Chatbot

Chintan Acharya 60004180016
Deep Nanda 60004180019
Kevin Pattni 60004180044

In [1]:
```
import nltk
from nltk.stem import WordNetLemmatizer
import json
import pickle
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout
from keras.optimizers import SGD
from keras.models import load_model
import random
from pprint import pprint
nltk.download('punkt')
nltk.download('wordnet')

words=[]
classes = []
documents = []
ignore_words = ['?', '!']
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

In [2]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call d
rive.mount("/content/drive", force_remount=True).
```

Loading datafile that has predefined patterns and responses

In [3]:
```
data_file = open('/content/drive/MyDrive/Chatbot/intents.json').read()
intents = json.loads(data_file)
pprint(intents['intents'])
```

```
[{'context': [''],
  'patterns': ['Hi there',
               'How are you',
               'Is anyone there?',
               'Hey',
               'Hola',
               'Hello',
               'Good day'],
  'responses': ['Hello, thanks for asking',
                'Good to see you again',
                'Hi there, how can I help?'],
  'tag': 'greeting'},
 {'context': [''],
  '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.'],
  'tag': 'goodbye'},
 {'context': [''],
  'patterns': ['Thanks',
               'Thank you',
               "That's helpful",
               'Awesome, thanks',
               'Thanks for helping me'],
  'responses': ['Happy to help!', 'Any time!', 'My pleasure'],
  'tag': 'thanks'},
 {'context': [''],
  'patterns': [],
  'responses': ["Sorry, can't understand you",
                'Please give me more info',
                'Not sure I understand'],
  'tag': 'noanswer'},
 {'context': [''],
  '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'],
  'tag': 'options'},
 {'context': [''],
  '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'],
  'tag': 'adverse_drug'},
 {'context': [''],
  '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'],
  'tag': 'blood_pressure'},
 {'context': ['search_blood_pressure_by_patient_id'],
  '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?'],
  'tag': 'blood_pressure_search'},
 {'context': [''],
  'patterns': [],
  'responses': ['Loading Blood pressure result for Patient'],
  'tag': 'search_blood_pressure_by_patient_id'},
 {'context': ['search_pharmacy_by_name'],
  'patterns': ['Find me a pharmacy',
               'Find pharmacy',
               'List of pharmacies nearby',
               'Locate pharmacy',
               'Search pharmacy'],
  'responses': ['Please provide pharmacy name'],
  'tag': 'pharmacy_search'},
 {'context': [''],
  'patterns': [],
  'responses': ['Loading pharmacy details'],
  'tag': 'search_pharmacy_by_name'},
 {'context': ['search_hospital_by_params'],
  'patterns': ['Lookup for hospital',
               'Searching for hospital to transfer patient',
               'I want to search hospital data',
               'Hospital lookup for patient',
               'Looking up hospital details'],
  'responses': ['Please provide hospital name or location'],
  'tag': 'hospital_search'},
 {'context': ['search_hospital_by_type'],
  'patterns': [],
  'responses': ['Please provide hospital type'],
  'tag': 'search_hospital_by_params'},
 {'context': [''],
  'patterns': [],
  'responses': ['Loading hospital details'],
  'tag': 'search_hospital_by_type'}]
```

Preprocessing the data:

Tokenize words and add documents to corpus. Also add unseen tags in the class list

In [4]:
```
for intent in intents['intents']:
    for pattern in intent['patterns']:
        w = nltk.word_tokenize(pattern)
        words.extend(w)
        documents.append((w, intent['tag']))
        if intent['tag'] not in classes:
            classes.append(intent['tag'])
```

Lemmaztize and lower each word and remove duplicates

In [5]:
```
lemmatizer = WordNetLemmatizer()
words = [lemmatizer.lemmatize(w.lower()) for w in words if w not in ignore_words
words = sorted(list(set(words)))
classes = sorted(list(set(classes)))
print(len(classes), "Classes", classes)
print(len(words), "unique lemmatized words")
pprint(words)
pickle.dump(words,open('/content/drive/MyDrive/Chatbot/words.pkl','wb'))
pickle.dump(classes,open('/content/drive/MyDrive/Chatbot/classes.pkl','wb'))
```

```
9 Classes ['adverse_drug', 'blood_pressure', 'blood_pressure_search', 'goodby
e', 'greeting', 'hospital_search', 'options', 'pharmacy_search', 'thanks']
88 unique lemmatized words
["'s",
 ',',
 'a',
 'adverse',
 'all',
 'anyone',
 'are',
 'awesome',
 'be',
 'behavior',
 'blood',
 'by',
 'bye',
 'can',
 'causing',
 'chatting',
 'check',
 'could',
 'data',
 'day',
 'detail',
 'do',
 'dont',
 'drug',
 'entry',
 'find',
 'for',
 'give',
 'good',
 'goodbye',
 'have',
 'hello',
 'help',
 'helpful',
 'helping',
 'hey',
 'hi',
 'history',
 'hola',
 'hospital',
 'how',
 'i',
 'id',
 'is',
 'later',
 'list',
 'load',
 'locate',
 'log',
 'looking',
 'lookup',
 'management',
 'me',
 'module',
 'nearby',
 'next',
 'nice',
 'of',
 'offered',
 'open',
 'patient',
 'pharmacy',
 'pressure',
 'provide',
 'reaction',
 'related',
 'result',
 'search',
 'searching',
 'see',
 'show',
 'suitable',
 'support',
 'task',
 'thank',
 'thanks',
 'that',
 'there',
 'till',
 'time',
 'to',
 'transfer',
 'up',
 'want',
 'what',
 'which',
 'with',
 'you']
```

Creating training and testing data

In [6]:
```
training = []
output_empty = [0] * len(classes)

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

random.shuffle(training)
training = np.array(training)

train_x = list(training[:,0])
train_y = list(training[:,1])
print("Training data created")
```

```
Training data created

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:16: VisibleDepreca
tionWarning: Creating an ndarray from ragged nested sequences (which is a list-
or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is de
precated. If you meant to do this, you must specify 'dtype=object' when creatin
g the ndarray
  app.launch_new_instance()
```

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

In [7]:
```
input_shape=(len(train_x[0]),)
model = Sequential()
model.add(Dense(128, input_shape=input_shape, 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'))
model.summary()
sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 128)               11392
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 9)                 585
=================================================================
Total params: 20,233
Trainable params: 20,233
Non-trainable params: 0
_________________________________________________________________
```

Compile and Fit the model

In [8]:
```
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy
```

In [9]:
```
hist = model.fit(np.array(train_x), np.array(train_y), epochs=200, batch_size=5,
```

```
Epoch 1/200
10/10 [==============================] - 1s 2ms/step - loss: 2.2525 - accurac
y: 0.0548
Epoch 2/200
10/10 [==============================] - 0s 2ms/step - loss: 2.2114 - accurac
y: 0.0909
Epoch 3/200
10/10 [==============================] - 0s 2ms/step - loss: 2.0237 - accurac
y: 0.1232
Epoch 4/200
10/10 [==============================] - 0s 2ms/step - loss: 2.0137 - accurac
y: 0.2758
Epoch 5/200
10/10 [==============================] - 0s 2ms/step - loss: 1.9217 - accurac
y: 0.3564
Epoch 6/200
10/10 [==============================] - 0s 2ms/step - loss: 1.7647 - accurac
y: 0.4817
Epoch 7/200
```

Save the model that has been trained

In [10]:
```
model.save('/content/drive/MyDrive/Chatbot/chatbot_model.h5', hist)
```

Loading the saved model for use

In [11]:
```
model = load_model('/content/drive/MyDrive/Chatbot/chatbot_model.h5')
intents = json.loads(open('/content/drive/MyDrive/Chatbot/intents.json').read())
words = pickle.load(open('/content/drive/MyDrive/Chatbot/words.pkl','rb'))
classes = pickle.load(open('/content/drive/MyDrive/Chatbot/classes.pkl','rb'))
```

Preprocessing the input

In [12]:
```
def clean_up_sentence(sentence):
    sentence_words = nltk.word_tokenize(sentence)
    sentence_words = [lemmatizer.lemmatize(word.lower()) for word in sentence_wor
    return sentence_words

def bow(sentence, words, show_details=True):
    sentence_words = clean_up_sentence(sentence)
    bag = [0]*len(words)
    for s in sentence_words:
        for i,w in enumerate(words):
            if w == s:
                bag[i] = 1
                if show_details:
                    print ("found in bag: %s" % w)
    return(np.array(bag))

def predict_class(sentence, model):
    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]
    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
```

Getting response for the input queries

```
In [13]: 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(text):
             ints = predict_class(text, model)
             res = getResponse(ints, intents)
             return res
```

Chatbot response for trial queries

```
In [14]: chatbot_response('hello')

Out[14]: 'Hello, thanks for asking'
```

```
In [15]: chatbot_response('Thank you')

Out[15]: 'Any time!'
```

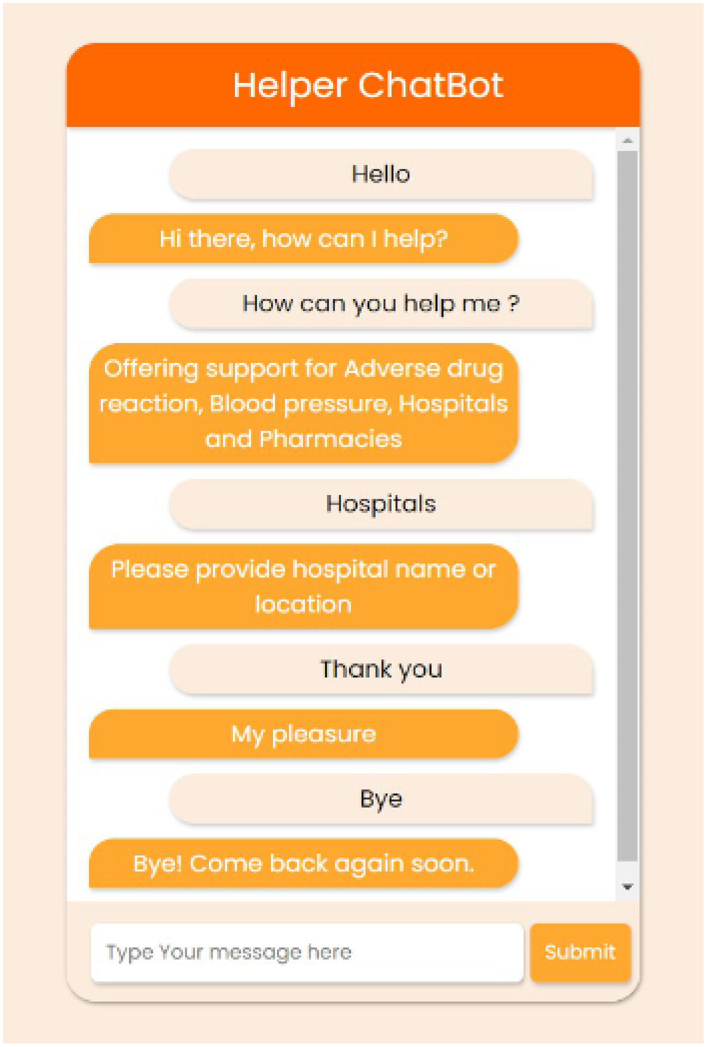
```
In [16]: chatbot_response('How are you?')

Out[16]: 'Hi there, how can I help?'
```

Implementation using React.js for Frontend and Flask for Backend

In [17]:
```
import cv2 as cv
from google.colab.patches import cv2_imshow
from skimage import io
from PIL import Image

image = io.imread("/content/drive/MyDrive/Implementation.jpg")
image= cv.cvtColor(image, cv.COLOR_BGR2RGB)
image = cv.resize(image, (350, 520))
cv2_imshow(image)
```



Helper ChatBot

Hello

Hi there, how can I help?

How can you help me ?

Offering support for Adverse drug reaction, Blood pressure, Hospitals and Pharmacies

Hospitals

Please provide hospital name or location

Thank you

My pleasure

Bye

Bye! Come back again soon.