**Project Report (6th Month)**

**On**

**“CHATBOT USING DEEP LEARNING”**

**Submitted in partial fulfilment of the requirements for the award of the degree of**

Bachelor of Technology In

# Computer Science Engineering



**Submitted by: Submitted to:**

**“VIVEK REDHU” “DHIRAJ KHURANA”**

19609, SEMESTER 8 “CSE, UIET, MDU”

UNIVERSITY INSTITUTE OF ENGINEERING & TECHNOLOGY

MAHARSHI DAYANAND UNIVERSITY,

ROHTAK – 124001

JUNE/JULY 2021

# Abstract

Dialogue Generation or Intelligent Conversational Agent development using Artificial Intelligence or Machine Learning technique is an interesting problem in the field of Natural Language Processing. In many research and development projects, they are using Artificial Intelligence, Machine Learning algorithms and Natural Language Processing techniques for developing conversation/dialogue agent. Their research and development are still under progress and under experimentation. Dialogue/conversation agents are predominately used by businesses, government organizations and non-profit organizations. They are frequently deployed by financial organizations like bank, credit card companies, businesses like online retail stores and start-ups. These virtual agents are adopted by businesses ranging from very small start-ups to large corporations. There are many chatbot development frameworks available in market both code based and interface based. But they lack the flexibility and usefulness in developing real dialogues. Among popular intelligent personal assistants includes Apple’s Siri, Amazon’s Alexa, Microsoft’s Cortana and Google’s Google Assistant.

The functioning of these agents is limited, are retrieval-based agent and also, they are not aimed at holding conversations which emulate real human interaction. Among current chatbots, many are developed using rule-based techniques, simple machine learning algorithms or retrieval-based techniques which do not generate good results. In this project, I have developed intelligent conversational agent using a special recurrent neural network Long-Short-Term-Memory (LSTM). For developing intelligent chatbot, I have used TensorFlow, Keras and Natural Language Processing ToolKit (NLTK).

## ACKNOWLEDGEMENT

I am highly grateful to the Dr Yudhvir Singh, Director, University Institute of Engineering and Technology (UIET) for providing this opportunity to carry out the major project work at UIET, MDU.

The constant guidance and encouragement received from Dr VIKAS SIWACH, H.O.D., CSE Department, UIET, MDU, Rohtak has been of great help in carrying out the project work and is acknowledged with reverential thanks.

I would like to express a deep sense of gratitude and thanks profusely to *Dhiraj Khurana*, without his wise counsel and able guidance, it would have been impossible to complete the project in this manner.

I express gratitude to other faculty members of Computer Science and Engineering Department of UIET, MDU, Rohtak for their intellectual support throughout the course of this work.

Finally, I am indebted to all whosoever have contributed in this report work.

**VIVEK REDHU**

**19609**

**Semester 8**

### B. Tech (CSE), UIET, MDU

**CANDIDATE DECLARATION CERTIFICATE**

I hereby certify that the work which is being presented in the project report entitled “CHATBOT USING DEEP LEARNING” by VIVEK REDHU in partial fulfilment of requirements for the award of degree of B. Tech submitted to Department of CSE, U.I.E.T at M.D.U Rohtak is an authentic record of my own work carried out during a period from January to July, 2021 under the supervision of DHIRAJ KHURANA.

Signature of the Student

This is to certify that the above statement by the candidate is correct to the best of my knowledge.

Signature of the SUPERVISOR

# LIST OF FIGURES

**Fig. No. Figure Description Page No.**

1.1 Reference Introduction 9

2.1 Sequence to Sequence Model 10

2.2 Long-Short-Term-Memory 11

3.1 Undesirable Chatbot Replies 16

4.1 Recurrent Neural Network 17

5.1 Hidden Layer of LSTM 19

5.2 LSTM Cell 20

11.1 GUI 33

# LIST OF TABLES

|  |  |
| --- | --- |
| **Table No. Table Description** | **Page No.** |
| 8.1 Algorithm Details | 31 |
| 8.2 Front End and Back End | 31 |
| 9.1 Hardware Platform- Local Machine (Laptop) | 32 |
| 10.1 Result | 32 |
|  |  |

# TABLE OF CONTENTS

|  |  |
| --- | --- |
| **Contents** | **Page No.** |
| ***Abstract*** | ***ii*** |
| ***Acknowledgement*** | ***iii*** |
| ***Candidate Declaration Certificate*** | ***iv*** |
| ***List of Figures*** | ***v*** |
| ***List of Tables*** | ***vi*** |
| ***Table of Contents*** | ***vii*** |
|  |  |

1. **Introduction 9**
2. **Related Works 10**
   1. Sequence to Sequence (Seq2Seq) ………………………………….10
   2. Keras ………………...……………………….…………….………12
      1. Keras and TensorFlow ……………………………………….12
      2. Keras Principles ……………………………………………...12
      3. Why Keras? ………………………………………………….13
      4. Keras Back Ends……………………………………………...13
      5. Keras Models…………………………………………………13
         1. Keras Sequential Models ……………………………...14
         2. Keras Functional API …………………………………15
      6. Keras Layers …………………………………………………15
3. **Limitations 16**
4. **Deep Neural Network for Chatbot 17**
   1. Recurrent Neural Network ………………………………...………17
      1. Recurrent Neural Network Architecture…………....…….….17
      2. Long-Short-Term-Memory ……………………….…………18
5. **Architecture 19**
   1. Long-Short-Term-Memory (LSTM) Architecture…………………19
6. **Data 21**
   1. Data Collection …………………………………………………….21
      1. Dataset: Intents ………………………………………………21
   2. Data Pre-processing ………………………………………………..23
7. **Source Code 25**
   1. Trainingchatbot.py …………………………………………………25
   2. Chatbotapp.py ……………………………………………………...28
8. **Implementation Summary 31**
9. **Hardware Specification 32**
10. **Result 32**
11. **Graphical User Interface 33**
12. **Challenges 33**
13. **Discussion 34**
14. **Future Work 34**
15. **Conclusion 35**
16. **Introduction**

Conversational agent or Chatbot is a program that generates response based on given input to emulate human conversations in text or voice mode. These applications are designed to simulate human-human interactions. Chatbots are predominantly used in business and corporate organizations including government, non-profit and private ones. Their functioning can range from customer service, product suggestion, product inquiry to personal assistant. Many of these chat agents are built using rule-based techniques, retrieval techniques or simple machine learning algorithms. In retrieval-based techniques, chat agents scan for keywords within the input phrase and retrieves relevant answers based on the query string. They rely on keyword similarity and retrieved text is pulled from internal or external data sources including world wide web or organizational database. Some other advanced chatbots are developed with natural language processing (NLP) techniques and machine learning algorithms. Also, there are many commercial chat engines available, which help build chatbots based on client data input.

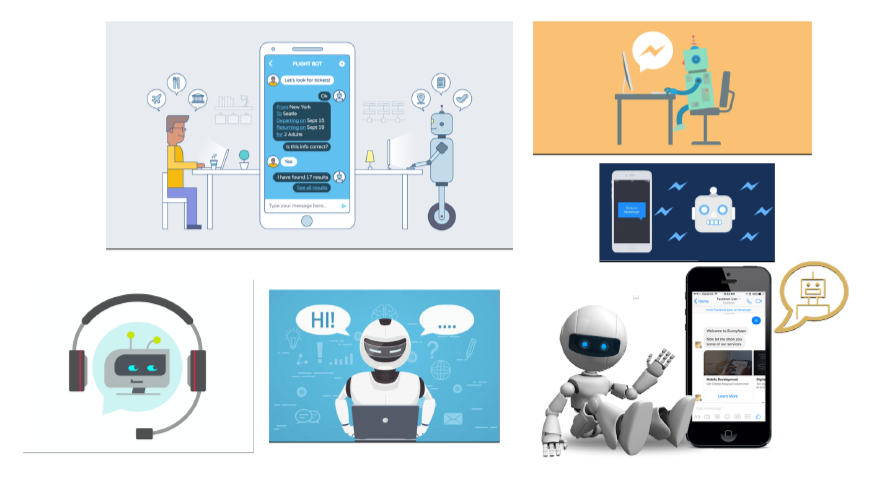


Figure 1.1: Reference Introduction

Recently, there have been major increase of interest in use and deployment of dialogue generation systems. Many major tech companies are using virtual assistant or chat agent to fill the needs of customers. Some of them include Google’s Google Assistant, Microsoft’s Cortana and Amazon’s Alexa. Though they are primarily question answering systems, their adoption by major corporations has peaked interest in customers and seems promising for more advanced conversational agent system research and development.

1. **Related Works**

There have been many recent development and experimentation in conversational agent system. Apart from traditional chatbot development techniques that use rule-based techniques, or simple machine learning algorithms, many advanced chatbots are using advanced Natural Language Processing (NLP) techniques and Deep Learning Techniques like Deep Neural Network (DNN) and Deep Reinforcement Learning (DRL).

* 1. **Sequence to Sequence (Seq2Seq)**

Some of the state-of-the-art techniques involve using Deep Neural Network and its architectural variations. Sequence to Sequence (Seq2Seq) model based on encoder-decoder architecture is such an architecture which is very popular for dialogue generation, language modelling and machine translation. Seq2Seq uses Recurrent Neural Network (RNN) which is a popular Deep Neural Network architecture specially for Natural Language Processing tasks. In Sequence to Sequence (Seq2Seq) model, many to many RNN architecture is used for decoder. In this, encoder-decoder architecture, input sequence is fed as a vector representation of text to encoder. Then, encoder produces some intermediate representation of information or thought vectors. Consequently, the thought vector generated by encoder is fed into decoder as input. Finally, decoder processes the thought vector and converts the sequence one by one word and produces multiple output from the decoder in form of target sequence. Though, vanilla RNN is default in Seq2Seq and works well for many NLP problems yet, due to higher complexity of language modelling problem, vanilla recurrent neural network cells often fail, specially, where long sequence of information needs to be remembered, as this information frequently becomes large for bigger datasets and turns to information bottleneck for the RNN network. Therefore, researchers use variations of recurrent neural network to handle such problem

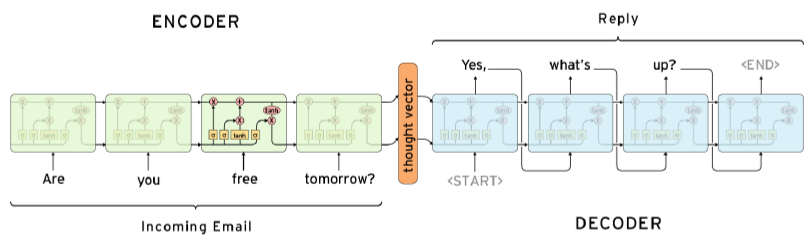


Figure 2.1: Sequence to Sequence Model

**Long-Short-Term-Memory (LSTM)** is a special variant of cell type of Recurrent Neural Network which has empirically shown to work well for language modelling. LSTM has forget gates along with input gates and output gates. This helps remember more relevant and contextual information and discards the rest of the sequence which is desirable in language modelling where dependency with sequence is sparse. Also, instead of using unidirectional cells, bidirectional LSTM cells can perform much better.

Another technique, Neural Attention Mechanism embedded in Seq2Seq module has significantly improved performance in dialogue generation system and other NLP tasks and thus become industry standard practice. In Neural attention mechanism, each hidden target compares with source hidden state, generates attention vector by calculating score and preserves the attention vector in memory to choose over other candidate. Also, other techniques like, Beam Search can help improve decoding performance further by choosing top candidates. Seq2Seq have also been applied for other NLP tasks including machine translation, text summarization and question-answering and image captioning.

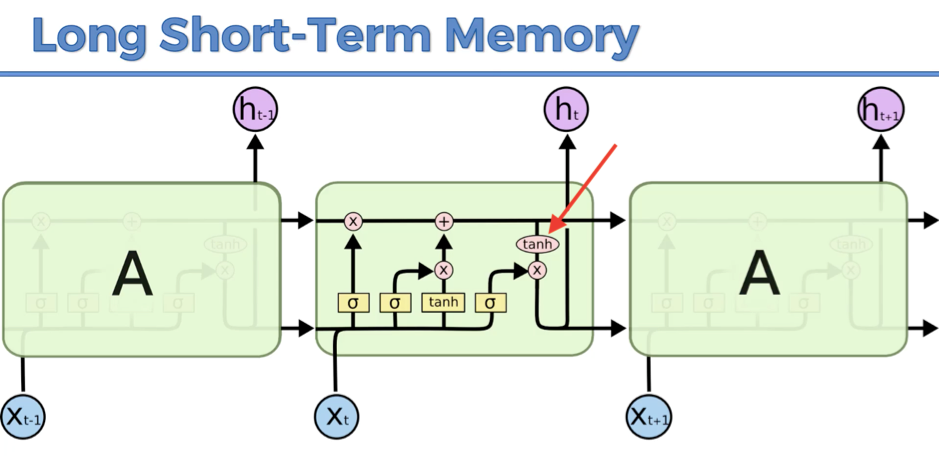


Figure 2.2: Long-Short-Term-Memory

* 1. **Keras**

While deep neural networks are all the rage, the complexity of the major frameworks has been a barrier to their use for developers new to machine learning. There have been several proposals for improved and simplified high-level APIs for building neural network models, all of which tend to look similar from a distance but show differences on closer examination.

Keras is one of the leading high-level neural networks APIs. It is written in Python and supports multiple back-end neural network computation engines.

* + 1. **Keras and TensorFlow**

Given that the TensorFlow project has adopted Keras as the high-level API for the upcoming TensorFlow 2.0 release, Keras looks to be a winner, if not necessarily the winner. In this article, we'll explore the principles and implementation of Keras, with an eye towards understanding why it’s an improvement over low-level deep learning APIs.

Even in TensorFlow 1.12, the official Get Started with TensorFlow tutorial uses the high-level Keras API embedded in TensorFlow, **tf.keras**. By contrast, the TensorFlow Core API requires working with TensorFlow computational graphs, tensors, operations, and sessions, some of which can be hard to understand when you're just beginning to work with TensorFlow. There are some advantages to using the low-level TensorFlow Core API, mostly when debugging, but fortunately you can mix the high-level and low-level TensorFlow APIs as needed.

* + 1. **Keras principles**

Keras was created to be user friendly, modular, easy to extend, and to work with Python. The API was “designed for human beings, not machines,” and “follows best practices for reducing cognitive load.”

Neural layers, cost functions, optimizers, initialization schemes, activation functions, and regularization schemes are all standalone modules that you can combine to create new models. New modules are simple to add, as new classes and functions. Models are defined in Python code, not separate model configuration files.

* + 1. **Why Keras?**

The biggest reasons to use Keras stem from its guiding principles, primarily the one about being user friendly. Beyond ease of learning and ease of model building, Keras offers the advantages of broad adoption, support for a wide range of production deployment options, integration with at least five back-end engines (TensorFlow, CNTK, Theano, MXNet, and PlaidML), and strong support for multiple GPUs and distributed training. Plus, Keras is backed by Google, Microsoft, Amazon, Apple, Nvidia, Uber, and others.

* + 1. **Keras back ends**

Keras proper does not do its own low-level operations, such as tensor products and convolutions; it relies on a back-end engine for that. Even though Keras supports multiple back-end engines, its primary (and default) back end is TensorFlow, and its primary supporter is Google. The Keras API comes packaged in TensorFlow as tf.keras, which as mentioned earlier will become the primary TensorFlow API as of TensorFlow 2.0.

To change back ends, simply edit your $HOME/.keras/keras.json file and specify a different back-end name, such as theano or CNTK. Alternatively, you can override the configured back end by defining the environment variable KERAS\_BACKEND, either in your shell or in your Python code using the os.environ["KERAS\_BACKEND"] property.

* + 1. **Keras models**

The **Model** is the core Keras data structure. There are two *main* types of models available in Keras: The Sequential model, and the Model class used with the functional API.

* + - 1. **Keras Sequential models**

The Sequential model is a linear stack of layers, and the layers can be described very simply. Here is an example from the Keras documentation that uses model.add() to define two dense layers in a Sequential model:

**import** keras  
**from** keras.models **import** **Sequential**  
**from** keras.layers **import** **Dense**  
  
*#Create Sequential model with Dense layers, using the add method*model = **Sequential**()  
  
*#Dense implements the operation:  
#        output = activation(dot(input, kernel) + bias)  
#Units are the dimensionality of the output space for the layer,  
#     which equals the number of hidden units  
#Activation and loss functions may be specified by strings or classes*model.add(**Dense**(units=64, activation='relu', input\_dim=100))  
model.add(**Dense**(units=10, activation='softmax'))  
  
*#The compile method configures the model’s learning process*model.compile(loss='categorical\_crossentropy',  
              optimizer='sgd',  
              metrics=['accuracy'])  
  
*#The fit method does the training in batches  
# x\_train and y\_train are Numpy arrays --just like in the Scikit-Learn API.*model.fit(x\_train, y\_train, epochs=5, batch\_size=32)  
  
*#The evaluate method calculates the losses and metrics  
#     for the trained model*loss\_and\_metrics = model.evaluate(x\_test, y\_test, batch\_size=128)  
  
*#The predict method applies the trained model to inputs  
#     to generate outputs*classes = model.predict(x\_test, batch\_size=128)

The comments in the code above are worth reading. It’s also worth noting how little cruft there is in the actual code compared to, say, the low-level TensorFlow APIs. Each layer definition requires one line of code, the compilation (learning process definition) takes one line of code, and fitting (training), evaluating (calculating the losses and metrics), and predicting outputs from the trained model each take one line of code.

* + - 1. **Keras functional API**

The Keras Sequential model is simple but limited in model topology. The Keras functional API is useful for creating complex models, such as multi-input/multi-output models, directed acyclic graphs (DAGs), and models with shared layers.

The functional API uses the same layers as the Sequential model but provides more flexibility in putting them together. In the functional API you define the layers first, and then create the Model, compile it, and fit (train) it. Evaluation and prediction are essentially the same as in a Sequential model, so have been omitted in the sample code below.

**from** keras.layers **import** **Input**, **Dense**  
**from** keras.models **import** **Model**  
  
*# This returns a tensor*inputs = **Input**(shape=(784,))  
  
*# a layer instance is callable on a tensor, and returns a tensor*x = **Dense**(64, activation='relu')(inputs)  
x = **Dense**(64, activation='relu')(x)  
predictions = **Dense**(10, activation='softmax')(x)  
  
*# This creates a model that includes  
# the Input layer and three Dense layers*model = **Model**(inputs=inputs, outputs=predictions)  
model.compile(optimizer='rmsprop',  
              loss='categorical\_crossentropy',  
              metrics=['accuracy'])  
model.fit(data, labels)  *# starts training*

* + 1. **Keras layers**

In the previous examples we only used Dense layers. Keras has a wide selection of predefined layer types, and also supports writing your own layers.

Core layers include Dense (dot product plus bias), Activation (transfer function or neuron shape), Dropout (randomly set a fraction of input units to 0 at each training update to avoid overfitting), Lambda (wrap an arbitrary expression as a Layer object), and several others. Convolution layers (the use of a filter to create a feature map) run from 1D to 3D and include the most common variants, such as cropping and transposed convolution layers for each dimensionality. 2D convolution, which was inspired by the functionality of the visual cortex, is commonly used for image recognition.

1. **Limitations**

Although, there are many chatbots currently available, majority of them are limited in functionality, domain function, context and coherence. They often fail in long conversations and have reduced relevancy in dialogue generation. Most of these chatbots are developed for restricted domain. Majority of them are using simple rule-based techniques. They perform well in question answering sessions and in very structured conversational modes. But, fail to emulate real human conversation and lacks flexibility in functioning. Some of the chatbots using machine learning algorithms often adhere to simple algorithms. They lack in complexity and sophistication needed to produce good result specifically in open domain conversation. Some chat engines are available in market which are often used by businesses for developing automated customer support. They are also black box and business clients have limited knowledge of their internal architectures. Hence, they produce results that can become unreliable and fail to fill the need of customers. Following is an example of failed chatbot replies.

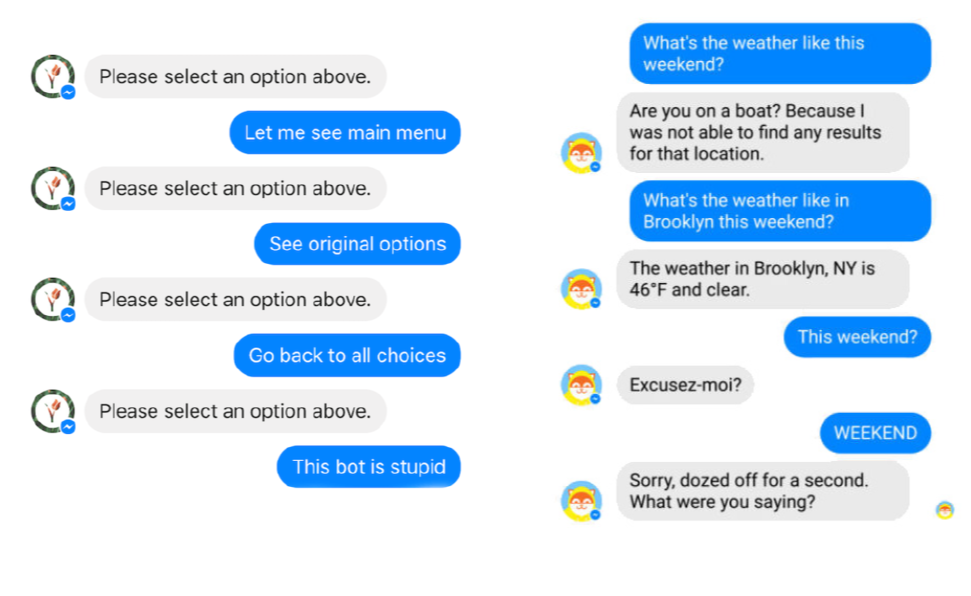


Figure 3.1: Undesirable Chatbot Replies

1. **Deep Neural Network for Chatbot**
   1. **Recurrent neural Network**

Recurrent Neural Network is a special Deep Neural Network Architecture used predominantly in Natural Language Processing (NLP) problems. In traditional Deep Neural Network, memory or sequence information is not taken into account. But, in Recurrent Neural Network, the sequential information is stored in memory and utilized for further processing which makes RNN suitable for sequential data or time series data where dependency exists in sequence.

* + 1. **Recurrent Neural Network Architecture**

Recurrent Neural Network (RNN) is composed of input layer, multiple hidden layers and output layer. In input layer, input is feed as vector representation. Then, input vector is multiplied by some weight and some biases are added. Then, the output from input layer is passed to next hidden layer where each consecutive hidden layer is composed of numerous RNN cells. After getting output from input layer, the cells in hidden layer multiplies the generated output from input layer by their own cell weights and biases. Next, in each of the hidden layer cells, some global activation function (sigmoid, tangent) is applied to generate output from hidden layer. Then, output from each hidden layer cells is passed to successive hidden layer. Similar to previous hidden layer cells, some weight, biases and activation function is applied to the input of current hidden layer cell. This procedure propagates though all consequent hidden layers. Finally, output generated from the final hidden layer is passed to output layer and the output layer applies some function (e.g., Softmax) to generate final output. For RNN, the output vector from final output layer is then again fed into the input layer as an input vector. Hence, the sequence information is stored in the memory and utilized.

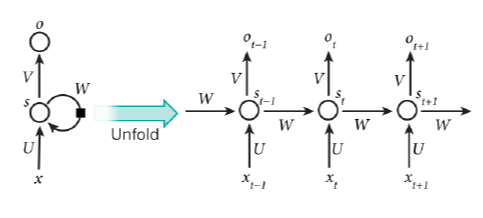


Figure 4.1: Recurrent Neural Network

* + 1. **Long-Short-Term-Memory (LSTM)**

Long-Short-Term-Memory (LSTM) is a special type of Recurrent Neural Network cell, which solves the data bottleneck for longer sequences. LSTM has forget gate along with the input and output gates. This helps remember longer sequence without overloading the network by discarding less relevant information.

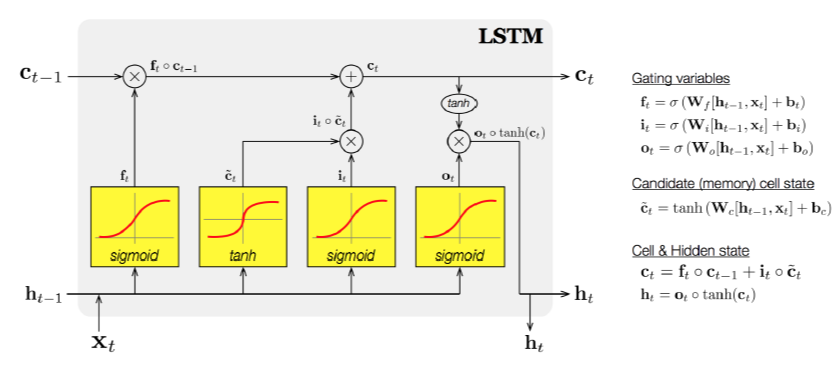
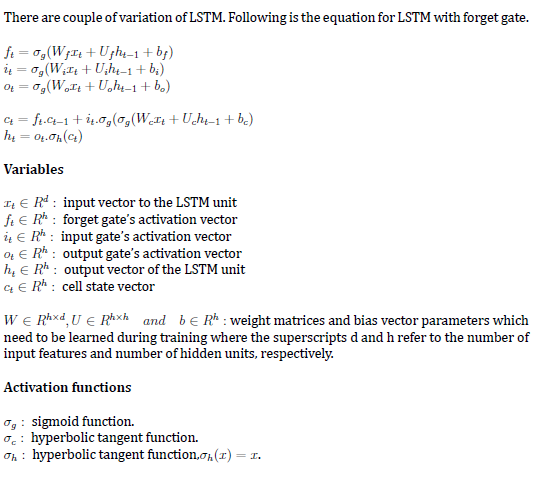


Figure 4.2: Long-Short-Term-Memory (LSTM)



1. **Architecture**
   1. **Long-Short-Term-Memory (LSTM) Architecture**

The basic difference between the architectures of RNNs and LSTMs is that the hidden layer of LSTM is a gated unit or gated cell. It consists of four layers that interact with one another in a way to produce the output of that cell along with the cell state. These two things are then passed onto the next hidden layer. Unlike RNNs which have got the only single neural net layer of tanh, LSTMs comprises of three logistic sigmoid gates and one tanh layer. Gates have been introduced in order to limit the information that is passed through the cell. They determine which part of the information will be needed by the next cell and which part is to be discarded. The output is usually in the range of 0-1 where ‘0’ means ‘reject all’ and ‘1’ means ‘include all’.

**Hidden layers of LSTM :**

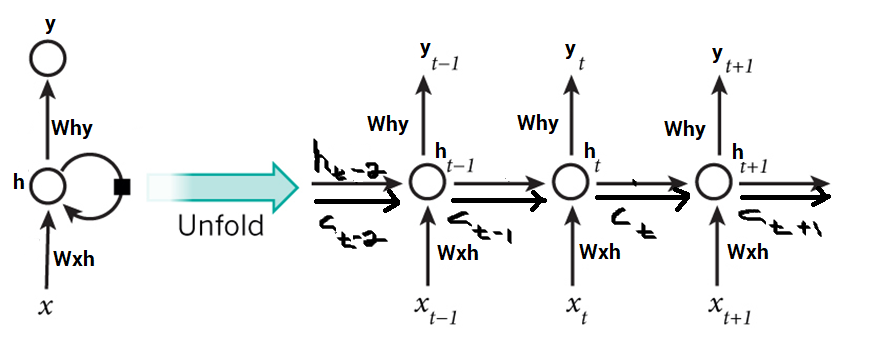


Figure 5.1: Hidden Layer of LSTM

Each LSTM cell has three inputs and two outputs. For a given time t, ht is the hidden state, Ct is the cell state or memory, xt is the current data point or input. The first sigmoid layer has two inputs– h(t-1) and xt where h(t-1) is the hidden state of the previous cell. It is known as the forget gate as its output selects the amount of information of the previous cell to be included. The output is a number in [0,1] which is multiplied (point-wise) with the previous cell state C(t-1).

**Conventional LSTM:**

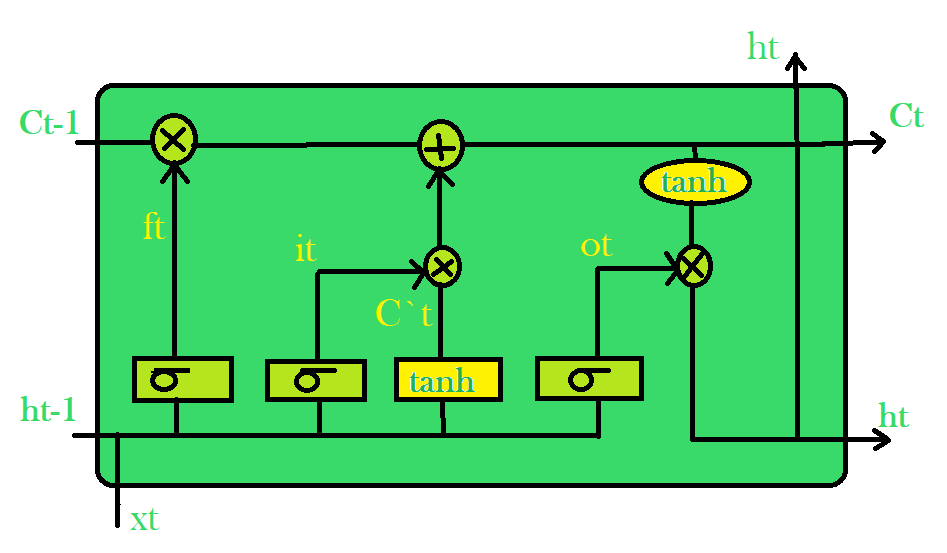


Figure 5.2: LSTM Cell

The second sigmoid layer is the input gate that decides what new information is to be added to the cell. It takes two inputs. The tanh layer creates a vector Ct of the new candidate values. Together, these two layers determine the information to be stored in the cell state. Their point-wise multiplication (it?Ct) tells us the amount of information to be added to the cell state. The result is then added with the result of the forget gate multiplied with previous cell state (ft\*Ct-1) to produce the current cell state Ct. Next, the output of the cell is calculated using a sigmoid and a tanh layer. The sigmoid layer decides which part of the cell state will be present in the output whereas tanh layer shifts the output in the range of [-1,1]. The results of the two layers undergo point-wise multiplication to produce the output ht of the cell.

1. **Data**
   1. **Data Collection**

In this project, the dataset” Intents” has been primarily used for final model training. Due to lack of data quality, big amount of data has been eliminated from final training. For future work, more robust and real-life conversation-based corpus can be incorporated for dialogue generation model building if available.

* + 1. **Dataset: Intents**

**Intents.json –** The data file which has predefined patterns and responses.

**This is how our intents.json file looks like:**

{"intents": [

{"tag": "greeting",

"patterns": ["Hi there", "How are you", "Is anyone there?","Hey","Hola", "Hello", "Good day"],

"responses": ["Hey", "Hello", "Good to see you again", "Hi there, how can I help?"],

"context": [""]

},

{"tag": "goodbye",

"patterns": ["Bye", "See you later", "Goodbye", "Nice chatting to you, bye", "Till next time"],

"responses": ["See you!", "Have a nice day", "Bye! Come back again soon."],

"context": [""]

},

{"tag": "thanks",

"patterns": ["Thanks", "Thank you", "That's helpful", "Awesome, thanks", "Thanks for helping me"],

"responses": ["Happy to help!", "Any time!", "My pleasure"],

"context": [""]

},

{"tag": "noanswer",

"patterns": [],

"responses": ["Sorry, can't understand you", "Please give me more info", "Not sure I understand"],

"context": [""]

},

{"tag": "options",

"patterns": ["How you could help me?", "What you can do?", "What help you provide?", "How you can be helpful?", "What support is offered"],

"responses": ["I can guide you through Adverse drug reaction list, Blood pressure tracking, Hospitals and Pharmacies", "Offering support for Adverse drug reaction, Blood pressure, Hospitals and Pharmacies"],

"context": [""]

},

{"tag": "adverse\_drug",

"patterns": ["How to check Adverse drug reaction?", "Open adverse drugs module", "Give me a list of drugs causing adverse behavior", "List all drugs suitable for patient with adverse reaction", "Which drugs dont have adverse reaction?" ],

"responses": ["Navigating to Adverse drug reaction module"],

"context": [""]

},

{"tag": "blood\_pressure",

"patterns": ["Open blood pressure module", "Task related to blood pressure", "Blood pressure data entry", "I want to log blood pressure results", "Blood pressure data management" ],

"responses": ["Navigating to Blood Pressure module"],

"context": [""]

},

{"tag": "blood\_pressure\_search",

"patterns": ["I want to search for blood pressure result history", "Blood pressure for patient", "Load patient blood pressure result", "Show blood pressure results for patient", "Find blood pressure results by ID" ],

"responses": ["Please provide Patient ID", "Patient ID?"],

"context": ["search\_blood\_pressure\_by\_patient\_id"]

},

{"tag": "search\_blood\_pressure\_by\_patient\_id",

"patterns": ["PID"],

"responses": ["Your Blood Pressue while vaccination was 120/80"],

"context": [""]

},

{"tag": "pharmacy\_search",

"patterns": ["Find me a pharmacy", "Find pharmacy", "List of pharmacies nearby", "Locate pharmacy", "Search pharmacy" ],

"responses": ["Please provide pharmacy name"],

"context": ["search\_pharmacy\_by\_name"]

},

{"tag": "search\_pharmacy\_by\_name",

"patterns": [""],

"responses": ["Haryana Medicos, Medical Mod, Rotak", "Hooda Medicos, Sheela Bypass, Rohtak", "XYZ Medicos, Somewhere, Rohtak"],

"context": ["pharmacy\_details"]

},

{"tag": "hospital\_search",

"patterns": ["yes", "Lookup for hospital", "Search for hospital to transfer patient", "I want to search hospital data", "Hospital lookup for patient", "Looking up hospital details" ],

"responses": ["Please provide hospital location"],

"context": ["search\_hospital\_by\_location"]

},

{"tag": "search\_hospital\_by\_location",

"patterns": ["rohtak"],

"responses": ["Please provide hospital type"],

"context": ["search\_hospital\_by\_type"]

},

{"tag": "search\_hospital\_by\_type",

"patterns": ["ortho", "cardiac", "emergency", "trauma centre", "opd"],

"responses": ["Noble Heart Hospital at Delhi Bypass, Rohtak", "Holy Heart Hospital at Delhi Bypass, Rohtak", "PGIMS, Rohtak", "Kynos Hospital at Kalhawad, Rohtak", "Civil Hospital, Rohtak"],

"context": [""]

},

{"tag": "symptoms",

"patterns": ["I'm not feeling well", "Something is wrong", "I'm ill", "I'm sick", "I have been feeling sick", "i have been sick since"],

"responses": ["What are your symptoms"],

"context": ["symptoms\_reply"]

},

{"tag": "after\_vaccination",

"patterns": ["Since vaccinion", "after vaccination", "after vaccine"],

"responses": ["What are your symptoms?"],

"context": ["symptoms"]

},

{"tag": "symptoms\_reply",

"patterns": ["fever", "cough", "cold", "chill", "body ache", "head ache", "ache", "pain", "nausea", "vomiting", "anxiety", "sore"],

"responses": ["Should I search for a hospital near you?"],

"context": ["hospital\_search"]

}

]

}

* 1. **Data Pre-processing**

When working with text data, we need to perform various pre-processing on the data before we make a machine learning or a deep learning model. Based on the requirements we need to apply various operations to pre-process the data.

Tokenizing is the most basic and first thing you can do on text data. Tokenizing is the process of breaking the whole text into small parts like words.

Here we iterate through the patterns and tokenize the sentence using nltk.word\_tokenize() function and append each word in the words list. We also create a list of classes for our tags.

for intent in intents['intents']:

for pattern in intent['patterns']:

#tokenize each word

w = nltk.word\_tokenize(pattern)

words.extend(w)

#add documents in the corpus

documents.append((w, intent['tag']))

# add to our classes list

if intent['tag'] not in classes:

classes.append(intent['tag'])

Now we will lemmatize each word and remove duplicate words from the list. Lemmatizing is the process of converting a word into its lemma form and then creating a pickle file to store the Python objects which we will use while predicting.

# lemmatize, lower each word and remove duplicates

words = [lemmatizer.lemmatize(w.lower()) for w in words if w not in ignore\_words]

words = sorted(list(set(words)))

# sort classes

classes = sorted(list(set(classes)))

# documents = combination between patterns and intents

print (len(documents), "documents")

# classes = intents

print (len(classes), "classes", classes)

# words = all words, vocabulary

print (len(words), "unique lemmatized words", words)

pickle.dump(words,open('words.pkl','wb'))

pickle.dump(classes,open('classes.pkl','wb'))

1. **Source Code**
   1. **trainingchatbot.py**

import nltk

from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

import json

import pickle

import numpy as np

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Activation, Dropout

from tensorflow.keras.optimizers import SGD

import random

words=[]

classes = []

documents = []

ignore\_words = ['?', '!']

data\_file = open('intents.json').read()

intents = json.loads(data\_file)

for intent in intents['intents']:

for pattern in intent['patterns']:

#tokenize each word

w = nltk.word\_tokenize(pattern)

words.extend(w)

#add documents in the corpus

documents.append((w, intent['tag']))

# adding to our classes list

if intent['tag'] not in classes:

classes.append(intent['tag'])

# lemmaztize and lower each word and remove duplicates

words = [lemmatizer.lemmatize(w.lower()) for w in words if w not in ignore\_words]

words = sorted(list(set(words)))

# sort classes

classes = sorted(list(set(classes)))

# documents = combination between patterns and intents

print (len(documents), "documents")

# classes = intents

print (len(classes), "classes", classes)

# words = all words, vocabulary

print (len(words), "unique lemmatized words", words)

pickle.dump(words,open('words.pkl','wb'))

pickle.dump(classes,open('classes.pkl','wb'))

# create our training data

training = []

# create an empty array for our output

output\_empty = [0] \* len(classes)

# training set, bag of words for each sentence

for doc in documents:

# initialize our bag of words

bag = []

# list of tokenized words for the pattern

pattern\_words = doc[0]

# lemmatize each word - create base word, in attempt to represent related words

pattern\_words = [lemmatizer.lemmatize(word.lower()) for word in pattern\_words]

# create our bag of words array with 1, if word match found in current pattern

for w in words:

bag.append(1) if w in pattern\_words else bag.append(0)

# output is a '0' for each tag and '1' for current tag (for each pattern)

output\_row = list(output\_empty)

output\_row[classes.index(doc[1])] = 1

training.append([bag, output\_row])

# shuffle our features and turn into np.array

random.shuffle(training)

training = np.array(training)

# create train and test lists. X - patterns, Y - intents

train\_x = list(training[:,0])

train\_y = list(training[:,1])

print("Training data created")

# Create model - 3 layers. First layer 128 neurons, second layer 64 neurons and 3rd output layer contains number of neurons

# equal to number of intents to predict output intent with softmax

model = Sequential()

model.add(Dense(128, input\_shape=(len(train\_x[0]),), activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(64, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(len(train\_y[0]), activation='softmax'))

# Compile model. Stochastic gradient descent with Nesterov accelerated gradient gives good results for this model

sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)

model.compile(loss='categorical\_crossentropy', optimizer=sgd, metrics=['accuracy'])

#fitting and saving the model

hist = model.fit(np.array(train\_x), np.array(train\_y), epochs=1000, batch\_size=5, verbose=1)

model.save('chatbot\_model.h5', hist)

print("model created")

* 1. **chatbotapp.py**

import nltk

from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

import pickle

import numpy as np

from tensorflow.keras.models import load\_model

model = load\_model('chatbot\_model.h5')

import json

import random

intents = json.loads(open('intents.json').read())

words = pickle.load(open('words.pkl','rb'))

classes = pickle.load(open('classes.pkl','rb'))

def clean\_up\_sentence(sentence):

# tokenize the pattern - split words into array

sentence\_words = nltk.word\_tokenize(sentence)

# stem each word - create short form for word

sentence\_words = [lemmatizer.lemmatize(word.lower()) for word in sentence\_words]

return sentence\_words

# return bag of words array: 0 or 1 for each word in the bag that exists in the sentence

def bow(sentence, words, show\_details=True):

# tokenize the pattern

sentence\_words = clean\_up\_sentence(sentence)

# bag of words - matrix of N words, vocabulary matrix

bag = [0]\*len(words)

for s in sentence\_words:

for i,w in enumerate(words):

if w == s:

# assign 1 if current word is in the vocabulary position

bag[i] = 1

if show\_details:

print ("found in bag: %s" % w)

return(np.array(bag))

def predict\_class(sentence, model):

# filter out predictions below a threshold

p = bow(sentence, words,show\_details=False)

res = model.predict(np.array([p]))[0]

ERROR\_THRESHOLD = 0.25

results = [[i,r] for i,r in enumerate(res) if r>ERROR\_THRESHOLD]

# sort by strength of probability

results.sort(key=lambda x: x[1], reverse=True)

return\_list = []

for r in results:

return\_list.append({"intent": classes[r[0]], "probability": str(r[1])})

return return\_list

def getResponse(ints, intents\_json):

tag = ints[0]['intent']

list\_of\_intents = intents\_json['intents']

for i in list\_of\_intents:

if(i['tag']== tag):

result = random.choice(i['responses'])

break

return result

def chatbot\_response(msg):

ints = predict\_class(msg, model)

res = getResponse(ints, intents)

return res

#Creating GUI with tkinter

import tkinter

from tkinter import \*

def send():

msg = EntryBox.get("1.0",'end-1c').strip()

EntryBox.delete("0.0",END)

if msg != '':

ChatLog.config(state=NORMAL)

ChatLog.insert(END, "You: " + msg + '\n\n')

ChatLog.config(foreground="#442265", font=("Verdana", 12 ))

res = chatbot\_response(msg)

ChatLog.insert(END, "Bot: " + res + '\n\n')

ChatLog.config(state=DISABLED)

ChatLog.yview(END)

base = Tk()

base.title("AdverseEffectsBot")

base.geometry("400x500")

base.resizable(width=FALSE, height=FALSE)

#Creating Chat window

ChatLog = Text(base, bd=0, bg="white", height="8", width="50", font="Arial",)

ChatLog.config(state=DISABLED)

#Binding scrollbar to Chat window

scrollbar = Scrollbar(base, command=ChatLog.yview, cursor="heart")

ChatLog['yscrollcommand'] = scrollbar.set

#Creating Button to send message

SendButton = Button(base, font=("Verdana",12,'bold'), text="Send", width="12", height=5,

bd=0, bg="#32de97", activebackground="#3c9d9b",fg='#ffffff',

command= send )

#Creating the box to enter message

EntryBox = Text(base, bd=0, bg="white",width="29", height="5", font="Arial")

EntryBox.bind("<Return>", send)

#Placing all components on the screen

scrollbar.place(x=376,y=6, height=386)

ChatLog.place(x=6,y=6, height=386, width=370)

EntryBox.place(x=128, y=401, height=90, width=265)

SendButton.place(x=6, y=401, height=90)

base.mainloop()

1. **Implementation Summary**

|  |  |
| --- | --- |
| DEEP LEARNING MODULE : | TensorFlow v2.5.0 |
| ALGORITHM : | Deep Neural Network (DNN), Recurrent Neural Network (RNN) |
| MAIN TECHNIQUE : | Seq2Seq Modelling with LSTM |
| ENHANCEMENTTECHNIQUES : | Natural Language Processing ToolKit (NLTK), Long-Short-Term-Memory Based RNN Cell, Bidirectional LSTM |

Table 8.1: Algorithm Details

|  |  |
| --- | --- |
| FRONT END : | Python with Tkinter |
| BACK END : | Python |

Table 8.2: Front End and Back End

1. **Hardware Specifications**

For training, personal laptop with 8th Generation Intel Core i5 processor, dedicated graphics- Nvidia GeForce940mx and 16 GB Ram has been used. For further extended training, cloud computing platform Amazon AWS can be deployed.

|  |  |
| --- | --- |
| PROCESSOR | 8th Gen Intel Core i5 |
| RAM | 16 GB DDR4 Memory |
| GRAPHICS CARD | Nvidia GeForce940mx |

Table 9.1: Hardware Platform- Local Machine (Laptop)

1. **Result**

Following are some response derived after training on full dataset with “intents.json” as training text with 1000 epochs. The initial test result produced moderately coherent sentences. The following responses were generated after inference from trained model.

Input(Person) Output(AdverseEffectsBot)

|  |  |
| --- | --- |
| hi | Hi there, how can I help? |
| I’m sick | What are your symptoms? |
| Fever And Cough | Should I search for a hospital near you? |
| Yes | Please provide your location |
| Rohtak | Please provide hospital type |
| Cardiac | Holy Heart Hospital at Delhi Bypass, Rohtak |
| Find pharmacy | Please provide pharmacy location |
| Rohtak | Haryana Medicos, Medical Mod, Rohtak |
| Blood pressure for patient | Patient ID? |
| PID 11223344 | Patient’s Blood Pressure while vaccination was 120/80 |

Table 10.1: Result

1. **Graphical User Interface (GUI)**

Desktop Graphical Interface (GUI) was developed using Python and Tkinter module. On left top side, the chat response and history are shown, and on the right side there is a scrollbar present to look at all texts. All possible response of input text is shown.

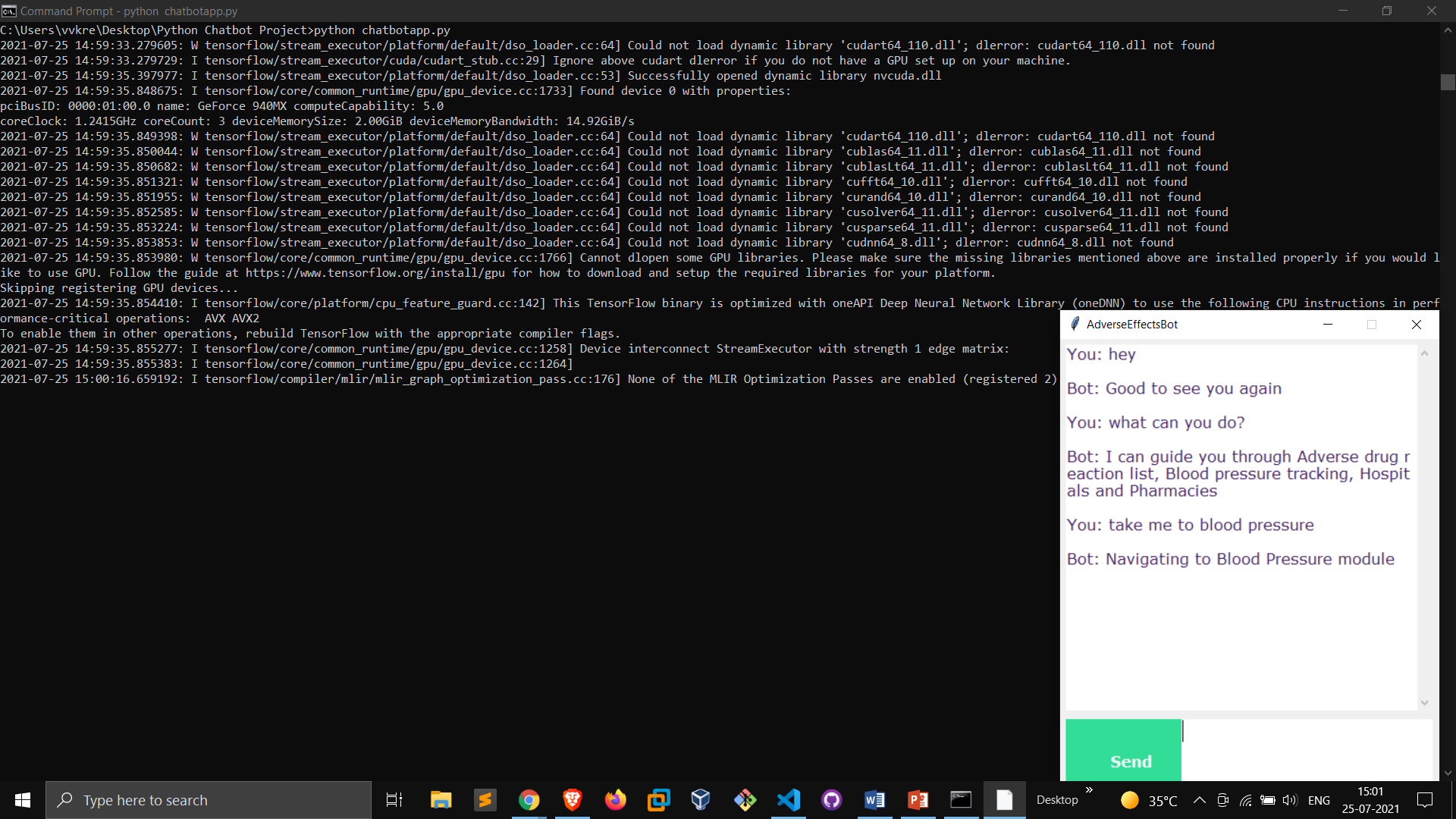


Figure 11.1: GUI

1. **Challenges**

The challenge in developing chatbot or dialogue generator lies in developing coherent dialogue generation system. As the model used in this experiment is for machine translation, the dialogue generation is treated as translation problem, where history of earlier conversations is not taken into account. Hence, the model can be limited in performance regarding long conversation. Also, training is a long process which demands higher processing power and configured computing machine. Another problem is finding right hyper parameters to optimize the translation module for chat bot or dialogue generation system

1. **Discussion**

Development of dialogue agent using Neural Machine Translation (NMT) is widely practiced. Some other approaches use only Sequence to Sequence modelling. Many people also use their own Sequence to Sequence module. But, due to their lack complexity they fail to perform well. However, with more time and effort building a more comprehensive dialogue generation unit could solve dialogue generation problem is unambiguously better. Hence, Sequence to Sequence module with encoder-decoder architecture built on bi directional LSTM cells and Neural attention mechanism, Beam Search could be a more preferable task.

Apart from algorithm improvement, performance could be further optimized if better datasets were available.

The intelligent chatbot has produced moderate result. But, some of replies looks repetitive and lacks proper relevancy. This can be reduced with addition of more diverse and healthy data. Also, adding more length to original text could help improve responses and make them more relevant. A large proportion of data was lost due to choosing restrictive length where utterance more than 100 length was discarded but later full-length text was adopted. This might have caused redundancy in training process. Also, repeated utterance by same character was eliminated and only last utterance by repeated speaker was kept, which further reduced data size. Therefore, more data with optimum sequence length can

help build more intelligent chatbot.

1. **Future Work**

The chatbot developed using TensorFlow can be further improved with more robust, high quality real-life conversational datasets which could better emulate human interaction. Also, hyper-parameters of the TensorFlow model can be further fine-tuned and optimized for performance enhancement. Based on available opportunity to further advance the project, Deep Reinforcement Learning (DRL) can be applied that could significantly improve performance. Reinforcement Learning algorithm can be applied after the initial.

1. **Conclusion**

The training on intents corpus produced result which needs further improvement and more attention and speculation on training parameters. Adding more quality data will further improve performance. Also, the training model should be trained with other hyper-parameters and different dataset for further experimentation. This was an attempt to experiment with Deep Neural Network for dialogue generation in order to develop intelligent chatbot