('the', '', 2), ('train', '', 2), ('names', '', 2), ('that', '', 2), ('run', '', 2), ('from', '', 2), ('howrah', 'LOC', 3), ('to', '', 2), ('bandel', 'LOC', 3), ('on', '', 2),

```

Please help me getting the trains that run between kolkata and durgapur on thursday.

INTJ VERB PRON VERB DET NOUN ADJ VERB ADP NOUN CCONJ ADJ ADP NOUN

Please help me getting the trains that run between kolkata and durgapur on **thursday DATE** .

We can see clearly that, “Kolkata” and “Durgapur” have not been selected as location fields. So, we need to train on standard spaCy NER.

Training set up with examples

Then we trained our model using this:

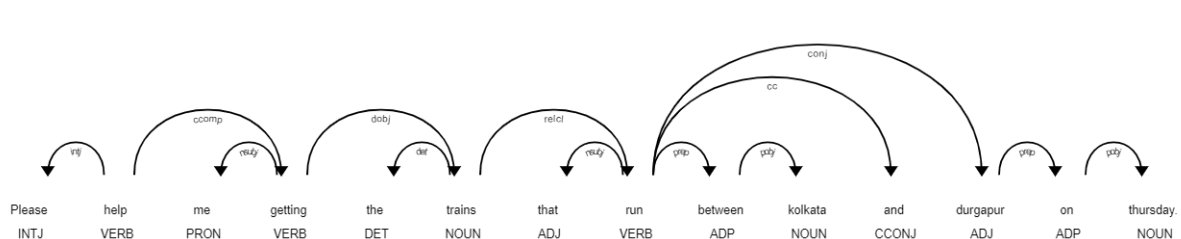
Entity Recognition

preparing training data

```
[1] TRAIN_DATA = [  
    ("Can you please provide the live train location of train whose number is 34567?", {"entities": [(72, 77, "PNR")]}),  
    ("Provide me live status of train number 23456", {"entities": [(39, 44, "PNR")]}),  
    ("Give the train names that run from howrah to bandel on saturday?", {"entities": [(35, 41, "LOC"), (45, 51, "LOC")]}),  
    ("On thursday, which train runs between howrah and jammu-kashmir ?", {"entities": [(38, 44, "LOC"), (49, 61, "LOC")]})  
]
```

Based on the above, we can create four training sentences with three entities in total. What we consider a “correct annotation” will always depend on **what we want the model to learn**. While there are some entity annotations that are more or less universally correct – like “Canada” being a “geopolitical entity” – our application may have its very own definition of the NER annotation scheme.

Now the result after training with our own set of sentences is as follows:



Training spaCy NER model by following above procedures, we have been able to classify our train information specific entities out of a user query easily like below :

Please help me getting the trains that run between **kolkata LOC** and **durgapur LOC** on thursday.'

Now, we can see clearly that, “Kolkata” and “Durgapur” have been selected as location fields.

Here are the experiments results below :

```
Accuracy: 0.722330  
Precision: 0.71560  
Recall: 0.712445  
F1 score: 0.715678  
Cohen's kappa: 0.702887
```

5.4 Response Generation

After detecting entities, we fed the entity values to an url which is then sent to the API called “Railway API”[8] and get the responses back.

For example, we passed the query to railway API “What are the trains between HWH and BBSR”. It falls under the intent class “Train_between_stations”

```
def four():
    print("Train Between Stations")
    url = "https://api.railwayapi.com/v2/between/source/<stn code>/dest/<stn code>/date/<dd-mm-yyyy>/apikey/<apikey>/"
    return 'Train Between Stations'
```

We get output as :

```
to_station : {'u'lat': 20.1703784, 'u'lng': 85.70586929999999, 'u'code': 'u'KUR', 'u'name': 'u'KHURDA ROAD JN'}
name : BGP-YPR ANGA EXPRESS
dest_arrival_time : 06:35
src_departure_time : 23:30
number : 12254
days : [{'u'runs': 'u'N', 'u'code': 'u'MON'}, {'u'runs': 'u'N', 'u'code': 'u'TUE'}, {'u'runs': 'u'Y', 'u'code': 'u'WED'}, {'u'runs': 'u'N', 'u'code': 'u'THU'}, {'u'runs': 'u'N', 'u'code': 'u'FRI'}, {'u'runs': 'u'N', 'u'code': 'u'SAT'}, {'u'runs': 'u'N', 'u'code': 'u'SUN'}]
travel_time : 07:05
from_station : {'u'lat': 22.5957689, 'u'lng': 88.26363940000002, 'u'code': 'u'HWH', 'u'name': 'u'HOWRAH JN'}
classes : [{'u'code': 'u'1A', 'u'name': 'u'FIRST AC'}, {'u'code': 'u'FC', 'u'name': 'u'FIRST CLASS'}, {'u'code': 'u'3E', 'u'name': 'u'3rd AC ECONOMY'}, {'u'code': 'u'SL', 'u'name': 'u'SLEEPER CLASS'}, {'u'code': 'u'2', 'u'name': 'u'2nd AC ECONOMY'}]

to_station : {'u'lat': 20.1703784, 'u'lng': 85.70586929999999, 'u'code': 'u'KUR', 'u'name': 'u'KHURDA ROAD JN'}
name : SC-SHM SF EXP
dest_arrival_time : 00:20
src_departure_time : 08:30
number : 22850
days : [{'u'runs': 'u'N', 'u'code': 'u'MON'}, {'u'runs': 'u'N', 'u'code': 'u'TUE'}, {'u'runs': 'u'N', 'u'code': 'u'WED'}, {'u'runs': 'u'N', 'u'code': 'u'THU'}, {'u'runs': 'u'Y', 'u'code': 'u'FRI'}, {'u'runs': 'u'N', 'u'code': 'u'SAT'}, {'u'runs': 'u'N', 'u'code': 'u'SUN'}]
travel_time : 15:50
from_station : {'u'lat': 22.5839176, 'u'lng': 88.2826103, 'u'code': 'u'SRC', 'u'name': 'u'SANTRAGACHI JN'}
classes : [{'u'code': 'u'1A', 'u'name': 'u'FIRST AC'}, {'u'code': 'u'FC', 'u'name': 'u'FIRST CLASS'}, {'u'code': 'u'3E', 'u'name': 'u'3rd AC ECONOMY'}, {'u'code': 'u'SL', 'u'name': 'u'SLEEPER CLASS'}, {'u'code': 'u'2', 'u'name': 'u'2nd AC ECONOMY'}]

to_station : {'u'lat': 20.1703784, 'u'lng': 85.70586929999999, 'u'code': 'u'KUR', 'u'name': 'u'KHURDA ROAD JN'}
name : HWH-YPR-HWH HUMSAFAR EXP
dest_arrival_time : 20:05
src_departure_time : 12:40
number : 22887
days : [{'u'runs': 'u'N', 'u'code': 'u'MON'}, {'u'runs': 'u'Y', 'u'code': 'u'TUE'}, {'u'runs': 'u'N', 'u'code': 'u'WED'}, {'u'runs': 'u'N', 'u'code': 'u'THU'}, {'u'runs': 'u'N', 'u'code': 'u'FRI'}, {'u'runs': 'u'N', 'u'code': 'u'SAT'}, {'u'runs': 'u'N', 'u'code': 'u'SUN'}]
travel_time : 07:25
from_station : {'u'lat': 22.5957689, 'u'lng': 88.26363940000002, 'u'code': 'u'HWH', 'u'name': 'u'HOWRAH JN'}
classes : [{'u'code': 'u'1A', 'u'name': 'u'FIRST AC'}, {'u'code': 'u'FC', 'u'name': 'u'FIRST CLASS'}, {'u'code': 'u'3E', 'u'name': 'u'3rd AC ECONOMY'}, {'u'code': 'u'SL', 'u'name': 'u'SLEEPER CLASS'}, {'u'code': 'u'2', 'u'name': 'u'2nd AC ECONOMY'}]

to_station : {'u'lat': 20.1703784, 'u'lng': 85.70586929999999, 'u'code': 'u'KUR', 'u'name': 'u'KHURDA ROAD JN'}
name : HWH-SRI SATYA SAI NILAYAM
dest_arrival_time : 22:40
src_departure_time : 15:35
number : 22831
days : [{'u'runs': 'u'N', 'u'code': 'u'MON'}, {'u'runs': 'u'N', 'u'code': 'u'TUE'}, {'u'runs': 'u'N', 'u'code': 'u'WED'}, {'u'runs': 'u'N', 'u'code': 'u'THU'}, {'u'runs': 'u'Y', 'u'code': 'u'FRI'}, {'u'runs': 'u'N', 'u'code': 'u'SAT'}, {'u'runs': 'u'N', 'u'code': 'u'SUN'}]
travel_time : 07:05
from_station : {'u'lat': 22.5957689, 'u'lng': 88.26363940000002, 'u'code': 'u'HWH', 'u'name': 'u'HOWRAH JN'}
classes : [{'u'code': 'u'1A', 'u'name': 'u'FIRST AC'}, {'u'code': 'u'FC', 'u'name': 'u'FIRST CLASS'}, {'u'code': 'u'3E', 'u'name': 'u'3rd AC ECONOMY'}, {'u'code': 'u'SL', 'u'name': 'u'SLEEPER CLASS'}, {'u'code': 'u'2', 'u'name': 'u'2nd AC ECONOMY'}]

to_station : {'u'lat': 20.1703784, 'u'lng': 85.70586929999999, 'u'code': 'u'KUR', 'u'name': 'u'KHURDA ROAD JN'}
name : NCJ-SHM GURUDEV EXPRESS
dest_arrival_time : 05:20
src_departure_time : 13:18
number : 12659
days : [{'u'runs': 'u'N', 'u'code': 'u'MON'}, {'u'runs': 'u'N', 'u'code': 'u'TUE'}, {'u'runs': 'u'N', 'u'code': 'u'WED'}, {'u'runs': 'u'N', 'u'code': 'u'THU'}, {'u'runs': 'u'N', 'u'code': 'u'FRI'}, {'u'runs': 'u'N', 'u'code': 'u'SAT'}, {'u'runs': 'u'N', 'u'code': 'u'SUN'}]
travel_time : 40:02
from_station : {'u'lat': 22.5839176, 'u'lng': 88.2826103, 'u'code': 'u'SRC', 'u'name': 'u'SANTRAGACHI JN'}
classes : [{'u'code': 'u'1A', 'u'name': 'u'FIRST AC'}, {'u'code': 'u'FC', 'u'name': 'u'FIRST CLASS'}, {'u'code': 'u'3E', 'u'name': 'u'3rd AC ECONOMY'}, {'u'code': 'u'SL', 'u'name': 'u'SLEEPER CLASS'}, {'u'code': 'u'2', 'u'name': 'u'2nd AC ECONOMY'}]

to_station : {'u'lat': 20.1703784, 'u'lng': 85.70586929999999, 'u'code': 'u'KUR', 'u'name': 'u'KHURDA ROAD JN'}
name : SRC MAS ANTYODAYA EXPRESS
dest_arrival_time : 01:55
src_departure_time : 19:00
number : 22841
days : [{'u'runs': 'u'Y', 'u'code': 'u'MON'}, {'u'runs': 'u'N', 'u'code': 'u'TUE'}, {'u'runs': 'u'N', 'u'code': 'u'WED'}, {'u'runs': 'u'N', 'u'code': 'u'THU'}, {'u'runs': 'u'N', 'u'code': 'u'FRI'}, {'u'runs': 'u'N', 'u'code': 'u'SAT'}, {'u'runs': 'u'N', 'u'code': 'u'SUN'}]
travel_time : 06:55
from_station : {'u'lat': 22.5839176, 'u'lng': 88.2826103, 'u'code': 'u'SRC', 'u'name': 'u'SANTRAGACHI JN'}
classes : [{'u'code': 'u'1A', 'u'name': 'u'FIRST AC'}, {'u'code': 'u'FC', 'u'name': 'u'FIRST CLASS'}, {'u'code': 'u'3E', 'u'name': 'u'3rd AC ECONOMY'}, {'u'code': 'u'SL', 'u'name': 'u'SLEEPER CLASS'}, {'u'code': 'u'2', 'u'name': 'u'2nd AC ECONOMY'}]

to_station : {'u'lat': 20.1703784, 'u'lng': 85.70586929999999, 'u'code': 'u'KUR', 'u'name': 'u'KHURDA ROAD JN'}
name : HWH-SC FALAKNUMA EXP
dest_arrival_time : 14:40
src_departure_time : 07:25
number : 12703
days : [{'u'runs': 'u'Y', 'u'code': 'u'MON'}, {'u'runs': 'u'Y', 'u'code': 'u'TUE'}, {'u'runs': 'u'Y', 'u'code': 'u'WED'}, {'u'runs': 'u'Y', 'u'code': 'u'THU'}, {'u'runs': 'u'Y', 'u'code': 'u'FRI'}, {'u'runs': 'u'Y', 'u'code': 'u'SAT'}, {'u'runs': 'u'Y', 'u'code': 'u'SUN'}]
travel_time : 07:15
from_station : {'u'lat': 22.5957689, 'u'lng': 88.26363940000002, 'u'code': 'u'HWH', 'u'name': 'u'HOWRAH JN'}
classes : [{'u'code': 'u'1A', 'u'name': 'u'FIRST AC'}, {'u'code': 'u'FC', 'u'name': 'u'FIRST CLASS'}, {'u'code': 'u'3E', 'u'name': 'u'3rd AC ECONOMY'}, {'u'code': 'u'SL', 'u'name': 'u'SLEEPER CLASS'}, {'u'code': 'u'2', 'u'name': 'u'2nd AC ECONOMY'}]
```


For other response generations. url looks like below :

For function : **Station Name to Code**

url = https://api.railwayapi.com/v2/name-to_code/station/<stnname>/apikey/<apikey>/

For function : **Train Route**

url = <https://api.railwayapi.com/v2/route/train/<train number>/apikey/<apikey>/>

For function : **Seat availability**

url = <https://api.railwayapi.com/v2/check-seat/train/<train number>/source/<stn code>/dest/<dest code>/date/<dd-mm-yyyy>/pref/<class code>/quota/<quota code>/apikey/<apikey>/>

For function : **Live Train Status**

url = <https://api.railwayapi.com/v2/live/train/<train-number>/station/<station-code>/date/<dd-mm-yyyy>/apikey/<apikey>/>

For function : **Train Between Stations**

url = <https://api.railwayapi.com/v2/between/source/<stn code>/dest/<stn code>/date/<dd-mm-yyyy>/apikey/<apikey>/>

For function : **Train Fare Enquiry**

url = <https://api.railwayapi.com/v2/fare/train/<train number>/source/<stn code>/dest/<stn code>/age/<age>/pref/<class code>/quota/<quota code>/date/<dd-mm-yyyy>/apikey/<apikey>/>

Chapter 6: Conclusion and Future Work

6.1 Conclusion

The goal of this thesis work was to build a stepping stone of a chatbot that will help user providing railway specific information. For this, the first thing that we did is the dataset collection, but what we came to know after studying all the available datasets is that, no datasets are capable of training our intent classification model. As we all know that, we need labeled data for training an intent classifier and in that labeled data, we need intents associated with user input which are specific to the train information related queries. For that, we have built a chatbot framework that are capable of creating a dataset containing real user data with proper intent classification name associated with it. We are also able to take out the entity values of that user input query depending on that intent name. Using these entity values we can build a dialogue tracker system which will be able to pertain a context during the conversation making our bot more human like. Using the dataset, we have collected so far, we have built two intent classifier using CNN and LSTM framework. With more data, which are to be gathered in coming days, we will be able to make our bot more steady. Correctly classifying intents is the first stepping stone of building a more human like bot. So, we have given utmost importance building the intent classification framework in this project. The demo bot that we have built using messenger, dialogflow and node.js can be used near future gathering data in any time. It will play a vital role helping to build a dialogue management model coming days. In the entity recognition model, we have used spaCy libraries to build a entity recognizer, which is able to identify correct entity (like `source_station`, `destination_station`, `journey_day` etc) from the user query which is very important for generating responses to the user. As for now, we are providing responses based on railway API. To generate a proper response, we needed the proper entity parameter values out of a user query after classifying its intent, for which the entity recognizer model has been built.

6.2 Future Work

The main challenge of chatbot is to pertain the context of a particular intent. As we have already mentioned in the challenges part, after identifying an intent, sometimes, a chatbot answers to that query depending on that intent statistically or API based. It is all about one to one query-answering system. But what happens, sometimes user refers to a entity value(s) of a previous query in the current query. For this type of scenarios, more study is needed, so that we can keep session of intent live until a new intent is classified for a new user query. So, a dialogue system is needed to be developed so that, we can keep track the context of a user query, until the request is being served by the chatbot to the user.

Another challenge, which is very important and most strenuous, is to make the bot platform independent. For example, here, we have developed the intent classification framework based on train related queries. If in future one can make the intent classification model run on all platforms (be it railways or health-care or judiciary system or agricultural helping system etc), being trained on a particular dataset, it will be a great success in chatbot domain.

So, bringing context and remembering the course of the conversation over a time is the most challenging future work in this circumstances. So far, as per our planning goes, if we will be able to associate a session to each intent and if we are able to remember the parameter entity values of that intent over a time, we can use those slot values in later conversation in proper scenario, thus making the bot more human like.

6.3 References

- [1] Blog Name : “Data is the New Oil” by Michael Palmer
Blog Link : https://ana.blogs.com/maestros/2006/11/data_is_the_new.html
- [2] NLTK toolkit : <https://www.nltk.org/>
- [3] Written, Yoon Kim. (2014) *Convolutional Neural Networks for Sentence Classification*. In *Proceedings of EMNLP 2014*
- [4] Written, Zhang, Y., Wallace, B. (2015). *A Sensitivity Analysis of (and Practitioners’ Guide to) Convolutional Neural Networks for Sentence Classification*.
- [5] Written, Ronan Collobert., Jason Weston. (2008). *A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning*.
- [6] <https://spacy.io/usage/linguistic-features>
- [7] <https://nlp.stanford.edu/software/CRF-NER.html>
- [8] <https://railwayapi.com/>
- [9] Written, Y. Bengio, R. Ducharme, P. Vincent. 2003. *Neural Probabilistic Language Model*. In *Journal of Machine Learning Research* 3:1137–1155.
- [10] Written, R. Collobert, J. Weston, L. Bottou, M. Karlen, K.Kavukcuglu, P. Kuksa. 2011. *Natural Language Processing (Almost) from Scratch*. In *Journal of Machine Learning Research* 12:2493–2537.
- [11] Written, Lian Meng., Minlie Huang.,(2017) *Dialogue Intent Classification with Long Short-Term Memory Networks*
- [12] S. A. Ali, N. Sulaiman, A. Mustapha, and N. Mustapha. *Improving accuracy of intentionbased response classification using decision tree*. *Information Technology Journal*, 8(6), 2009.

- [13] J. Duchi, E. Hazan, Y. Singer. 2011 *Adaptive subgradient methods for online learning and stochastic optimization*. Journal of Machine Learning Research, 12:2121–2159.
- [14] L. Dong, F. Wei, S. Liu, M. Zhou, K. Xu. 2014. *A Statistical Parsing Framework for Sentiment Classification*. CoRR, abs/1401.6330.
- [15] A. Graves, A. Mohamed, G. Hinton. 2013. *Speech recognition with deep recurrent neural networks*. In Proceedings of ICASSP 2013.
- [16] G. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, R. Salakhutdinov. 2012. *Improving neural networks by preventing co-adaptation of feature detectors*. CoRR, abs/1207.0580.
- [17] K. Hermann, P. Blunsom. 2013. *The Role of Syntax in Vector Space Models of Compositional Semantics*. In Proceedings of ACL 2013.
- [18] M. Hu, B. Liu. 2004. *Mining and Summarizing Customer Reviews*. In Proceedings of ACM SIGKDD 2004.
- [19] M. Iyyer, P. Enns, J. Boyd-Graber, P. Resnik. 2014. *Political Ideology Detection Using Recursive Neural Networks*. In Proceedings of ACL 2014.
- [20] N. Kalchbrenner, E. Grefenstette, P. Blunsom. 2014. *A Convolutional Neural Network for Modelling Sentences*. In Proceedings of ACL 2014.
- [21] A. Krizhevsky, I. Sutskever, G. Hinton. 2012. *ImageNet Classification with Deep Convolutional Neural Networks*. In Proceedings of NIPS 2012.
- [22] Q. Le, T. Mikolov. 2014. *Distributed Representations of Sentences and Documents*. In Proceedings of ICML 2014.
- [23] B. Pang, L. Lee. 2004. *A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts*. In Proceedings of ACL 2004.
- [24] B. Pang, L. Lee. 2005. *Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales*. In Proceedings of ACL 2005.
- [25] A.S. Razavian, H. Azizpour, J. Sullivan, S. Carlsson. 2014. *CNN Features off-the-shelf: an Astounding Baseline*. CoRR, abs/1403.6382.

- [26] Y. Shen, X. He, J. Gao, L. Deng, G. Mesnil. 2014. *Learning Semantic Representations Using Convolutional Neural Networks for Web Search*. In Proceedings of WWW 2014.
- [27] J. Silva, L. Coheur, A. Mendes, A. Wichert. 2011. *From symbolic to sub-symbolic information in question classification*. Artificial Intelligence Review, 35(2):137–154.
- [28] R. Socher, J. Pennington, E. Huang, A. Ng, C. Manning. 2011. *Semi-Supervised Recursive Autoencoders for Predicting Sentiment Distributions*. In Proceedings of EMNLP 2011.
- [29] R. Socher, B. Huval, C. Manning, A. Ng. 2012. *Semantic Compositionality through Recursive Matrix- Vector Spaces*. In Proceedings of EMNLP 2012.
- [30] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. Manning, A. Ng, C. Potts. 2013. *Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank*. In Proceedings of EMNLP 2013.
- [31] J. Wiebe, T. Wilson, C. Cardie. 2005. *Annotating Expressions of Opinions and Emotions in Language*. Language Resources and Evaluation, 39(2-3): 165– 210.
- [32] S. Wang, C. Manning. 2012. *Baselines and Bigrams: Simple, Good Sentiment and Topic Classification*. In Proceedings of ACL 2012.
- [33] S. Wang, C. Manning. 2013. *Fast Dropout Training*. In Proceedings of ICML 2013.
- [34] B. Yang, C. Cardie. 2014. *Context-aware Learning for Sentence-level Sentiment Analysis with Posterior Regularization*. In Proceedings of ACL 2014.
- [35] W. Yih, K. Toutanova, J. Platt, C. Meek. 2011. *Learning Discriminative Projections for Text Similarity Measures*. Proceedings of the Fifteenth Conference on Computational Natural Language Learning, 247–256.
- [36] <https://thespoon.tech/meet-heston-bot-a-skype-bot-that-helps-you-figure-out-what-to-make-for-dinner/>
- [37] Written, Heung-yeung SHUM., Xiao-dong HE., Di LI. (2018)., *From Eliza to XiaoIce: challenges and opportunities with social chatbots*, In D. Frontiers Inf Technol Electronic Eng (2018) 19: 10., <https://doi.org/10.1631/FITEE.1700826>