Machine Learning Assignment

Machine Learning for Cyber Security

BSc (Hons.) in Information Technology Specializing in Cyber Security

Sri Lanka Institute of Information Technology
Sri Lanka

May 2020

IT17354516
B I Sariffodeen

Contents

1. Introduction

1.1. Objective

Construct a simple, functional chatbot using a Machine Learning module.

1.2. Prerequisites

- Anaconda - Free and Open Source Distribution of Python Language to simplify Package Management and distribution
- Jupyter Notebook on conda - Open Source Web Application to develop live code
- TensorFlow - Open Source software library for data flow and differentiable programming
- Keras - Open Source Neural Network library in python
- NLTK - Symbolic and Statistical Natural Language Processing
- nvidia CUDA - Parallel Computing Platform and API model
- Pickle - Pickling converts objects hierarchy to byte stream and unpickling does the reverse
- Azure and Azure CLI - To host chatbot

1.3. Data Source

Dataset is collected from - data-flair

url : https://drive.google.com/file/d/1763Y5zy7HmRYsOoBLQgUxQRGY6xCgQiN/view

(FIle used - intents.json)

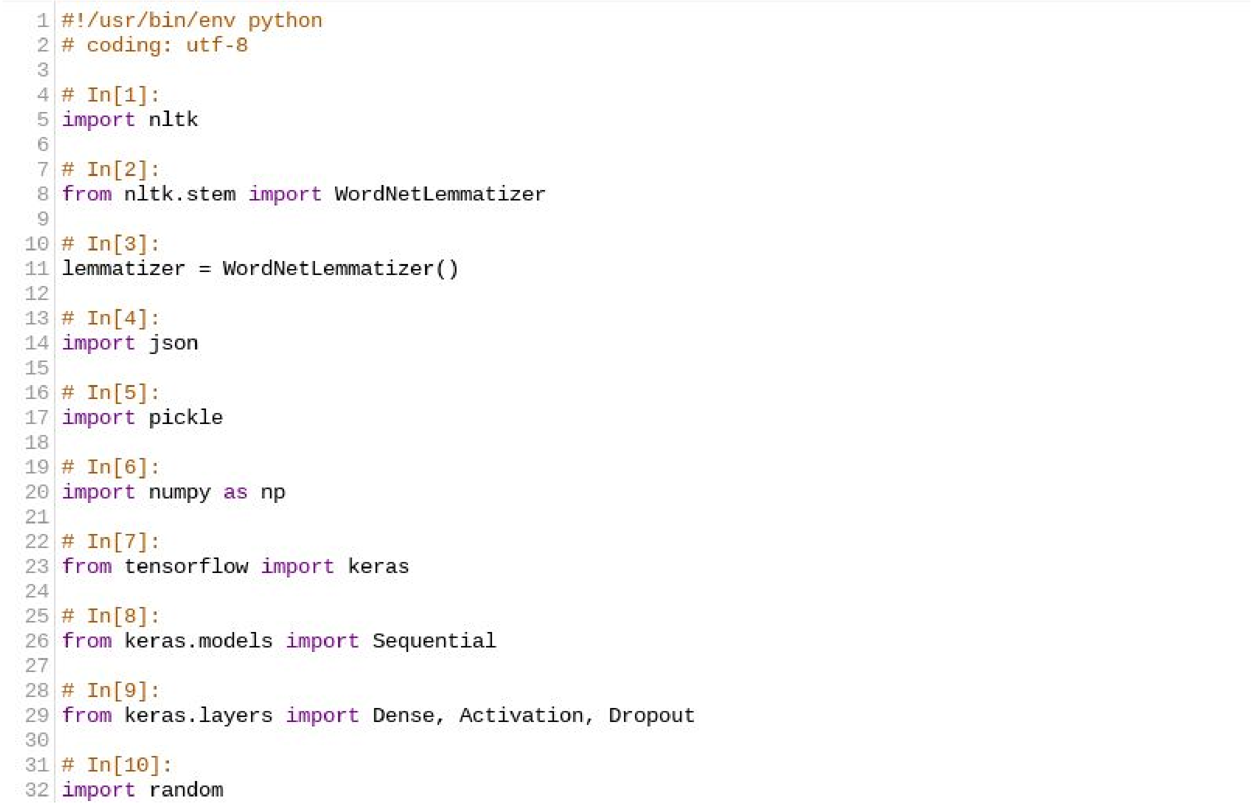
1.4. Files to be used

- intents.json
- train_chatbot.py & chatgui.py
- Words.pkl & Classes.pkl
- chatbot_model.h5

2. Initial Processes

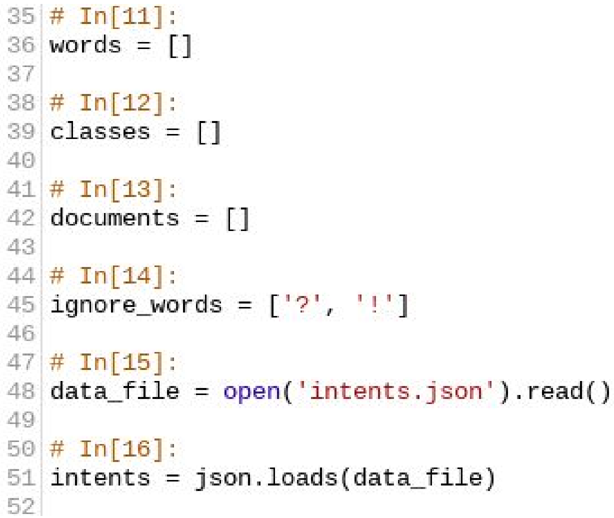
2.1. Import and Load data

Import Necessary Packages



```
#!/usr/bin/env python
# coding: utf-8

# In[1]:
import nltk

# In[2]:
from nltk.stem import WordNetLemmatizer

# In[3]:
lemmatizer = WordNetLemmatizer()

# In[4]:
import json

# In[5]:
import pickle

# In[6]:
import numpy as np

# In[7]:
from tensorflow import keras

# In[8]:
from keras.models import Sequential

# In[9]:
from keras.layers import Dense, Activation, Dropout

# In[10]:
import random
```

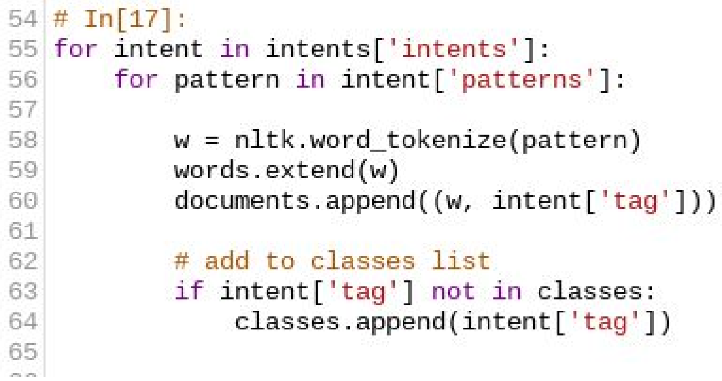
Parse intents.json files to python and read it



```
# In[11]:
words = []

# In[12]:
classes = []

# In[13]:
documents = []

# In[14]:
ignore_words = ['?', '!']

# In[15]:
data_file = open('intents.json').read()

# In[16]:
intents = json.loads(data_file)

```

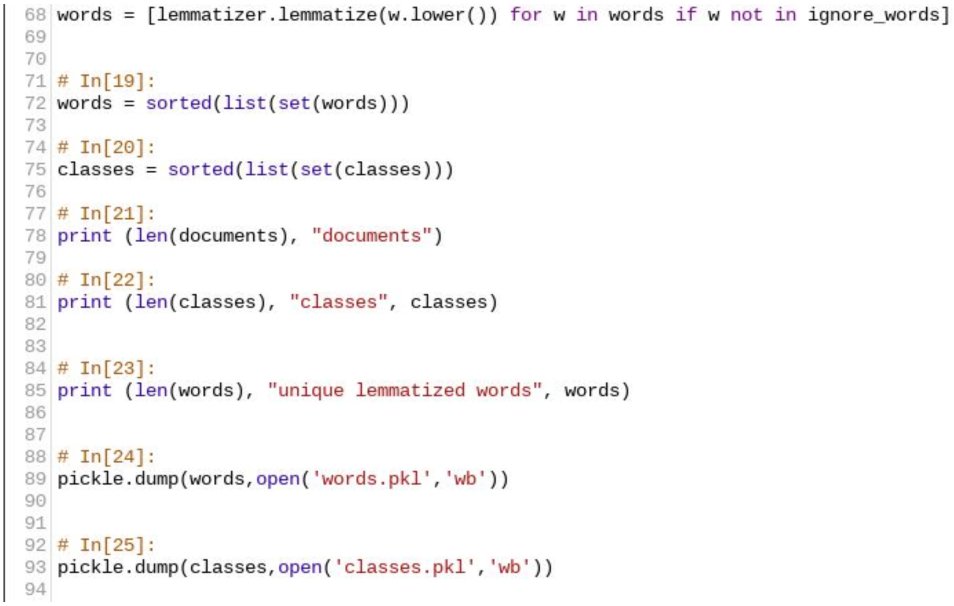
2.2. Pre-process Data

Tokenize the data



```
# In[17]:
for intent in intents['intents']:
    for pattern in intent['patterns']:

        w = nltk.word_tokenize(pattern)
        words.extend(w)
        documents.append((w, intent['tag']))

        # add to classes list
        if intent['tag'] not in classes:
            classes.append(intent['tag'])

```

Lemmatize the words



```
words = [lemmatizer.lemmatize(w.lower()) for w in words if w not in ignore_words]


# In[19]:
words = sorted(list(set(words)))

# In[20]:
classes = sorted(list(set(classes)))

# In[21]:
print (len(documents), "documents")

# In[22]:
print (len(classes), "classes", classes)


# In[23]:
print (len(words), "unique lemmatized words", words)


# In[24]:
pickle.dump(words,open('words.pkl','wb'))


# In[25]:
pickle.dump(classes,open('classes.pkl','wb'))

```

3. Train data and Build Model

3.1. About Algorithm

LSTM - Deep Learning model based on RNN architecture. Used to process sequences of data.

Commonly used in handwriting and speech recognition.

Mainly built to resolve long term dependency problem.

LSTM composes of 3 gates (input, output, forget) and a cell.

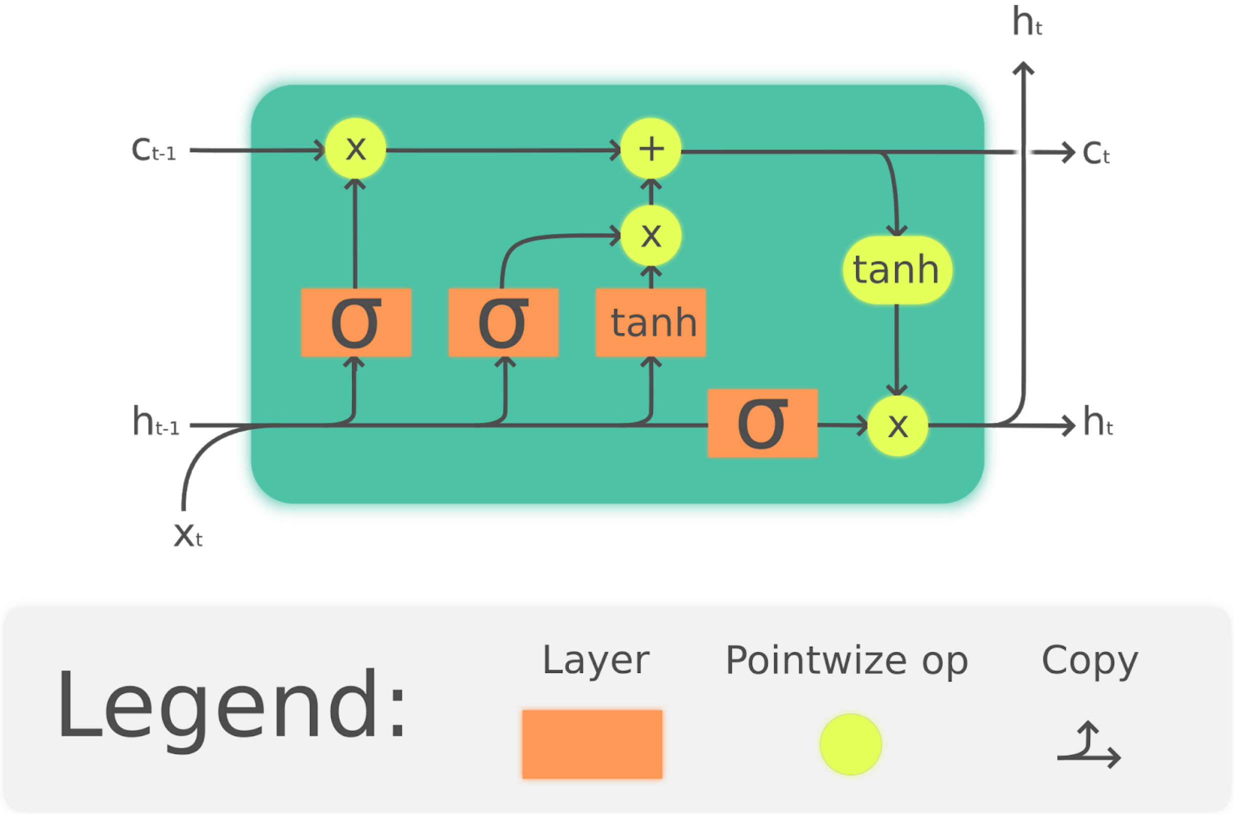


figure 1: Process of a cell in LSTM

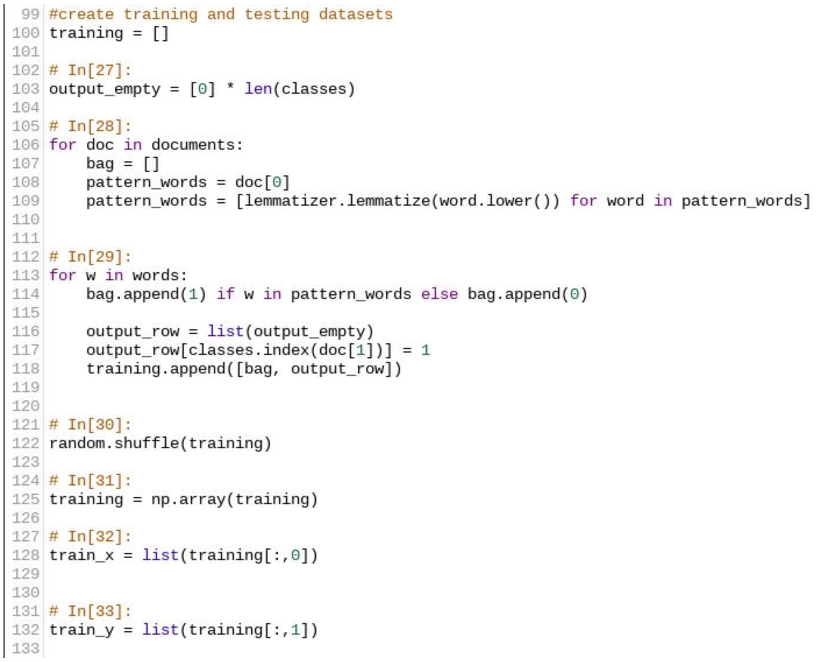
Natural Language Understanding occurs via a bi-directional LSTM network.

An input is processed through three layers to bring out a conclusive result.

3.2. Training data and building the model

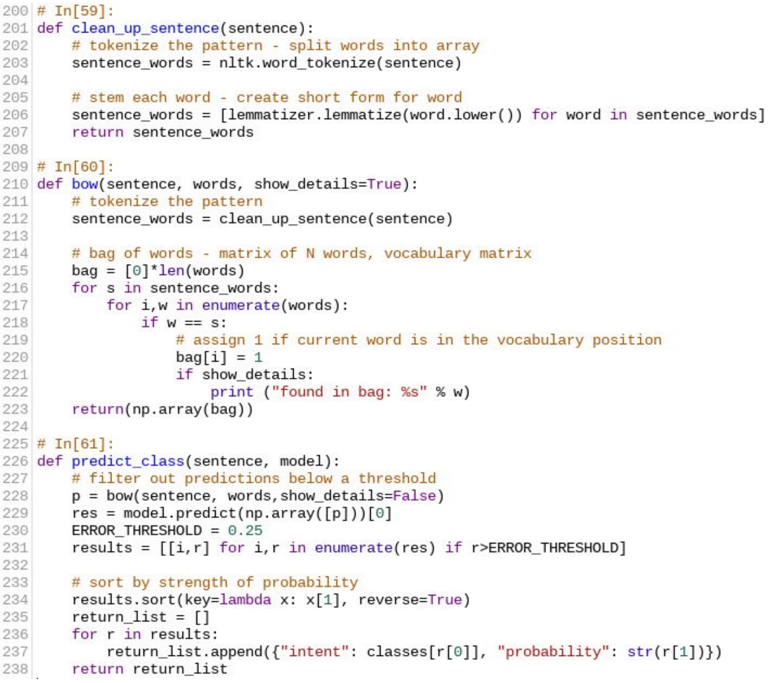
In training data, the input will be the pattern and output will be the class the pattern belongs to.

For the machine to understand patterns, text is converted to numbers



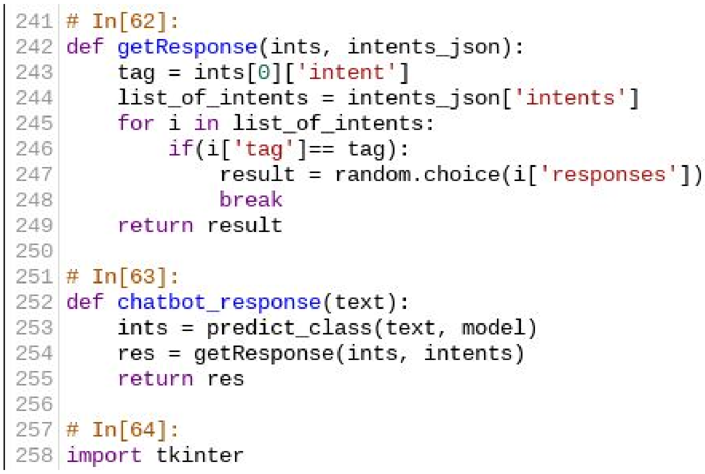
```
#create training and testing datasets
training = []

# In[27]:
output_empty = [0] * len(classes)

# In[28]:
for doc in documents:
    bag = []
    pattern_words = doc[0]
    pattern_words = [lemmatizer.lemmatize(word.lower()) for word in pattern_words]


# In[29]:
for w in words:
    bag.append(1) if w in pattern_words else bag.append(0)

    output_row = list(output_empty)
    output_row[classes.index(doc[1])] = 1
    training.append([bag, output_row])


# In[30]:
random.shuffle(training)

# In[31]:
training = np.array(training)

# In[32]:
train_x = list(training[:,0])


# In[33]:
train_y = list(training[:,1])

```

Predict the class of response using text-preprocessing



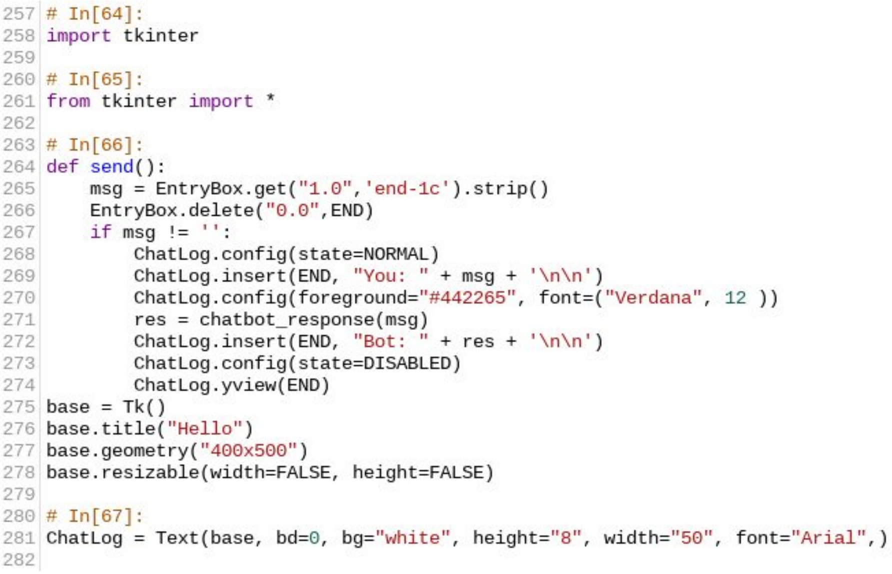
```
# In[59]:
def clean_up_sentence(sentence):
    # tokenize the pattern - split words into array
    sentence_words = nltk.word_tokenize(sentence)

    # stem each word - create short form for word
    sentence_words = [lemmatizer.lemmatize(word.lower()) for word in sentence_words]
    return sentence_words

# In[60]:
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))

# In[61]:
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
```

On predicting the class of response, random message is generated from relevant list of intents

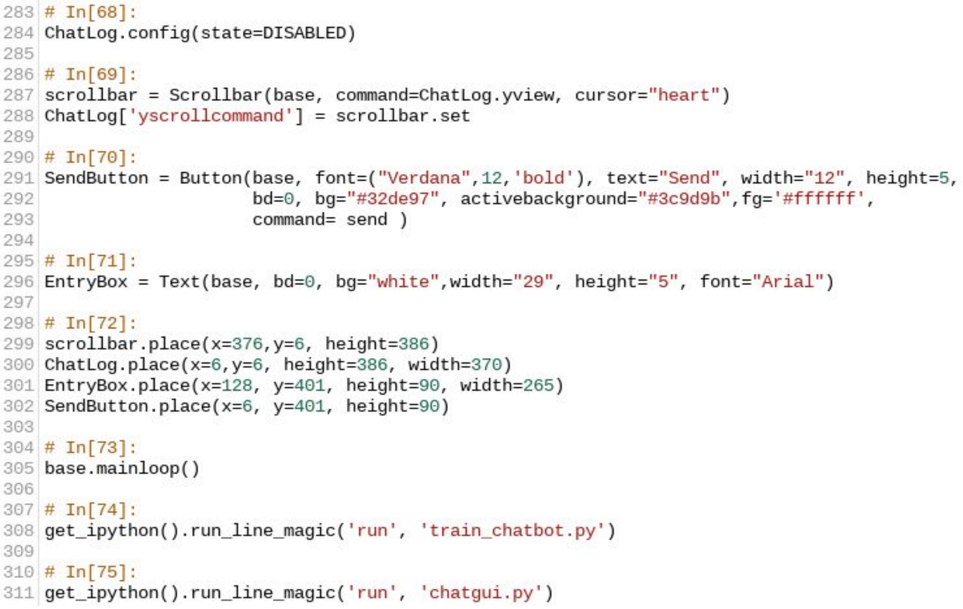


```
# In[62]:
def getResponse(ints, intents_json):
    tag = ints[0]['intent']
    list_of_intents = intents_json['intents']
    for i in list_of_intents:
        if(i['tag']== tag):
            result = random.choice(i['responses'])
            break
    return result

# In[63]:
def chatbot_response(text):
    ints = predict_class(text, model)
    res = getResponse(ints, intents)
    return res

# In[64]:
import tkinter
```

Create GUI and execute the model



```
# In[64]:
import tkinter

# In[65]:
from tkinter import *

# In[66]:
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("Hello")
base.geometry("400x500")
base.resizable(width=FALSE, height=FALSE)

# In[67]:
ChatLog = Text(base, bd=0, bg="white", height="8", width="50", font="Arial",)

```



```
# In[68]:
ChatLog.config(state=DISABLED)

# In[69]:
scrollbar = Scrollbar(base, command=ChatLog.yview, cursor="heart")
ChatLog['yscrollcommand'] = scrollbar.set

# In[70]:
SendButton = Button(base, font=("Verdana",12,'bold'), text="Send", width="12", height=5,
                    bd=0, bg="#32de97", activebackground="#3c9d9b",fg='#ffffff',
                    command= send )

# In[71]:
EntryBox = Text(base, bd=0, bg="white",width="29", height="5", font="Arial")

# In[72]:
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)

# In[73]:
base.mainloop()

# In[74]:
get_ipython().run_line_magic('run', 'train_chatbot.py')

# In[75]:
get_ipython().run_line_magic('run', 'chatgui.py')
```

4. Discussion

LSTM consists of memory cells which can store data for a certain period. Three gates control when data is input, output or forgotten. This helps LSTM learn longer term dependencies, neutralizing the vanishing gradient problem. This is why LSTM is ideal for the chatbot constructed.

This is a ‘Retrieval based’ chatbot which uses predefined classes of data to predict best responses.