A Project Report
On

Hue: Chatbot Using Deep Learning

Submitted in partial fulfillment of the
Requirement for the award of the degree of

B.Tech in Computer Science and Engineering

GALGOTIAS UNIVERSITY
(Established under Galgotias University Uttar Pradesh Act No. 14 of 2011)

Under The Supervision of
Ms. Kiran Singh
Assistant Professor
Department of Computer Science and Engineering

Submitted By:
Dev Singh Chauhan
(18SCSE1180039)

Arpit Rastogi
(18SCSE1010581)

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GALGOTIAS UNIVERSITY, GREATER NOIDA
INDIA OCTOBER,2021

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
GALGOTIAS UNIVERSITY, GREATER NOIDA

CANDIDATE'S DECLARATION

I/We hereby certify that the work which is being presented in the thesis/project/dissertation, entitled **"HUE: CHATBOT USING DEEP LEARNING"** in partial fulfillment of the requirements for the award of **the Bachelor of Technology** submitted in **the School of Computing Science and Engineering of Galgotias University, Greater Noida**, is an original work carried out during the period of month, **JULY-2021 TO DECEMBER 2021**, under the supervision of **Ms. Kiran Singh (Assistant Professor), Department of Computer Science and Engineering** of School of Computing Science and Engineering , Galgotias University, Greater Noida.

The matter presented in the thesis/project/dissertation has not been submitted by me/us for the award of any other degree of this or any other places.

Dev Singh Chauhan, 18SCSE1180039

Arpit Rastogi, 18SCSE1010581

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

Supervisor

(Ms. Kiran Singh , Assistant Professor)

CERTIFICATE

The Final Thesis/Project/ Dissertation Viva-Voce examination of **Dev Singh Chauhan: 18SCSE1180039, Arpit Rastogi : 18SCSE1010581** has been held on _____ and his/her work is recommended for the award of **Bachelor of Technology In Computer Science and Engineering.**

Signature of Examiner(s) **Signature of Supervisor(s)**

Signature of Project Coordinator **Signature of Dean**

Date : 20 December, 2021

Place : Galgotias University, Greater Noida

ABSTRACT

Depression is a typical sickness worldwide with conceivably serious ramifications. Early distinguishing proof of burdensome side effects is a vital initial move towards appraisal, intercession, and backslide avoidance. With an expansion in informational indexes with importance for discouragement, and the headway of AI, there is a possibility to foster smart frameworks to recognize side effects of misery in composed material. This work proposes an effective methodology utilizing Long Short-Term Memory (LSTM)- based Recurrent Neural Network (RNN) to recognize texts portraying self-saw indications of depression. The methodology is applied on an enormous dataset from a public web-based data divert for youngsters in Norway. The dataset comprises of youth's own text-put together inquiries with respect to this data channel. Highlights are then given from a one-hot cycle on powerful elements extricated from the impression of potential manifestations of melancholy pre-characterized by clinical and mental specialists. The elements are superior to traditional methodologies, which are for the most part dependent on the word frequencies (i.e., some highest successive words are picked as highlights from the entire message dataset and applied to show the fundamental occasions in any instant message) rather than manifestations. Then, at that point, a profound learning approach is applied (i.e., RNN) to prepare the time-successive highlights separating texts depicting gloom manifestations from posts with no such portrayals (non-sorrow posts). At last, the prepared RNN is utilized to consequently anticipate melancholy posts. The framework is thought about against traditional methodologies where it accomplished predominant execution than others. The straight discriminant space obviously uncovers the vigor of the highlights by creating preferred grouping over other conventional elements.

Keywords – Depression, LSTM (Long Short Term Memory), chatbot, therapy

INDEX

List of Figures

CHAPTER-1

INTRODUCTON

Depression, or depression disorder, is a common disorder. According to the World Health Organization (WHO), the number of people suffering from depression is estimated at more than 300 million worldwide. Depression can seriously affect well-being and work, school, and family, and can even lead to self-injury. Adolescent depression is associated with emotional disorders and severe mental illness in adult life. About 0.8 million people die from suicide each year and suicide is the fourth leading cause of death among 15-19 year olds, according to the WHO. Of the major illnesses that contribute to disability or dysfunction, five are mental illnesses — the most common of which is depression. Therefore, the burden of disease due to depression is high. The prevalence of depression in adults is about 5% in all cultures, and 20% in your soft tissues (i.e., incomplete symptoms, mild depression, and possible depression). For older people, those most at risk are within the middle age group. Also, the incidence of depression worldwide is increasing, with an increase of 18% between 2005 and 2015. However, early intervention by a specialist can improve psychological symptoms (e.g., intestinal problems and sleep disorders) in many cases.

Early detection of symptoms of depression following diagnosis and treatment can significantly improve the chances of preventing symptoms and the underlying disease; reducing the negative social and health impacts as well as personal, economic, and social health. However, finding symptoms of depression is challenging and requires resources. Current methods are based on clinical discussions and questionnaires conducted by hospitals or agencies, where psychological analysis tables are used to make predictions of mental disorders. This method is based largely on individual questionnaires and can diagnose almost any psychological disorder of depression.

Another method of discussion or predictions based on a list of stress questions is to analyze informal texts provided by users. Previous research on medical psychology has shown that the relationship between language user (e.g., speaker or author) and their text is meaningful and powerful for the future. A recent study by Havigerová et al. demonstrate the power of text-based diagnosis for people at risk of depression, using a sample of informal text written about the holiday. Therefore, online records and data are increasingly seen as an important data source in supporting health care with decision support. The method of identifying stress symptoms in informal texts is promising, as it allows for the benefit of the latest advances in natural language processing and artificial intelligence (AI). AI has applied for natural language processing using language technologies and computer techniques to help machines understand basic things such as emotions or feelings from texts. If so, the main objective is to analyze ideas, opinions, and assumptions using the polarity allocation can be negative or positive.

Previous work has found that automatic analysis of depressive symptoms from texts can be used, for example, to recall emotions in suicidal notes and to find obscene or depressing words or sentences in conversations or blog posts. However, there is still a great deal of unused energy in research to remove symptoms of depression in the literature. Key challenges include presenting significant indicators of stress from the literature. Also, there is a major obstacle to finding symptoms of depression in short texts.

To contribute to solving these challenges, we aim to develop an automated algorithm for detecting depressive symptoms in texts, using a text-based sample of young people seeking advice on depressive symptoms they see. We believe that our automated discovery method, which explains user problems in the native language, could make a significant contribution to this research field. Therefore, the current study focuses on how depressive symptoms are expressed in text in a natural language using AI.

Among the various methods of analyzing the human body and mind from different data sources, machine learning has become widely used. As machine learning models are

gradually being used to make important predictions in important day-to-day situations, the need to demonstrate things increases in such situations from various stakeholders in the AI industry. The main danger in this is to make and apply AI decisions that are unfair and without explanations of the behavior of the models. Therefore, the definitions of the model effect are important. For example, specialists in the field of medicine need more information from machine learning models than simple speculation to support their diagnosis. Such needs may also arise in other areas, such as medical emergencies. Therefore, focusing only on the performance of AI models, gradually makes systems more susceptible to reactions in some cases. Therefore, the current study highlighted the importance of descriptive Artificial Intelligence (XAI) for establishing trust in decisions based on machine learning with descriptions of black box models. Popular modern definition algorithms include Model-Agnostic Definitions (LIME), Additional SHApley Definitions (SHAP), and Layer-wise relevance propagation (LRP). Since then, LIME has a very low weight but is trying to produce quick and satisfying explanations after the hoc. Therefore, this function accepts LIME to determine definitions (i.e., value of features) once a decision has been given to the model.

To visualize sample data of different groups in different applications, Line Discrimination Analysis (LDA) is a good discriminatory data-based tool. Works on collecting samples of similar classes. It seeks to find directions where classes are best divided by considering minimizing the internal disintegration of the classroom while maximizing inter-class dispersal. LDA has already been used in a variety of functional programs such as facial recognition and recognition of human activity. LDA is a sample data project for different classes in the lower vector area. Therefore, rates of intermediate phase dispersion and in-class dispersion are increased to achieve maximum discrimination.

1.1) ADVANTAGES OF USING CHATBOT

- Accessible anytime: chatbots are digital robots that never get tired and keep running. Throughout the year, they will continue to operate every day without having to take a break
- Flexible attribute: chatbots have the benefit that it can quite easily be used in any industry
- Communication with the user is easy
- Reduces human effort for reading files
- Easy for maintenance
- Anonymity, privacy, and security could be maintained with a privacy-enhancing approach.
- Chatbots are designed to be nonjudgmental and therefore more compassionate toward patient worries. This could reassure individuals to unhesitatingly unwrap.

1.2) PROBLEM FORMULATION

This review means to see how clients connect with and are diverted through a chatbot for sadness (Hue) to give plan proposals.

1.3) TECHNOLOGY USED

- Python = 3.6 - Python is an interpreted high-level general-purpose programming language. Its design philosophy emphasizes code readability with its use of significant indentation.

- Pandas= 0.22.0 - pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool ,built on top of the Python programming language.

- Numpy= 1.14.3 - The fundamental package for scientific computing with Python

- Tensorfow= 1.13 - The core open source library to help you develop and train ML models. Get started quickly by running Colab notebooks directly in your browser.

- Flask - Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries.

- Jupyter Notebook - The Jupyter Notebook is a web application for creating and sharing documents that contain code, visualizations, and text. It can be used for data science, statistical modeling, machine learning, and much more.

- Spyder- Spyder is a free and open source scientific environment written in Python, for Python, and designed by and for scientists, engineers and data analysts. It features a unique combination of the advanced editing, analysis, debugging, and profiling functionality of a comprehensive development tool with the data exploration, interactive execution, deep inspection, and beautiful visualization capabilities of a scientific package.

CHAPTER-2

LITERATURE SURVEY

2.1) Title:“ Emassnuela Haaller and Traiian Raebedea, “Designing a Chat-bot that Simulates an Historical Figure”, IEEE Conference Publications, July 2013.

There might be applications that are consolidating man-like appearance and expecting to copy human, however in the greater part of the situations the data of the conversational bot is put away in a db made by somebody who has delayed information in that field. In any case, scarcely any specialists might have examined making a Chat Bot utilizing a counterfeit person and character starting from pages or plain-text of someone in particular. The paper expounds bringing up the significant realities in texts clarifying the existence of an antiquated figure for making a specialist that is utilized in school situations.

2.2 Title: Maja Pantic, Reinier Zwitserloot, and Robbert Jan Grootjans, “Teaching Introductory Artificial Intelligence Using Asimple Agent Framework”, IEEE Transactions On Education, Vol. 48, No. 3, August 2005.

The paper clarify a method of showing man-made consciousness (AI) utilizing a certified, credulous specialist systems just for this course. However numerous specialist systems has been proposed in the writing, none of the accessible constructions was simple or easy to be utilized by to be alumni of CSE. The principle objective of utilizing such a review was to keep occupied the understudies into which they observed to be extremely fascinating. A useful methodology and a conventional methodology was utilized with the goal that understudies learn viably.

As of late, there are such countless organizations have created AI applications to help for discouraged individuals, up to this point give security and namelessness. These applications, focused on client, were created to proactively keep an eye on patients, be prepared to listen them also visit whenever, anyplace with suggest exercises which attempts to work on client's psychological needs. One of the fruitful chatbots that have been in the market up to this point is woebot which filter the states of mind and make a stage wherein clients will communicate their contemplations and feelings.

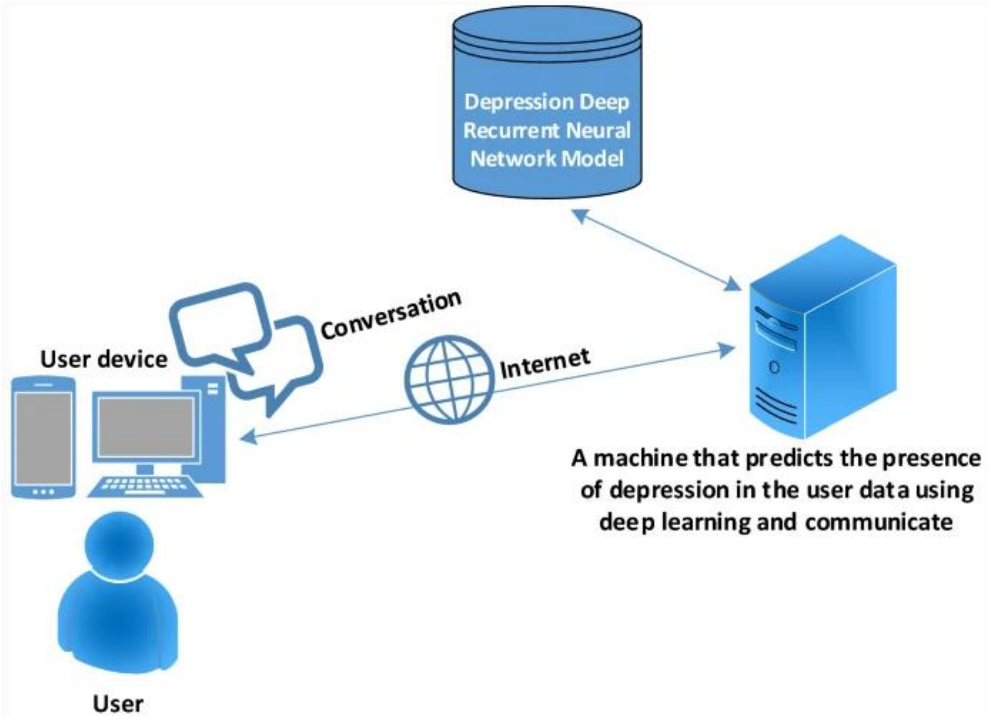
- Moodkit: It is an application that assists clients with diminishing stress of client utilizing CBT treatment. This application attempt to perceive and change negative considered client. It similar as an every day suspected record.
- Pacifica: It is an application which assists with making due stress dependent on CBT Therapy. It incorporates reflection, unwinding, mind-set and wellbeing following instruments.
- Wysa: It is a sincerely savvy chatbot that assist clients with dealing with their feelings and contemplations in light of CBT, DBT Therapy with contemplation practice and inspirational talking.

Every one of these applications are not intended to recommend legitimate treatment. They are only the initial step of perceive and to oversee gloom. This application is by all accounts attainable arrangement able to do dealing with uneasiness and wretchedness the executives upto a few limit.

CHAPTER-3

PROJECT DESIGN

Emotional situations can be represented as chronological words in text data while communicating with others. Therefore, a machine learning model capable of encoding consecutive data coding is ideal for such type of work. Therefore, Recurrent Neural Networks (RNNs) were adopted for this project.



3.1) LEARNING ABOUT LSTM RNN MODEL

RNN can be considered as the mostwell-known deep learning methods used to model time sequence information. RNNs basically contain a recurring link between history to present the state and the hidden circumstances. That is the most important memory role in neural networks. Conventional RNN algorithms often deal with the problem of extinct gradient, the limit of long-term data processing known as Long-Term Dependence. To

overcome the problem, Long-Term Short Memory (LSTM) was developed. Figure 1 shows a sample of a deep neural network consisting of 50 LSTM units.

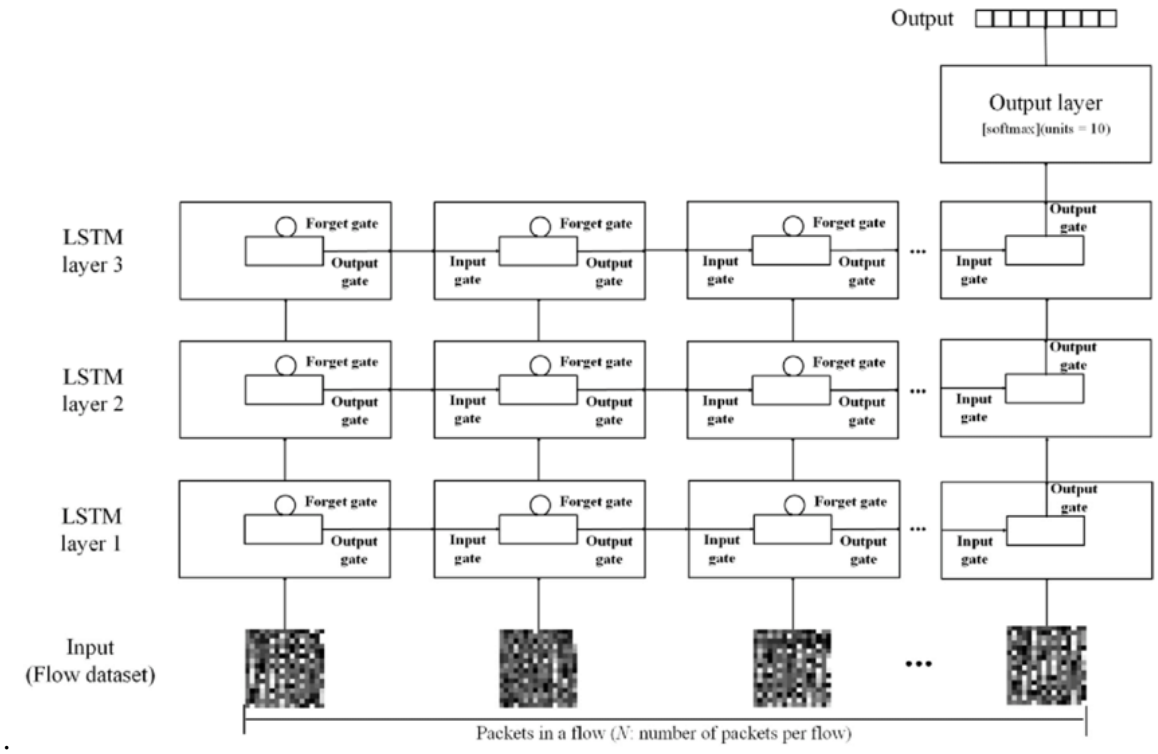


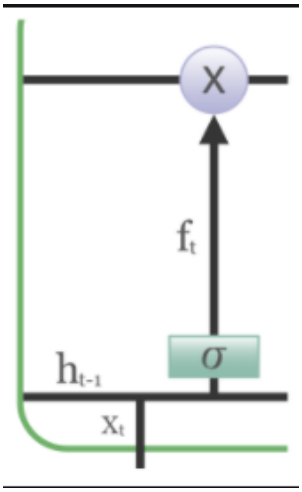
Fig.1 LSTM MODEL

LSTM networks are an extension of recurrent neural networks (RNNs) mainly introduced to handle situations where RNNs fail. Talking about RNN, it is a network that works on the present input by taking into consideration the previous output (feedback) and storing in its memory for a short period of time (short-term memory). Out of its various applications, the most popular ones are in the fields of speech processing, non-Markovian control, and music composition. Nevertheless, there are drawbacks to RNNs. First, it fails to store information for a longer period of time. At times, a reference to certain information stored quite a long time ago is required to predict the current output. But RNNs are absolutely incapable of handling such “long-term dependencies”. Second, there is no finer control over which part of the context needs to be carried forward and how much of the past needs to be ‘forgotten’. Other issues with RNNs are exploding and vanishing gradients (explained later) which occur during the training process of a

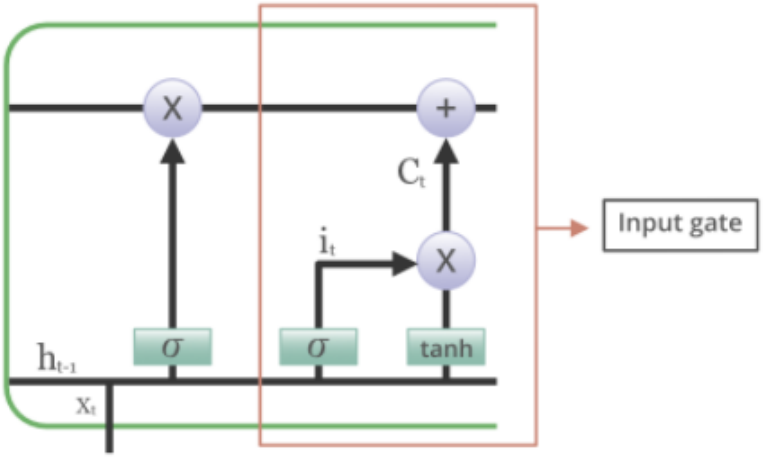
network through backtracking. Thus, Long Short-Term Memory (LSTM) was brought into the picture. It has been so designed that the vanishing gradient problem is almost completely removed, while the training model is left unaltered. Long time lags in certain problems are bridged using LSTMs where they also handle noise, distributed representations, and continuous values. With LSTMs, there is no need to keep a finite number of states from beforehand as required in the hidden Markov model (HMM). LSTMs provide us with a large range of parameters such as learning rates, and input and output biases. Hence, no need for fine adjustments. The complexity to update each weight is reduced to O(1) with LSTMs, similar to that of Back Propagation Through Time (BPTT), which is an advantage. Information is retained by the cells and the memory manipulations are done by the gates. There are three gates –

3.1.1) LSTM STRUCTURE

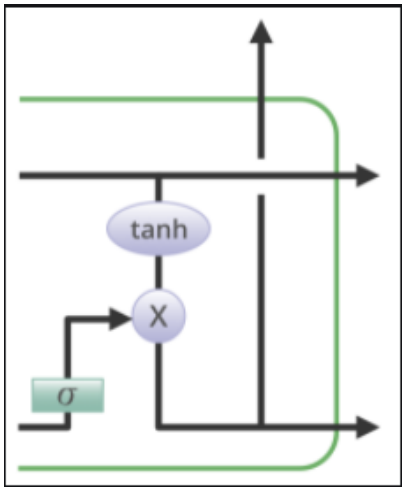
1. **Forget Gate**: The information that is no longer useful in the cell state is removed with the forget gate. Two inputs x_t (input at the particular time) and h_t-1 (previous cell output) are fed to the gate and multiplied with weight matrices followed by the addition of bias. The resultant is passed through an activation function which gives a binary output. If for a particular cell state the output is 0, the piece of information is forgotten and for output 1, the information is retained for future use.



2. **Input gate:** The addition of useful information to the cell state is done by the input gate. First, the information is regulated using the sigmoid function and filter the values to be remembered similar to the forget gate using inputs h_t-1 and x_t. Then, a vector is created using tanh function that gives an output from -1 to +1, which contains all the possible values from h_t-1 and x_t. At last, the values of the vector and the regulated values are multiplied to obtain the useful information.



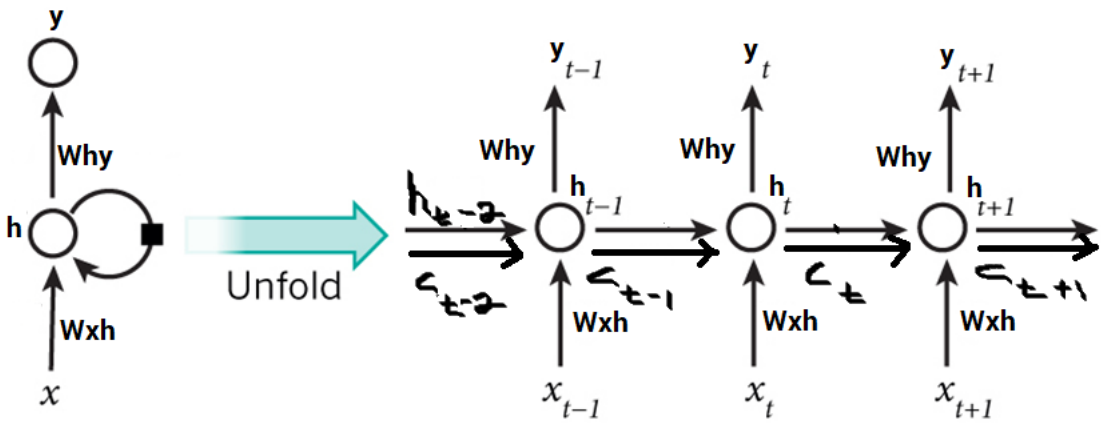
3. **Output gate:** The task of extracting useful information from the current cell state to be presented as output is done by the output gate. First, a vector is generated by applying tanh function on the cell. Then, the information is regulated using the sigmoid function and filter by the values to be remembered using inputs h_t-1 and x_t. At last, the values of the vector and the regulated values are multiplied to be sent as an output and input to the next cell.



3.2) ARCHITECTURE OF LSTM

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'.

Each LSTM cell has three inputs h_{t-1} ,C_{t-1} and x_t and two outputs h_t and C_t . For a given time t, h_t is the hidden state, C_t is the cell state or memory, x_t is the current data point or input. The first sigmoid layer has two inputs–h_{t-1} and x_t 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} .



Each LSTM memory block has a cell state as well as three gates, which are input, forget, and the output gates. The input gate Ft can be represented as

$$It=\beta(WLILt+WHLHt-1+bI) \quad (1)$$

where W is weight matrix, b bias vectors, and β a logistic function. The forget gate F can be expressed as

$$Ft=\beta(WLFLt+WHFHt-1+bF). \quad (2)$$

The long-term memory is stored in a cell state vector S that is expressed as

$$St=FtSt-1+Fttanh(WLSLt+WHSHt-1+bS). \quad (3)$$

The output gate V produces the output for the unit and can be expressed as

$$Vt=\beta(WLVLt+WHVHt-1+bV). \quad (4)$$

The hidden state H is expressed as

$$Ht=Vttanh(St). \quad (5)$$

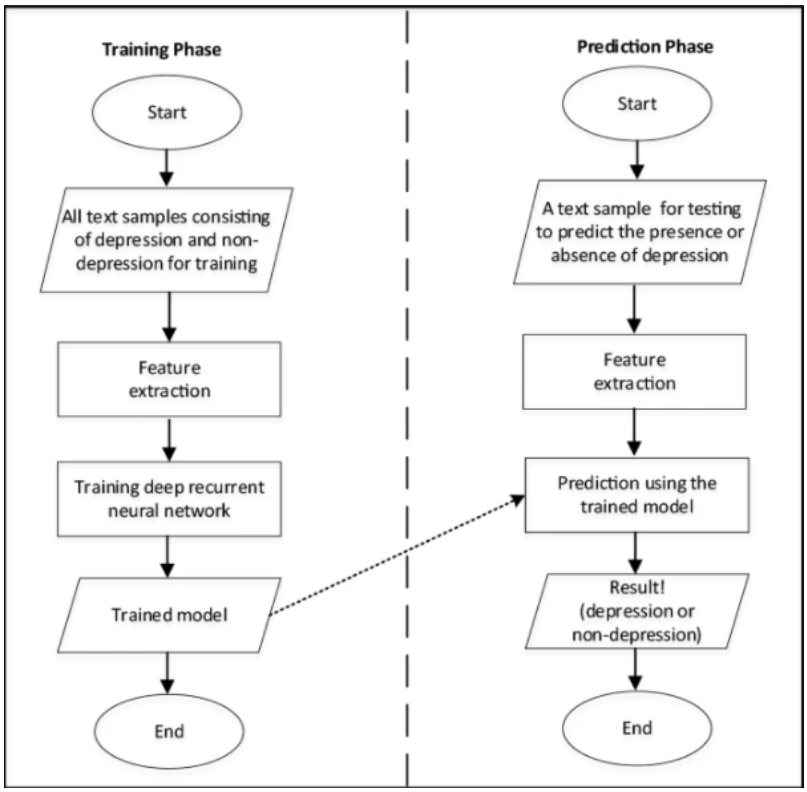
We adopt an attention layer over the LSTM units before applying dense layer as

$$A(att)t=LSTM(Ht,A(att)t-1) \quad (6)$$

The attention technique is basically used for emphasising important information in the current task rather than other useless information. Hence, it can be applied on top of the LSTM layers to improve the model's accuracy. Finally, the output can be determined using a softmax function as

$$O=\text{softmax}(WOAO+bO) \quad (7)$$

where W and b represent weights and bias, respectively. Figures 11 and 12 show the algorithms for training and prediction of depression or non-depression through RNN, respectively.

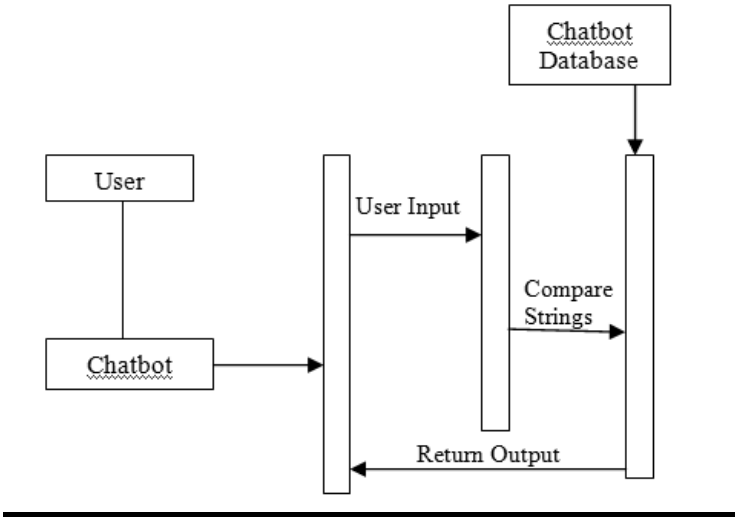


DATASET - For separating on the web sites information web scratching instrument, excellent soup is utilized. Additionally twitter's feeling examination information is considered to investigate the most continuous words utilized by discouraged individuals. The extricated information put away into yml documents for additional pre-handling information step which incorporates changing over text into lowercase and eliminate accentuation marks.

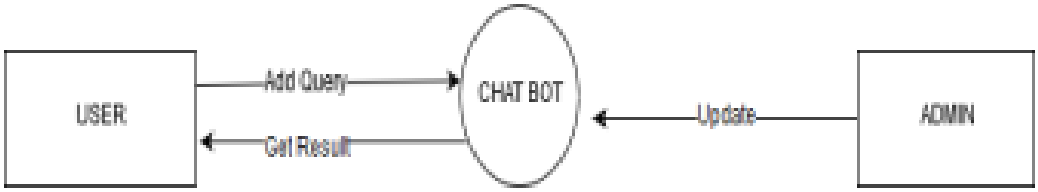
TRAINING AND TESTING - For the preparation and testing part dataset is partitioned into two section. To remove the feeling from text, profound learning model, LSTM a fake repetitive neural organization is helpful for this interaction.

MODEL - Seq2Seq LSTM (Long transient memory) a fake intermittent neural organization model learns long conditions and turn out incredibly for consecutive information. LSTM is fundamentally an answer for resolve the disappearing inclination issue of RNN throughout different time steps

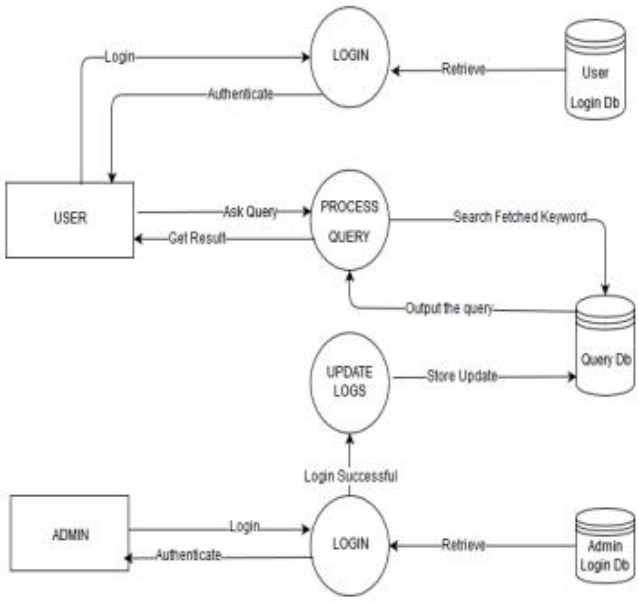
3.3) SEQUENTIAL DIAGRAM -



3.4) DATA FLOW DIAGRAM (LEVEL-0) –



DATA FLOW DIAGRAM (LEVEL-1) –



CHAPTER-4

SOURCE CODE

```
import os
import nltk
import numpy as np
import json

# For making a precision, recall report and confusion matrix on the classes
from sklearn.metrics import classification_report, confusion_matrix

import matplotlib.pyplot as plt
from nltk.stem.lancaster import LancasterStemmer
from string import punctuation
anger_training_set = []
fear_training_set = []
sadness_training_set = []
joy_training_set = []

anger_test_set = []
fear_test_set = []
sadness_test_set = []
joy_test_set = []
stemmer = LancasterStemmer()
all_words=[]

# Here I am loading the dataset from stored folder. The training data is stored as
text file and each tweet is accompanied

# by the magnitude of its sentiment (0 to 1). I had to go through the tweets myself
and observed that a threshold of 0.5 is
```

 # good enough to classify a tweet according to its sentiment. Tweets with lesser threshold were not definitive to be trained as per their mentioned classification

 # I only read those tweets that have a dominant classification factor i.e. above 0.5

 # Here i am setting each tweet's threshold magnitude accordingly

```
def load_training_data(sentiment):
    data = open("ML Project/datasets/"+sentiment+"_training_set.txt",encoding="utf8")
    if sentiment == "anger":
        threshold = 0.5
    elif sentiment == "fear":
        threshold = 0.6
    elif sentiment == "sadness":
        threshold = 0.5
    elif sentiment == "joy":
        threshold = 0.5
    else:
        pass
    return data,threshold


def load_test_data(sentiment):
    data = open("ML Project/datasets/"+sentiment+"_test_set.txt",encoding="utf8")
    return data
```

 # In this method, I am cleaning the tweet data removing punctuations and then tokenizing the words in tweet removing name tags
 # and appending them to training set

```
def clean_data(training_data,threshold):
    training_set = []
    for line in training_data:
        line = line.strip().lower()
        if line.split()[-1] == "none":
```

```
            line = " ".join(filter(lambda x:x[0]!='@', line.split()))
            punct = line.maketrans("","",'.*%$^0123456789#!][\?&/)/(+-<>')
            result = line.translate(punct)
            tokened_sentence = nltk.word_tokenize(result)
            sentence = tokened_sentence[0:len(tokened_sentence)-1]
            label = tokened_sentence[-2]
            training_set.append((sentence,label))
        else:
            intensity = float(line.split()[-1])
            if (intensity>=threshold):
                line = " ".join(filter(lambda x:x[0]!='@', line.split()))
                punct = line.maketrans("","",'.*%$^0123456789#!][\?&/)/(+-<>')
                result = line.translate(punct)
                tokened_sentence = nltk.word_tokenize(result)
                sentence = tokened_sentence[0:len(tokened_sentence)-1]
                label = tokened_sentence[-1]
                training_set.append((sentence,label))
    return training_set
```

 # This method collects all the unique words that are contained in the entire tweet dataset, finds their stem and
 # encodes each sentence according to the bag of words appending it to training set

```
def bag_of_words(all_data):
    training_set = []
    all_words = []
    for each_list in all_data:
        for words in each_list[0]:
            word = stemmer.stem(words)
            all_words.append(word)
    all_words = list(set(all_words))

    for each_sentence in all_data:
        bag = [0]*len(all_words)
        training_set.append(encode_sentence(all_words,each_sentence[0],bag))
```

```
    return training_set,all_words
# Here we encode each tweet's words according to the words it contained from the bag of words which is based on all words in all tweets
def encode_sentence(all_words,sentence, bag):
    for s in sentence:
        stemmed_word = stemmer.stem(s)
        for i,word in enumerate(all_words):
            if stemmed_word == word:
                bag[i] = 1
    return bag


def main():
    bag = []
    all_data = []
    all_test_data = []
    labels = []
    classes = []
    labels = []
    test_labels = []
    words=[]
    test_words = []
```

######### **Here we read the whole training data for each class and the threshold we will use for its classification**

```
    anger_training_data,threshold = load_training_data("anger")
    anger_training_set = clean_data(anger_training_data,threshold)
    print(anger_training_set[0])

    fear_training_data,threshold = load_training_data("fear")
    fear_training_set = clean_data(fear_training_data,threshold)

    sadness_training_data,threshold = load_training_data("sadness")
    sadness_training_set = clean_data(sadness_training_data,threshold)
```

```
    joy_training_data,threshold = load_training_data("joy")
    joy_training_set = clean_data(joy_training_data,threshold)
```

######### Here we read the whole test data for each class and the threshold we will use for its classification

```
    anger_test_data = load_test_data("anger")
    anger_test_set = clean_data(anger_test_data,threshold)
    #print(anger_test_set[0])
    print(len(anger_test_set))

    fear_test_data = load_test_data("fear")
    fear_test_set = clean_data(fear_test_data,threshold)
   # print(fear_test_set[0])
    print(len(fear_test_set))

    sadness_test_data = load_test_data("sadness")
    sadness_test_set = clean_data(sadness_test_data,threshold)
  #  print(sadness_test_set[0])
    print(len(sadness_test_set))

    joy_test_data = load_test_data("joy")
    joy_test_set = clean_data(joy_test_data,threshold)
  #  print(joy_test_set[0])
    print(len(joy_test_set))
```

In every training set above we have a nested list whose first element is sentence and 2nd element its respective label

```
#    print(anger_training_set[0][0],anger_training_set[0][1])
#    print(joy_training_set[0][0],joy_training_set[0][1])
```

Here we combine all training sets in one list

```
    all_data.extend(anger_training_set)
    all_data.extend(fear_training_set)
    all_data.extend(sadness_training_set)
    all_data.extend(joy_training_set)

    all_data.extend(anger_test_set)
    all_data.extend(fear_test_set)
    all_data.extend(sadness_test_set)
    all_data.extend(joy_test_set)
```

Here we simply make a classification label list encoding our 4 classes as follows

```
    for i,j in all_data:
        if j == "anger":
            labels.append([1,0,0,0])
        elif j == "fear":
            labels.append([0,1,0,0])
        elif j == "sadness":
            labels.append([0,0,1,0])
        elif j == "joy":
            labels.append([0,0,0,1])
        else:
            pass

    print(len(labels))
print(len(test_labels))
    classes = ["anger","fear","sadness","joy"]
    print(classes)
    np.set_printoptions(threshold=np.inf)
```

 # Here we will have the whole training set and the all the words contained in whole training set

```
    training_set,words = bag_of_words(all_data)
```

 # We convert our training,test set and training, test labels in a numpy array as it is required for calculations in neural net

```
    dataset = np.array(training_set)
    labels = np.array(labels)
```

**   # It is important to shuffle dataset so your classifier does not attempt to memorize training set, this functions shuffles data and labels.**

```
    shuffling_function = np.random.permutation(dataset.shape[0])
    shuffled_dataset, shuffled_labels = np.zeros((dataset.shape)),np.zeros((dataset.shape))
    shuffled_dataset,shuffled_labels =
dataset[shuffling_function],labels[shuffling_function]


    split = int(len(shuffled_dataset)*0.8)
    training_data = shuffled_dataset[:split]
    training_labels = shuffled_labels[:split]
    test_data = shuffled_dataset[split:]
    test_labels = shuffled_labels[split:]
    print(training_data.shape)
    print(training_labels.shape)
    print(test_data.shape)
    print(test_labels.shape)
```

############ HERE WE HAVE A SHUFFLED DATASET WITH RESPECTIVE LABELS NOW WE HAVE TO TRAIN THIS DATA BOTH NUMPY ARRAYS ############

```
    Train_model(training_data,training_labels,words,classes)
    Test_model(test_data,test_labels,words,classes)
```

 # Method for calculating sigmoid

```
def sigmoid(z):
    return (1/(1+np.exp(-z)))
```

 # Method for calculating relu

```
def relu(z):
    A = np.array(z,copy=True)
    A[z<0]=0
    assert A.shape == z.shape
    return A
```

 # Method for calculating softmax

```
def softmax(x):
    num = np.exp(x-np.amax(x,axis=0,keepdims=True))
    return num/np.sum(num,axis=0,keepdims=True)
```

 # Method for calculating forward propagation

```
def forward_prop(n_x,n_h,n_y,m,X,W1,W2,b1,b2):
```

 # Forward propagate data ... dimensions should be 100x1547

```
    Z1 = np.dot(W1,X)+b1
    A1 = relu(Z1)
    Z2 = np.dot(W2,A1)+b2
    A2 = softmax(Z2)
    return Z1,A1,Z2,A2
```

 # Method for calculating relu activation's derivative

```
def relu_backward(da,dz):
    da1 = np.array(da,copy=True)
    da1[dz<0]=0
    assert da1.shape == dz.shape
    return da1
```

 # Method for calculating linear part of backward propagation

```
def linear_backward(dz,a,m,w,b):
    dw = (1/m)*np.dot(dz,a.T)
    db = (1/m)*np.sum(dz,axis=1,keepdims=True)
    da = np.dot(w.T,dz)
    assert (dw.shape==w.shape)
    assert (da.shape==a.shape)
    assert (db.shape == b.shape)
    return da,dw,db
```

 # Method for calculating loss function

```
def calculate_loss(Y,Yhat,m):
    loss = (-1/m)*np.sum(np.multiply(Y,np.log(Yhat)))
    return loss
```

 # Method for back propagation

```
def back_prop(Z1,A1,Z2,A2,X,Y,W1,W2,b1,b2,learning_rate,m):
    dZ2 = A2-Y
    da1,dw2,db2 = linear_backward(dZ2,A1,m,W2,b2)
    dZ1 = relu_backward(da1,Z1)
    da0,dw1,db1 = linear_backward(dZ1,X,m,W1,b1)
    W2 = W2 - learning_rate * dw2
    b2 = b2 - learning_rate * db2
    W1 = W1 - learning_rate * dw1
    b1 = b1 - learning_rate * db1
    return W1,b1,W2,b2
```

 # Method for training model

```
def Test_model(test_data, test_labels,words,classes):
    all_losses = []
    learning_rate = 0.1
    iterations = 50
    np.random.seed(1)
    X = test_data.T
    print(" Shape of X is ", X.shape)
    Y = test_labels.T
    print(" Shape of Y is ", Y.shape)
    # m is total number of training examples
    m = X.shape[1]
    print(" Shape of m is ", m)

    # Number of hidden layer neurons
    n_h = 100
    # Number of training points
    n_x = X.shape[0]
    # Number of output neurons because we have 4 classes
    n_y = 4

    weights_file = 'weights.json'
    with open(weights_file) as data_file:
        weights = json.load(data_file)
        W1 = np.asarray(weights['weight1'])
        W2 = np.asarray(weights['weight2'])
        b1 = np.asarray(weights['bias1'])
        b2 = np.asarray(weights['bias2'])

    print("#################### TEST MODEL STATISTICS
######################")
```

```
    for i in range(1):
        # input layer is our encoded sentence
        l0 = X
        # matrix multiplication of input and hidden layer
        l1 = relu(np.dot(W1,l0)+b1)
        # output layer
        l2 = softmax(np.dot(W2,l1)+b2)
        predictions = np.argmax(l2, axis=0)
        labels = np.argmax(Y, axis=0)
        print(classification_report(predictions,labels))

# Method for training model
def Train_model(training_data, training_labels,words,classes):
    all_losses = []
    learning_rate = 0.1
    iterations = 50
    np.random.seed(1)
    X = training_data.T
    print(" Shape of X is ", X.shape)
    Y = training_labels.T
    print(" Shape of Y is ", Y.shape)
    # m is total number of training examples
    m = X.shape[1]
    print(" Shape of m is ", m)
    # Number of hidden layer neurons
    n_h = 100
    # Number of training points
    n_x = X.shape[0]
    # Number of output neurons because we have 4 classes
    n_y = 4
    # Multiplying by 0.01 so that we get smaller weights .. dimensions 100x3787
    W1 = np.random.randn(n_h,n_x)*0.01
    print(" Shape of W1 is ", W1.shape)
    # Dimensions 100x1
    b1 = np.zeros((n_h,1))
```

```
    # Dimensions 1547 x 4
    W2 = np.random.randn(n_y,n_h)
    print(" Shape of W2 is ", W2.shape)
    # Forward propagate data ... dimensions should be 100x1547
    b2 = np.zeros((n_y,1))
    print("#################### TRAIN MODEL STATISTICS
######################")
    for i in range(0,iterations):
        Z1,A1,Z2,A2 = forward_prop(n_x,n_h,n_y,m,X,W1,W2,b1,b2)
        predictions = np.argmax(A2, axis=0)
        labels = np.argmax(Y, axis=0)
        print(classification_report(predictions,labels))
        Loss = calculate_loss(Y,A2,m)
        W1,b1,W2,b2 = back_prop(Z1,A1,Z2,A2,X,Y,W1,W2,b1,b2,learning_rate,m)
        all_losses.append(Loss)

    # storing weights so that we can reuse them without having to retrain the neural
network

    weights = {'weight1': W1.tolist(), 'weight2': W2.tolist(),
               'bias1':b1.tolist(), 'bias2':b2.tolist(),
               'words': words,
               'classes': classes
              }
    weights_file = "weights.json"



    with open(weights_file, 'w') as outfile:
        json.dump(weights, outfile, indent=4, sort_keys=True)
    print ("saved synapses to:", weights_file)
    plt.plot(all_losses)


if __name__ == '__main__':
    main()
```

```
# probability threshold

ERROR_THRESHOLD = 0.1
# load our calculated weight values

weights_file = 'weights.json'
with open(weights_file) as data_file:
    weights = json.load(data_file)
    W1 = np.asarray(weights['weight1'])
    W2 = np.asarray(weights['weight2'])
    b1 = np.asarray(weights['bias1'])
    b2 = np.asarray(weights['bias2'])
    all_words = weights['words']
    classes = weights['classes']

def clean_sentence(verification_data):
    line = verification_data

    # Remove whitespace from line and lower case iter
    line = line.strip().lower()

    # Removing word with @ sign as we dont need name tags of twitter
    line = " ".join(filter(lambda x:x[0]!='@', line.split()))

    # Remove punctuations and numbers from the line
    punct = line.maketrans("","",'.*%$^0123456789#!][\?&/)/(+-<>')
    result = line.translate(punct)

    # Tokenize the whole tweet sentence
    tokened_sentence = nltk.word_tokenize(result)

    # We take the tweet sentence from tokened sentence
    sentence = tokened_sentence[0:len(tokened_sentence)]
    return sentence
```

```
def verify(sentence, show_details=False):
    bag=[0]*len(all_words)
    cleaned_sentence = clean_sentence(sentence)

    # This line returns the bag of words as 0 or 1 if words in sentence are found in
all_words
    x = encode_sentence(all_words,cleaned_sentence,bag)
    x = np.array(x)
    x = x.reshape(x.shape[0],1)

#    print("Shape of X is ", x.shape)
    if show_details:
        print ("sentence:", sentence, "\n bow:", x)
    # input layer is our encoded sentence
    l0 = x
    # matrix multiplication of input and hidden layer
    l1 = relu(np.dot(W1,l0)+b1)
    # output layer
    l2 = softmax(np.dot(W2,l1)+b2)

    return l2

def classify(sentence, show_details=False):
    results = verify(sentence, show_details)
    results = [[i,r] for i,r in enumerate(results) if r>ERROR_THRESHOLD ]
    results.sort(key=lambda x: x[1], reverse=True)
    return_results =[[classes[r[0]],r[1]] for r in results]
    print ("%s \n classification: %s \n" % (sentence, return_results))
    return return_results

classify("I want to kill everyone @Name1 #why?")
classify("I am so happy @Name2 #yayyyy")
classify("This depression will kill me someday .. i am dying @Name3 #killme")
classify("I am afraid terrorists might attack us @Name4 #isis")
```

```
classify("What should I do when i am happy @Name5 ")
classify("I want to be happy")

from IPython.display import clear_output
input_sentiment = input("Hi :) How are you feeling today ? ")
#print(input_sentiment)
#print(classify(input_sentiment)[0][0])
sentiment = classify(input_sentiment)[0][0]
print(sentiment)
if sentiment == "anger" or sentiment == "sadness" or sentiment == "fear":
    answer = input("Sorry to hear that .... would you like to hear a joke to lighten your
mood ? Press Yes or No ")
    if answer == "N" or answer == "No" or answer == "no" or answer == "n":
        print("Have a nice day. Goodbye :) ")
    else:
        file = open('C:/Users/Shaya/Desktop/jokes.txt','r')
        while(1):
            full_file = file.readline()
            split_file = full_file.split('/')
        #    print(split_file)
            slashes = full_file.count('/')
        #    print(slashes)
            line_of_joke = []
            for i in range(slashes):
                k=0
            #    print(split_file[i])
                commas = split_file[i].count("")
        #        print(commas)
                length = int(commas/2)
                if length == 0:
                    line_of_joke.append(split_file[i])
                else:
                    for j in range(length):
        #            print("Here")
                        line_of_joke.append(split_file[i].split("")[k]+split_file[i].split("")[k+1])
```

```
                if j==length-1:
                    line_of_joke.append(split_file[i].split("")[k+2])
                k=k+2
    #    break
        for i in line_of_joke:
            print(i)
        user_input = input("Do you want another joke ? Write Yes or No\t")
        if user_input == "Y":
            clear_output()
            pass
        else:
            clear_output()
            break
       #print(line_of_joke[1])
```

Now we also have to deploy our created model with the help of flask. So here is the code,

APP.PY

```
from flask import Flask, request, render_template, jsonify
app = Flask(__name__)

import collections

import numpy as np

from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Model
from keras.layers import GRU, Input, Dense, TimeDistributed, Activation, RepeatVector,
Bidirectional
from keras.layers import Embedding, CuDNNLSTM, GlobalMaxPooling1D,
GlobalAveragePooling1D, CuDNNGRU
from keras.layers.embeddings import Embedding
from keras.optimizers import Adam
from keras.losses import sparse_categorical_crossentropy
from keras.utils.vis_utils import plot_model
from keras.callbacks import ModelCheckpoint, EarlyStopping, TensorBoard
from nltk.translate.bleu_score import sentence_bleu, corpus_bleu
import helper
import tensorflow as tf
from tensorflow.python.layers.core import Dense
from tensorflow.python.ops.rnn_cell_impl import _zero_state_tensors
from tensorflow.contrib.seq2seq import AttentionWrapper as attention_wrapper
from tensorflow.contrib.seq2seq import BeamSearchDecoder as beam_search_decoder
# print('TensorFlow Version: {}'.format(tf.__version__))
import json
import pandas as pd
import matplotlib.pyplot as plt
from tqdm import tqdm
tqdm.pandas()
```

```
import re
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)


contraction_mapping = {"ain't": "is not", "aren't": "are not","can't": "cannot", "'cause": "because", "could've": "could have", "couldn't": "could not", "didn't": "did not", "doesn't": "does not", "don't": "do not", "hadn't": "had not", "hasn't": "has not", "haven't": "have not", "he'd": "he would","he'll": "he will", "he's": "he is", "how'd": "how did", "how'd'y": "how do you", "how'll": "how will", "how's": "how is",  "I'd": "I would", "I'd've": "I would have", "I'll": "I will", "I'll've": "I will have","I'm": "I am", "I've": "I have", "i'd": "i would", "i'd've": "i would have", "i'll": "i will",  "i'll've": "i will have","i'm": "i am", "i've": "i have", "isn't": "is not", "it'd": "it would", "it'd've": "it would have", "it'll": "it will", "it'll've": "it will have","it's": "it is", "let's": "let us", "ma'am": "madam", "mayn't": "may not", "might've": "might have","mightn't": "might not","mightn't've": "might not have", "must've": "must have", "mustn't": "must not", "mustn't've": "must not have", "needn't": "need not", "needn't've": "need not have","o'clock": "of the clock", "oughtn't": "ought not", "oughtn't've": "ought not have", "shan't": "shall not", "sha'n't": "shall not", "shan't've": "shall not have", "she'd": "she would", "she'd've": "she would have", "she'll": "she will", "she'll've": "she will have", "she's": "she is", "should've": "should have", "shouldn't": "should not", "shouldn't've": "should not have", "so've": "so have","so's": "so as", "this's": "this is","that'd": "that would", "that'd've": "that would have", "that's": "that is", "there'd": "there would", "there'd've": "there would have", "there's": "there is", "here's": "here is","they'd": "they would", "they'd've": "they would have", "they'll": "they will", "they'll've": "they will have", "they're": "they are", "they've": "they have", "to've": "to have", "wasn't": "was not", "we'd": "we would", "we'd've": "we would have", "we'll": "we will", "we'll've": "we will have", "we're": "we are", "we've": "we have", "weren't": "were not", "what'll": "what will", "what'll've": "what will have", "what're": "what are",  "what's": "what is", "what've": "what have", "when's": "when is", "when've": "when have", "where'd": "where did", "where's": "where is", "where've": "where have", "who'll": "who will", "who'll've": "who will have", "who's": "who is", "who've": "who have", "why's": "why is", "why've": "why have", "will've": "will have", "won't": "will not", "won't've": "will not have",
```

```
"would've": "would have", "wouldn't": "would not", "wouldn't've": "would not have",
"y'all": "you all", "y'all'd": "you all would","y'all'd've": "you all would have","y'all're":
"you all are","y'all've": "you all have","you'd": "you would", "you'd've": "you would
have", "you'll": "you will", "you'll've": "you will have", "you're": "you are", "you've":
"you have", 'u.s':'america', 'e.g':'for example'}
def clean_contractions(text, mapping):
    specials = ["’", "‘", "´", "`"]
    for s in specials:
        text = text.replace(s, "'")
    text = ' '.join([mapping[t] if t in mapping else t for t in text.split(" ")])
    return text

punct = [',', '.', '"', ':', ')', '(', '-', '!', '?', '|', ';', "'", '$', '&', '/', '[', ']', '>', '%', '=', '#', '*', '+', '\\', '•',
'~', '@', '£',
 '·', '_', '{', '}', '©', '^', '®', '`',  '<', '→', '°', '€', '™', '›',  '♥', '←', '×', '§', '″', '′', 'Â', '█', '½', 'à',
'…',
 '“', '★', '”', '–', '●', 'â', '►', '−', '¢', '²', '¬', '░', '¶', '↑', '±', '¿', '▾', '═', '¦', '║', '—', '¥', '▓', '—',
'‹', '─',
 '▒', ' : ', '¼', '⊕', '▼', '▪', '†', '■', '’', '▀', '¨', '▄', '♫', '☆', 'é', '¯', '♦', '¤', '▲', 'è', ',', '¾', 'Ã', '·',
'‘', '∞',
 '·', ')  ', '↓', '、', '│', '  (', '»', ',   ', '♪', '╩', '╚', '³', ' ▪ ', '╤', '╣', '╔', '╗', '▬', '❤', 'ï', 'Ø', '¹', '≤',
'‡', '√', ]

punct_mapping = {"‘": "'", "₹": "e", "´": "'", "°": "", "€": "e", "™": "tm", "√": " sqrt ", "×":
"x", "²": "2", "—": "-", "–": "-", "’": "'", "_": "-", "`": "'", '“': '', '”': '', '‘‘': '', "£": "e", '∞':
'infinity', 'θ': 'theta', '÷': '/', 'α': 'alpha', '•': '.', 'à': 'a', '−': '-', 'β': 'beta', 'Ø': '', '³': '3', 'π': 'pi', '!':'
'}

def  clean_special_chars(text, punct, mapping):
    for p in mapping:
        text = text.replace(p, mapping[p])

    for p in punct:
        text = text.replace(p, f' {p} ')
```

```
    specials = {'\u200b': ' ', '…': ' ... ', '\ufeff': '', 'करना': '', 'है': ''}
    for s in specials:
        text = text.replace(s, specials[s])

    return text


mispell_dict = {'colour': 'color', 'centre': 'center', 'favourite': 'favorite', 'travelling': 'traveling', 'counselling': 'counseling', 'theatre': 'theater', 'cancelled': 'canceled', 'labour': 'labor', 'organisation': 'organization', 'wwii': 'world war 2', 'citicise': 'criticize', 'youtu ': 'youtube ', 'Qoura': 'Quora', 'sallary': 'salary', 'Whta': 'What', 'narcisist': 'narcissist', 'howdo': 'how do', 'whatare': 'what are', 'howcan': 'how can', 'howmuch': 'how much', 'howmany': 'how many', 'whydo': 'why do', 'doI': 'do I', 'theBest': 'the best', 'howdoes': 'how does', 'mastrubation': 'masturbation', 'mastrubate': 'masturbate', "mastrubating": 'masturbating', 'pennis': 'penis', 'Etherium': 'Ethereum', 'narcissit': 'narcissist', 'bigdata': 'big data', '2k17': '2017', '2k18': '2018', 'qouta': 'quota', 'exboyfriend': 'ex boyfriend', 'airhostess': 'air hostess', "whst": 'what', 'watsapp': 'whatsapp', 'demonitisation': 'demonetization', 'demonitization': 'demonetization', 'demonetisation': 'demonetization'}
def correct_spelling(x, dic):
    for word in dic.keys():
        x = x.replace(word, dic[word])
    return x

def preprocess_sentence(w):

    # creating a space between a word and the punctuation following it
    # eg: "he is a boy." => "he is a boy ."
    # Reference:- https://stackoverflow.com/questions/3645931/python-padding-punctuation-with-white-spaces-keeping-punctuation
#     w = re.sub(r"([?.!,¿])", r" \1 ", w)
#     w = re.sub(r'[" "]+', " ", w)

    # replacing everything with space except (a-z, A-Z, ".", "?", "!", ",")
```

```
    w = re.sub(r"[^a-zA-Z?.!¿]+", " ", w)

    w = w.rstrip().strip()

    # adding a start and an end token to the sentence
    # so that the model know when to start and stop predicting.
#     w = '<start> ' + w + ' <end>'

    return w


# with open('indextosent.pkl', 'rb') as f:
#     idx2sent = pickle.load(f)

# with open('senttoindex.pkl', 'rb') as f:
#     sent2idx = pickle.load(f)

reverse_vocab = json.load( open( "idxtosent.json" ) )
vocab = json.load( open( "senttoindex.json" ) )


enc_sentence_length = 17
dec_sentence_length = 17

_START_ = "_START_"
_PAD_ = "_PAD_"
_END_ = "_END_"

def tokenizer(sentence):
    tokens = re.findall(r"[\w]+|[^\s\w]", str(sentence))
    return tokens

def token2idx(word, vocab):
    return vocab[word]
```

```
def sent2idx(sent, vocab=vocab, max_sentence_length=enc_sentence_length,
is_target=False):
    tokens = tokenizer(sent)
    current_length = len(tokens)
    pad_length = max_sentence_length - current_length
    if is_target:
        return [0] + [token2idx(token, vocab) for token in tokens] + [2] + [1] * (pad_length-
1), current_length + 1
    else:
        return [token2idx(token, vocab) for token in tokens] + [1] * pad_length,
current_length

def idx2token(idx, reverse_vocab):
    return reverse_vocab[idx]

def idx2sent(indices, reverse_vocab=reverse_vocab):
    return " ".join([idx2token(str(idx), reverse_vocab) for idx in indices])

# # Enc Example
# print('hi what is your name?')
# print(sent2idx('hi what is your name?'))

# # Dec Example
# print('hi this is chinmay.')
# print(sent2idx('hi this is chinmay.', max_sentence_length=dec_sentence_length,
is_target=True))

# print(idx2sent([0, 16, 41, 7, 36, 3, 2, 1, 1, 1, 1]))

from model import BasicS2SModel

def respond(inp):
    tf.reset_default_graph()
    # model = BasicS2SModel(vocab=vocab,num_layers=3)
```

```
    with tf.Session() as sess:
        model =
BasicS2SModel(vocab=vocab,mode="inference",use_beam_search=False,num_layers=3)
        saver = tf.train.Saver(tf.global_variables())
        saver.restore(sess,tf.train.latest_checkpoint('./test_s2s/'))
        inputs = ['hi what is your name', inp]


        batch_preds = []
        batch_tokens = []
        batch_sent_lens = []
        for input_sent in inputs:
            tokens, sent_len = sent2idx(input_sent)
            batch_tokens.append(tokens)
            batch_sent_lens.append(sent_len)

        batch_preds = model.inference(sess,batch_tokens,batch_sent_lens)
    #      print(batch_preds)

        response = []
        preds = batch_preds[0][1]
        for i in range(1,len(preds)-1):
            if preds[i] == preds[i+1]:
                response.append(preds[i])
                # preds[i+1:9] = 1
                # preds[9] = 2
                break
            else:
                response.append(preds[i])


        # for input_sent, pred in zip(input_batch,  batch_preds[0]):
        #     print('Input:', input_sent)
        #     print(pred)
        #     print('Prediction:', idx2sent(pred, reverse_vocab=reverse_vocab))
```

```
        #     print('Target:', target_sent)

        response = idx2sent(response, reverse_vocab=reverse_vocab)
        return response



def preprocessing(text):
    text = text.lower()
    text = clean_contractions(text,contraction_mapping)
    text = clean_special_chars(text, punct, punct_mapping)
    text = correct_spelling(text, mispell_dict)
    text = re.sub(r'^https?:\/\/.*[\r\n]*', '', text, flags=re.MULTILINE)
    text = re.sub(r'\/r\/[a-z]+ ', '', text, flags=re.MULTILINE)
    text = preprocess_sentence(text)
    return text


@app.route('/ask', methods=['POST'])
def index():
        text = request.form['messageText']
        if text=='exit':
            exit()
        else:
            sentence = preprocessing(text)
            bot_response = respond(sentence)
            # print(text)
            return jsonify(({'status':'OK','answer':bot_response}))

@app.route('/')
def my_form():
    return render_template('my-form.html')
if __name__ == '__main__':
  app.run(debug=True)
```

MODEL.PY

```
import collections

import numpy as np

from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Model
from keras.layers import GRU, Input, Dense, TimeDistributed, Activation, RepeatVector,
Bidirectional
from keras.layers import Embedding, CuDNNLSTM, GlobalMaxPooling1D,
GlobalAveragePooling1D, CuDNNGRU
from keras.layers.embeddings import Embedding
from keras.optimizers import Adam
from keras.losses import sparse_categorical_crossentropy
from keras.utils.vis_utils import plot_model
from keras.callbacks import ModelCheckpoint, EarlyStopping, TensorBoard
from nltk.translate.bleu_score import sentence_bleu, corpus_bleu
import helper
import tensorflow as tf
from tensorflow.python.layers.core import Dense
from tensorflow.python.ops.rnn_cell_impl import _zero_state_tensors
from tensorflow.contrib.seq2seq import AttentionWrapper as attention_wrapper
from tensorflow.contrib.seq2seq import BeamSearchDecoder as beam_search_decoder
# print('TensorFlow Version: {}'.format(tf.__version__))
import json
import pandas as pd
import matplotlib.pyplot as plt
from tqdm import tqdm
tqdm.pandas()
import re
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
```

```
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)

_START_ = "_START_"
_PAD_ = "_PAD_"
_END_ = "_END_"


def get_optimizer(opt):
    if opt == "adam":
        optfn = tf.train.AdamOptimizer
    elif opt == "sgd":
        optfn = tf.train.GradientDescentOptimizer
    else:
        assert(False)
    return optfn


def single_rnn_cell(cell_name,dim_size, train_phase = True, keep_prob = 0.75):
    if cell_name == "gru":
        cell = tf.contrib.rnn.GRUCell(dim_size)
    elif cell_name == "lstm":
        cell = tf.contrib.rnn.LSTMCell(dim_size)
    else:
        cell = tf.contrib.rnn.BasicRNNCell(dim_size)
    if train_phase and keep_prob < 1.0:
        cell = tf.contrib.rnn.DropoutWrapper(
            cell=cell,
            input_keep_prob=keep_prob,
            output_keep_prob=keep_prob)
    return cell

def multi_rnn_cell(cell_name,dim_size,num_layers = 1, train_phase = True,
keep_prob=0.75):
    cells = []
```

```
    for _ in range(num_layers):
        cell = single_rnn_cell(cell_name,dim_size, train_phase, keep_prob)
        cells.append(cell)

    if len(cells) > 1:
        final_cell = tf.contrib.rnn.MultiRNNCell(cells=cells)
    else:
        final_cell = cells[0]
    return final_cell

class BasicS2SModel(object):
    def __init__(self, vocab, batch_size = 2, dim_size=128, rnn_cell = 'gru', num_layers=2,
max_gradient_norm=5.0, atten_size=30,
                 learning_rate=0.001, learning_rate_decay_factor=0.98,
dropout=0.2,max_inference_lenght=10,
                 max_source_len = 10, max_target_len = 10,beam_size =3, optimizer="adam",
mode ='train',
                 use_beam_search = False):
        assert mode in ['train', 'inference']
        self.start_token = vocab.get(_START_)
        self.end_token = vocab.get(_END_)
        self.train_phase = True if mode == 'train' else False
        self.cell_name = rnn_cell
        self.dim_size = dim_size
        self.vocab_size = len(vocab)
        self.num_layers = num_layers
        self.keep_prob_config = 1.0 - dropout
        self.atten_size = atten_size

        # decoder
        self.max_inference_lenght = max_inference_lenght

        # beam search
        self.beam_size = beam_size
        self.beam_search = use_beam_search
```

```
        # learning
        self.learning_rate = tf.Variable(float(learning_rate), trainable=False)
        self.learning_rate_decay_op = self.learning_rate.assign(self.learning_rate *
learning_rate_decay_factor)
        self.global_step = tf.Variable(0, trainable=False)

        # if we use beam search decoder, we need to specify the batch size and max
source len
        if self.beam_search:
            self.batch_size = batch_size
            self.source_tokens = tf.placeholder(tf.int32, shape=[batch_size, max_source_len])
            self.source_length = tf.placeholder(tf.int32, shape=[batch_size,])
        else:
            self.source_tokens = tf.placeholder(tf.int32,shape=[None,None])
            self.source_length = tf.placeholder(tf.int32,shape=[None,])

        if self.train_phase:
            self.target_tokens = tf.placeholder(tf.int32, shape=[None, None])
            self.target_length = tf.placeholder(tf.int32, shape=[None,])

        with tf.variable_scope("S2S",initializer = tf.uniform_unit_scaling_initializer(1.0)):
            self.setup_embeddings()
            #self.setup_encoder()
            self.setup_bidirection_encoder()
            self.setup_attention_decoder()

        if self.train_phase:
            opt = get_optimizer(optimizer)(self.learning_rate)
            params = tf.trainable_variables()
            gradients = tf.gradients(self.losses, params)
            clipped_gradients, _ = tf.clip_by_global_norm(gradients, max_gradient_norm)
            self.gradient_norm = tf.global_norm(gradients)
            self.param_norm = tf.global_norm(params)
```

```
            self.updates = opt.apply_gradients(zip(clipped_gradients, params),
global_step=self.global_step)

    def setup_embeddings(self):
        with tf.variable_scope("Embeddings"):
            with tf.device('/cpu:0'):
                self.enc_emd = tf.get_variable("encode_embedding", [self.vocab_size,
self.dim_size])
                self.dec_emd = tf.get_variable("decode_embedding", [self.vocab_size,
self.dim_size])
                self.encoder_inputs = tf.nn.embedding_lookup(self.enc_emd,
self.source_tokens)
                if self.train_phase:
                    self.decoder_inputs = tf.nn.embedding_lookup(self.dec_emd,
self.target_tokens)

    def setup_encoder(self):
        cell = multi_rnn_cell(self.cell_name,self.dim_size, self.num_layers,
self.train_phase,self.keep_prob_config)
        outputs,state =
tf.nn.dynamic_rnn(cell,inputs=self.encoder_inputs,sequence_length=self.source_length,d
type=tf.float32)
        self.encode_output = outputs
        self.encode_state = state
        # using the state of last layer of rnn as initial state
        self.decode_initial_state = self.encode_state[-1]

    def setup_bidirection_encoder(self):
        fw_cell = single_rnn_cell('gru',self.dim_size, train_phase=self.train_phase,
keep_prob=self.keep_prob_config)
        bw_cell = single_rnn_cell('gru',self.dim_size, train_phase=self.train_phase,
keep_prob=self.keep_prob_config)

        with tf.variable_scope("Encoder"):
            outputs,states = tf.nn.bidirectional_dynamic_rnn(
```

```
            cell_fw = fw_cell,
            cell_bw = bw_cell,
            dtype = tf.float32,
            sequence_length = self.source_length,
            inputs = self.encoder_inputs
            )
        outputs_concat = tf.concat(outputs, 2)
    self.encode_output = outputs_concat
    self.encode_state = states

    # use Dense layer to convert bi-direction state to decoder inital state
    convert_layer = Dense(self.dim_size,dtype=tf.float32,name="bi_convert")
    self.decode_initial_state = convert_layer(tf.concat(self.encode_state,axis=1))

  def setup_training_decoder_layer(self):
    max_dec_len = tf.reduce_max(self.target_length, name='max_dec_len')
    training_helper =
tf.contrib.seq2seq.TrainingHelper(self.decoder_inputs,self.target_length,name="training_
helper")
    training_decoder = tf.contrib.seq2seq.BasicDecoder(
        cell = self.dec_cell,
        helper = training_helper,
        initial_state = self.initial_state,
        output_layer = self.output_layer
    )
    train_dec_outputs, train_dec_last_state,_ = tf.contrib.seq2seq.dynamic_decode(
        training_decoder,
        output_time_major=False,
        impute_finished=True,
        maximum_iterations=max_dec_len)

    # logits: [batch_size x max_dec_len x vocab_size]
    logits = tf.identity(train_dec_outputs.rnn_output, name='logits')

    # targets: [batch_size x max_dec_len x vocab_size]
```

```
        targets = tf.slice(self.target_tokens, [0, 0], [-1, max_dec_len], 'targets')

        masks =
tf.sequence_mask(self.target_length,max_dec_len,dtype=tf.float32,name="mask")
        self.losses =
tf.contrib.seq2seq.sequence_loss(logits=logits,targets=targets,weights=masks,name="loss
es")

        # prediction sample for validation
        self.valid_predictions = tf.identity(train_dec_outputs.sample_id,
name='valid_preds')

    def setup_inference_decoder_layer(self):
        start_tokens = tf.tile(tf.constant([self.start_token],dtype=tf.int32),[self.batch_size])
        inference_helper = tf.contrib.seq2seq.GreedyEmbeddingHelper(
            embedding=self.dec_emd,
            start_tokens=start_tokens,
            end_token=self.end_token)

        inference_decoder = tf.contrib.seq2seq.BasicDecoder(
            cell = self.dec_cell,
            helper = inference_helper,
            initial_state=self.initial_state,
            output_layer=self.output_layer)

        infer_dec_outputs, infer_dec_last_state,_ = tf.contrib.seq2seq.dynamic_decode(
                inference_decoder,
                output_time_major=False,
                impute_finished=True,
                maximum_iterations=self.max_inference_lenght)
        # [batch_size x dec_sentence_length], tf.int32
        self.predictions = tf.identity(infer_dec_outputs.sample_id, name='predictions')

    def setup_beam_search_decoder_layer(self):
        start_tokens = tf.tile(tf.constant([self.start_token],dtype=tf.int32),[self.batch_size])
```

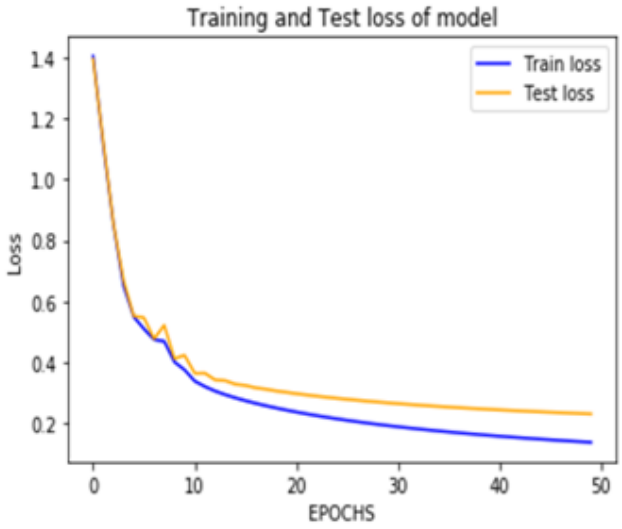
```
        bsd = tf.contrib.seq2seq.BeamSearchDecoder(
                cell=self.dec_cell,
                embedding=self.dec_emd,
                start_tokens= start_tokens,
                end_token=self.end_token,
                initial_state=self.initial_state,
                beam_width=self.beam_size,
                output_layer=self.output_layer,
                length_penalty_weight=0.0)
        # final_outputs are instances of FinalBeamSearchDecoderOutput
        final_outputs, final_state, final_sequence_lengths =
tf.contrib.seq2seq.dynamic_decode(
            bsd,
            output_time_major=False,
           # impute_finished=True,
            maximum_iterations=self.max_inference_lenght
        )
        beam_predictions = final_outputs.predicted_ids
        self.beam_predictions = tf.transpose(beam_predictions,perm=[0,2,1])
        self.beam_prob = final_outputs.beam_search_decoder_output.scores
        self.beam_ids = final_outputs.beam_search_decoder_output.predicted_ids

    def setup_attention_decoder(self):
        #dec_cell = multi_rnn_cell('gru',self.dim_size,num_layers=self.num_layers,
train_phase = self.train_phase, keep_prob=self.keep_prob_config)
        dec_cell =
[single_rnn_cell(self.cell_name,self.dim_size,self.train_phase,self.keep_prob_config) for
i in range(self.num_layers)]
        if self.beam_search:
            memory = tf.contrib.seq2seq.tile_batch(self.encode_output,multiplier =
self.beam_size)
            memory_sequence_length =
tf.contrib.seq2seq.tile_batch(self.source_length,multiplier = self.beam_size)
        else:
            memory = self.encode_output
```

```
            memory_sequence_length = self.source_length

        attn_mech = tf.contrib.seq2seq.BahdanauAttention(
            num_units = self.atten_size,
            memory = memory,
            memory_sequence_length = memory_sequence_length,
            name = "BahdanauAttention"
        )
        dec_cell[0] = tf.contrib.seq2seq.AttentionWrapper(
            cell=dec_cell[0],
            attention_mechanism=attn_mech,
            attention_layer_size=self.atten_size)

        if self.beam_search:
            tile_state = tf.contrib.seq2seq.tile_batch(self.decode_initial_state,self.beam_size)
            initial_state = [tile_state for i in range(self.num_layers)]
            cell_state =
dec_cell[0].zero_state(dtype=tf.float32,batch_size=self.batch_size*self.beam_size)
            initial_state[0] = cell_state.clone(cell_state=initial_state[0])
            self.initial_state = tuple(initial_state)
        else:
            # we use dynamic batch size
            self.batch_size = tf.shape(self.encoder_inputs)[0]
            initial_state = [self.decode_initial_state for i in range(self.num_layers)]
            cell_state = dec_cell[0].zero_state(dtype=tf.float32, batch_size = self.batch_size)
            initial_state[0] = cell_state.clone(cell_state=initial_state[0])
            self.initial_state = tuple(initial_state)

        print(self.initial_state)
        self.dec_cell = tf.contrib.rnn.MultiRNNCell(dec_cell)
        self.output_layer = Dense(self.vocab_size,kernel_initializer =
tf.truncated_normal_initializer(mean = 0.0, stddev=0.1))
        if self.train_phase:
            self.setup_training_decoder_layer()
        else:
```

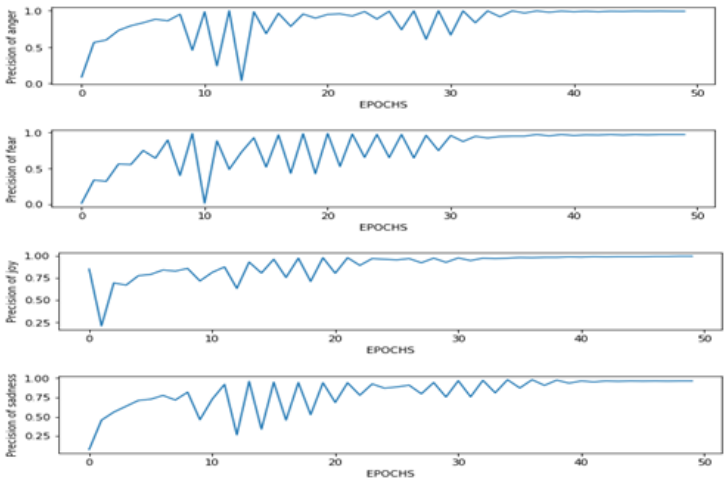
```
        if self.beam_search:
            self.setup_beam_search_decoder_layer()
        else:
            self.setup_inference_decoder_layer()

    def train_one_step(self,sess,encode_input,encode_len,decode_input,decode_len):
        feed_dict = {}
        feed_dict[self.source_tokens] = encode_input
        feed_dict[self.source_length] = encode_len
        feed_dict[self.target_tokens] = decode_input
        feed_dict[self.target_length] = decode_len
        valid_predictions,loss,_ =
sess.run([self.valid_predictions,self.losses,self.updates],feed_dict=feed_dict)
        return valid_predictions,loss

    def inference(self,sess,encode_input,encode_len):
        feed_dict = {}
        feed_dict[self.source_tokens] = encode_input
        feed_dict[self.source_length] = encode_len
        if self.beam_search:
            predictions,probs,ids =
sess.run([self.beam_predictions,self.beam_prob,self.beam_ids],feed_dict=feed_dict)
            return predictions,ids
        else:
            predictions = sess.run([self.predictions],feed_dict=feed_dict)
            return predictions

    def save_model(self,sess,checkpoint_dir):
        writer = tf.summary.FileWriter(checkpoint_dir, sess.graph)
        saver = tf.train.Saver(tf.global_variables())
        saver.save(sess,checkpoint_dir + "model.ckpt",global_step=self.global_step)

    def restore_model(self,sess,checkpoint_dir):
        saver = tf.train.Saver(tf.global_variables())
        saver.restore(sess,tf.train.latest_checkpoint(checkpoint_dir))
```

TRAINING PROCESS

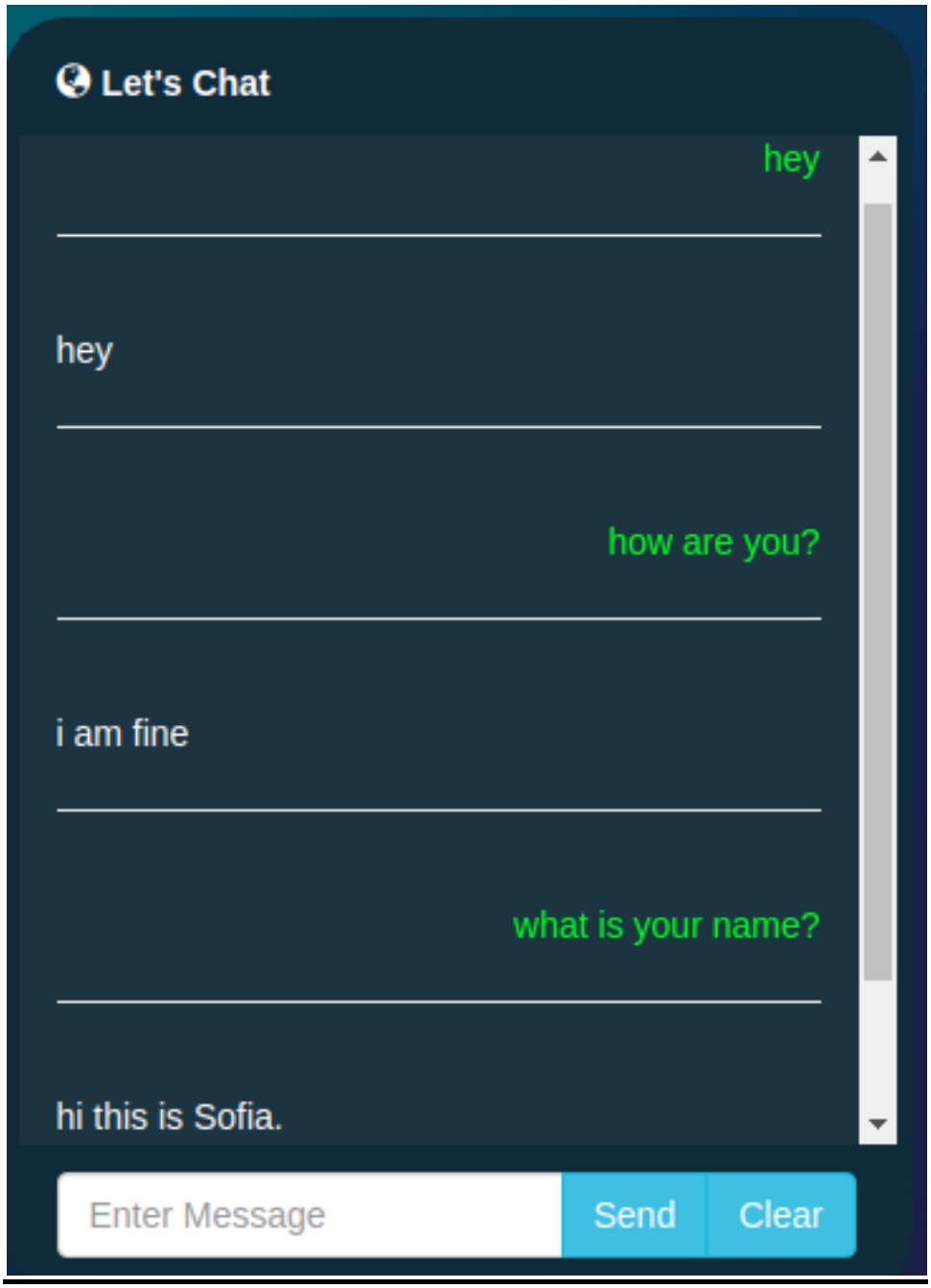
The model is trained for 50 epochs and learning rate is kept 0.1. Following training and validation set loss is observed. It is seen that our model converges nicely.



Following curve shows the precision of our model which is seen to be very good for all classes.



OUTPUT


Let's Chat
hey
hey
how are you?
i am fine
what is your name?
hi this is Sofia.
Enter Message
Send
Clear

RESULT

Arrangement to-grouping LSTM, a fake intermittent neural organization model shows more exact outcome for successive information by taking care of disappearing slope issue in RNN model. Proposing various approaches to the client to control their tension over sadness model ready to oversee around 70% exactness . The main hindrance of LSTM model is takes more time than other profound learning calculation as it deals with feedforward and back proliferation.

CONCLUSION

In this work, an astute chatbot is worked for the client who can talk transparently to the bot since there is no dread of judgment and it attempts to recommend a few methods for surviving their downturn. To recognize feeling of client from talk text and answer in like manner, profound learning calculations LSTM (RNN) is utilized which handle slope issue. There are still such countless variables which influences the discussion, where need to chip away at that.

We have seen bots like Woebot, Wysa, and Joy might have a great impact on providing help to people suffering from mental illness. Many people are still not educated about this technology of treating depression and how chatbots could be a real help. The chatbot is a great platform for anyone and is widely available anytime through an interface and internet access. The automated bot follows up the minimum standard it should meet- to respect user privacy, should be evidence-based and also ensure user's safety. Chatbots using LSTM have certainly made things sorted by making their clients identify emotions, distinguishing between healthy and unhealthy feelings and how distorted thoughts contribute to painful feelings. The chatbot therapist is an amazing invention in the healthcare department. It will encourage users to discuss their problems and emotions with ease.

FUTURE SCOPE

- Empowering various languages in chat.
- Review various parts of the language, which illuminate issues for chatbot-learning.
- Maintain a characteristic discussion stream of chatbot.

REFERENCES

[1] C. K. M, "Artificial paranoia: A computer program for the study of natural language communication between man and machine," ACM Communications, vol. 9, pp. 36–45, 1975

[2]] Bhargava V., Nikhil M. (2009), "An intelligent speech recognition system for education system", [Online]. Available: https://pdfs.semanticscholar.org/8ea7/725bab4e39 0e4d8ab8eb ee747d2a5340c3b2.pdf

[3] M. Wöllmer, F. Weninger, T. Knaup, B. Schuller, C. Sun, K. Sagae, and L. P. Morency, "Youtube movie reviews: Sentiment analysis in an audiovisual context," IEEE Intelligent Systems 28(3), pp. 46-53, 2013.

[4] B. Inkster, S. Sarda, and V. Subramanian, "An empathy-driven, conversational artificial intelligence agent (wysa) for digital mental wellbeing: real-world data evaluation mixed-methods study," JMIR mHealth and uHealth, vol. 6, no. 11, p. e12106, 2018.

[5] R. Plutchik, "Emotions and Life: Perspectives from Psychology, Biology, and Evolution," Washington, DC: American Psychological Association, 2002.

[6] B. K. Kim, J. Roh, S. Y. Dong, and S. Y. Lee, "Hierarchical committee of deep convolutional neural networks for robust facial expression recognition," Journal on Multimodal User Interfaces, pp. 1-17, 2016

[7] D. Elmasri, A. Maeder, "A Conversational Agent for an Online Mental Health Intervention," In proc. of International Conference on Brain and Health Informatics, pp. 243-251, 2016.

[8] T. Kiss and J. Strunk, "Unsupervised multilingual sentence boundary detection," Computational Linguistics, vol. 32, no. 4, pp. 485–525, 2006.