Introduction:

A chatbot is an intelligent piece of software that is capable of communicating and performing actions similar to a human. Chatbots are used a lot in customer interaction, marketing on social network sites and instantly messaging the client. There are two basic types of chatbot models based on how they are built; Retrieval based and Generative based models.

A retrieval-based chatbot uses predefined input patterns and responses. It then uses some type of heuristic approach to select the appropriate response. It is widely used in the industry to make goal-oriented chatbots where we can customize the tone and flow of the chatbot to drive our customers with the best experience.

They are based on seq 2 seq neural networks. It is the same idea as machine translation. In machine translation, we translate the source code from one language to another language but here, we are going to transform input into an output. It needs a large amount of data and it is based on Deep Neural networks.

Import all libraries:

We import the necessary packages for our chatbot and initialize the variables we will use in our Python project.The data file is in JSON format so we used the json package to parse the JSON file into Python.

In [1]:
```
import nltk
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
import json
import pickle

import numpy as np
from tensorflow import keras
from keras.models import sequential
from keras.layers import Dense,Activation, Dropout

import random

words=[]# words for patterns
classes = [] #This is simply the tags of each intent
documents = []
ignore_words = ['?', '!']
data_file = open('intents.json').read()
intents = json.loads(data_file)
```

Preprocessing Data:

Tokenizing is the most basic and first thing and Tokenizing is the process of breaking the whole text into small parts like words.we iterate through the patterns and tokenize the sentence using nltk.word_tokenize() function and append each word in the words list. We also create a list of classes for our tags

In [2]:
```
#Loop through all the intents
# tokenize each pattern and append tokens to words, the patterns and
# the associated tag to their associated list

for intent in intents['intents']:
    for pattern in intent['patterns']:

        #tokenize each word
        w = nltk.word_tokenize(pattern)
        words.extend(w)
        #add documents in the corpus
        documents.append((w, intent['tag']))

#add the tag to the classes if it's not there already
        if intent['tag'] not in classes:
            classes.append(intent['tag'])
```

Lemmatization:

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

In [3]:

```
# lemmatize all the words in the vocab and convert them to lowercase
# if the words don't appear in punctuation
words = [lemmatizer.lemmatize(w.lower()) for w in words if w not in ignore_words]

# sorting the vocab and classes in alphabetical order and taking the # set to ensure no dupli
words = sorted(list(set(words)))
# sort classes
classes = sorted(list(set(classes)))


# documents = combination between patterns and intents
print (len(documents), "documents")
# classes = intents
print (len(classes), "classes", classes)
# words = all words, vocabulary
print (len(words), "unique lemmatized words", words)

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

```
33 documents
8 classes ['about', 'complaint', 'createaccount', 'goodbye', 'greeting', 'help', 'name', 'thank
s']
52 unique lemmatized words ["'s", 'a', 'about', 'account', 'an', 'anyone', 'are', 'bye', 'call'
, 'can', 'complaint', 'could', 'create', 'do', 'for', 'give', 'goodbye', 'hand', 'have', 'hay',
'hello', 'help', 'helpful', 'hey', 'hi', 'how', 'i', 'is', 'later', 'me', 'name', 'need', 'new'
, 'open', 'please', 'raise', 'see', 'service', 'should', 'support', 'thank', 'thanks', 'that',
'the', 'there', 'to', 'want', 'what', 'whats', 'who', 'you', 'your']
```

Training and Testing data

We've separated our data, we are now ready to train our algorithm. However, Neural Networks expect numerical values, and not words, to be fed into them, therefore, we first have to process our data so that a neural network could read what we are doing. Create the training data in which we will provide the input and the output. Our input will be the pattern and output will be the class our input pattern belongs to.

To convert our data to numerical values, we are going to leverage a technique called bag of words.

In [4]:

```
# create our training data
training = []
# create an empty array for our output
output_empty = [0] * len(classes)

# creating the bag of words model
# training set, bag of words for each sentence
for doc in documents:
    # initialize our bag of words
    bag = []
    # list of tokenized words for the pattern
    pattern_words = doc[0]

    # lemmatize each word - create base word, in attempt to represent related words
    pattern_words = [lemmatizer.lemmatize(word.lower()) for word in pattern_words]

    # create our bag of words array with 1, if word match found in current pattern
    for w in words:
        bag.append(1) if w in pattern_words else bag.append(0)

    # output is a '0' for each tag and '1' for current tag (for each pattern)
    #mark the index of class that the current pattern is associated
    # to
    output_row = list(output_empty)
    output_row[classes.index(doc[1])] = 1
```

```
    training.append([bag, output_row])

# shuffle the data and convert it to an array
random.shuffle(training)
training = np.array(training)

# split the features and target labels
# create train and test lists. X - patterns, Y - intents
train_x = list(training[:,0])
train_y = list(training[:,1])
print("Training data created")
```

```
Training data created

<ipython-input-4-69c6df989b5f>:31: VisibleDeprecationWarning: Creating an ndarray from ragged n
ested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths
or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creatin
g the ndarray
  training = np.array(training)
```

Build the model using Neural Network:

We have our training data ready, now we will build a deep neural network that has 3 layers. We use the Keras sequential API for this. After training the model for 200 epochs, we achieved 100% accuracy on our model.

In [5]:
```
import tensorflow as tf

adam = tf.keras.optimizers.Adam(learning_rate = 0.01, decay = 1e-6)
```

In [6]:
```
# Create model - 3 layers. First layer 128 neurons, second layer 64 neurons and 3rd output la
# equal to number of intents to predict output intent with softmax
from tensorflow.keras.models import Sequential
model = Sequential()
model.add(Dense(128, input_shape=(len(train_x[0]),), activation='relu'))
```

```
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(len(train_y[0]), activation='softmax'))

# Compile model.
model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])

#fit and save the model
hist = model.fit(np.array(train_x), np.array(train_y), epochs=200, batch_size=5, verbose=1)
model.save('chatbot_model.h5', hist)

print("model created")
```

```
7/7 [==============================] - 0s 908us/step - loss: 0.0793 - accuracy: 0.9697
Epoch 150/200
7/7 [==============================] - 0s 995us/step - loss: 0.0200 - accuracy: 1.0000
Epoch 151/200
7/7 [==============================] - 0s 1ms/step - loss: 0.0952 - accuracy: 0.9697
Epoch 152/200
7/7 [==============================] - 0s 931us/step - loss: 0.0078 - accuracy: 1.0000
Epoch 153/200
7/7 [==============================] - 0s 1ms/step - loss: 0.4822 - accuracy: 0.9091
Epoch 154/200
7/7 [==============================] - 0s 953us/step - loss: 0.0287 - accuracy: 0.9697
Epoch 155/200
7/7 [==============================] - 0s 1ms/step - loss: 2.0133e-04 - accuracy: 1.0000
Epoch 156/200
7/7 [==============================] - 0s 926us/step - loss: 0.2836 - accuracy: 0.9697
Epoch 157/200
7/7 [==============================] - 0s 988us/step - loss: 0.0213 - accuracy: 1.0000
Epoch 158/200
7/7 [==============================] - 0s 913us/step - loss: 0.1352 - accuracy: 0.9697
Epoch 159/200
```

Prediction:

To predict the sentences and get a response from the user. We will load the trained model and then use a graphical user interface that will predict the response from the bot. The model will only tell us the class it belongs to, so we will implement some functions which will identify the class and then retrieve us a random response from the list of responses.

Again we import the necessary packages and load the 'words.pkl' and 'classes.pkl' pickle files which we have created when we trained our model

In [7]:
```
import nltk
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
import pickle
import numpy as np

from keras.models import load_model
model = load_model('chatbot_model.h5')
import json
import random
intents = json.loads(open('intents.json').read())
words = pickle.load(open('words.pkl','rb'))
classes = pickle.load(open('classes.pkl','rb'))
```

To predict the class, we will need to provide input in the same way as we did while training. So we will create some functions that will perform text preprocessing and then predict the class.

In [8]:

```
def clean_up_sentence(sentence):
    # tokenize the pattern - split words into array
    sentence_words = nltk.word_tokenize(sentence)
    # stem each word - create short form for word
    sentence_words = [lemmatizer.lemmatize(word.lower()) for word in sentence_words]
    return sentence_words
# return bag of words array: 0 or 1 for each word in the bag that exists in the sentence

def bow(sentence, words, show_details=True):
    # tokenize the pattern
    sentence_words = clean_up_sentence(sentence)
    # bag of words - matrix of N words, vocabulary matrix
    bag = [0]*len(words)
    for s in sentence_words:
        for i,w in enumerate(words):
            if w == s:
                # assign 1 if current word is in the vocabulary position
                bag[i] = 1
                if show_details:
                    print ("found in bag: %s" % w)
    return(np.array(bag))

def predict_class(sentence, model):
    # filter out predictions below a threshold
    p = bow(sentence, words,show_details=False)
    res = model.predict(np.array([p]))[0]
    ERROR_THRESHOLD = 0.25
    results = [[i,r] for i,r in enumerate(res) if r>ERROR_THRESHOLD]
    # sort by strength of probability
    results.sort(key=lambda x: x[1], reverse=True)
    return_list = []
    for r in results:
        return_list.append({"intent": classes[r[0]], "probability": str(r[1])})
    return return_list
```

In [9]:
```
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
```

GUI with tkinter

We will develop a graphical user interface. Let's use Tkinter library with tons of useful libraries for GUI. We will take the input message from the user and then use the helper functions we have created to get the response from the bot and display it on the GUI.

In [*]:
```
#Creating GUI with tkinter
import tkinter
from tkinter import *


def send():
    msg = EntryBox.get("1.0",'end-1c').strip()
    EntryBox.delete("0.0",END)

    if msg != '':
        ChatLog.config(state=NORMAL)
        ChatLog.insert(END, "You: " + msg + '\n\n')
        ChatLog.config(foreground="#442265", font=("Verdana", 12 ))

```

```
        res = chatbot_response(msg)
        ChatLog.insert(END, "Bot: " + res + '\n\n')

        ChatLog.config(state=DISABLED)
        ChatLog.yview(END)

base = Tk()
base.title("Hello")
base.geometry("400x500")
base.resizable(width=FALSE, height=FALSE)

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

ChatLog.config(state=DISABLED)

#Bind scrollbar to Chat window
scrollbar = Scrollbar(base, command=ChatLog.yview, cursor="heart")
ChatLog['yscrollcommand'] = scrollbar.set

#Create Button to send message
SendButton = Button(base, font=("Verdana",12,'bold'), text="Send", width="12", height=5,
                    bd=0, bg="#32de97", activebackground="#3c9d9b",fg='#ffffff',
                    command= send )

#Create the box to enter message
EntryBox = Text(base, bd=0, bg="white",width="29", height="5", font="Arial")
#EntryBox.bind("<Return>", send)


#Place all components on the screen
scrollbar.place(x=376,y=6, height=386)
ChatLog.place(x=6,y=6, height=386, width=370)
EntryBox.place(x=128, y=401, height=90, width=265)
SendButton.place(x=6, y=401, height=90)

base.mainloop()
```

In [*]:

```
Conclusion:
    Thus we created a simple chatbot using NLTK.
```