**CHATBOTS - Using Natural Language Processing and Tensorflow**

In this Jupyter Notebook, We are going to Build a Chatbot that Understands the Context of Sentense and Respond accordingly. These are the Things that we are going to do in this Project -

1. Transforming the Conversational Intents into Tensorflow model (Neural Network using TFLEARN) using NLP and Save it as Pickle also.
2. Load the Same Pickle and Model to Build the Framework to Process the Responses.
3. At Last, We Show How the Inputs are Processed and Give the Reponses.

**TFLEARN - TFlearn is a modular and transparent deep learning library built on top of Tensorflow. It was designed to provide a higher-level API to TensorFlow in order to facilitate and speed-up experimentations, while remaining fully transparent and compatible with it. (**[**http://tflearn.org/**](http://tflearn.org/)**)**

**TENSORFLOW - TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.**

In [1]:

**from** IPython.core.display **import** Image, display

display(Image('Untitled.png'))

*#Used in Tensorflow Model*

**import** numpy **as** np

**import** tensorflow **as** tf

**import** tflearn

**import** random

*#Usde to for Contextualisation and Other NLP Tasks.*

**import** nltk

**from** nltk.stem.lancaster **import** LancasterStemmer

stemmer **=** LancasterStemmer()

*#Other*

**import** json

**import** pickle

**import** warnings

warnings**.**filterwarnings("ignore")

curses is not supported on this machine (please install/reinstall curses for an optimal experience)

In [3]:

print("Processing the Intents.....")

**with** open('intents.json') **as** json\_data:

intents **=** json**.**load(json\_data)

Processing the Intents.....

In [4]:

words **=** []

classes **=** []

documents **=** []

ignore\_words **=** ['?']

print("Looping through the Intents to Convert them to words, classes, documents and ignore\_words.......")

**for** intent **in** intents['intents']:

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

*# tokenize each word in the sentence*

w **=** nltk**.**word\_tokenize(pattern)

*# add to our words list*

words**.**extend(w)

*# add to documents in our corpus*

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

*# add to our classes list*

**if** intent['tag'] **not** **in** classes:

classes**.**append(intent['tag'])

Looping through the Intents to Convert them to words, classes, documents and ignore\_words.......

In [5]:

print("Stemming, Lowering and Removing Duplicates.......")

words **=** [stemmer**.**stem(w**.**lower()) **for** w **in** words **if** w **not** **in** ignore\_words]

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

*# remove duplicates*

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

print (len(documents), "documents")

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

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

Stemming, Lowering and Removing Duplicates.......

27 documents

9 classes ['goodbye', 'greeting', 'hours', 'mopeds', 'opentoday', 'payments', 'rental', 'thanks', 'today']

48 unique stemmed words ["'d", "'s", 'a', 'acceiv', 'anyon', 'ar', 'bye', 'can', 'card', 'cash', 'credit', 'day', 'do', 'doe', 'good', 'goodby', 'hav', 'hello', 'help', 'hi', 'hour', 'how', 'i', 'is', 'kind', 'lat', 'lik', 'mastercard', 'mop', 'of', 'on', 'op', 'rent', 'see', 'tak', 'thank', 'that', 'ther', 'thi', 'to', 'today', 'we', 'what', 'when', 'which', 'work', 'yo', 'you']

In [6]:

print("Creating the Data for our Model.....")

training **=** []

output **=** []

print("Creating an List (Empty) for Output.....")

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

print("Creating Traning Set, Bag of Words for our Model....")

**for** doc **in** documents:

*# initialize our bag of words*

bag **=** []

*# list of tokenized words for the pattern*

pattern\_words **=** doc[0]

*# stem each word*

pattern\_words **=** [stemmer**.**stem(word**.**lower()) **for** word **in** pattern\_words]

*# create our bag of words array*

**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*

output\_row **=** list(output\_empty)

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

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

Creating the Data for our Model.....

Creating an List (Empty) for Output.....

Creating Traning Set, Bag of Words for our Model....

In [7]:

print("Shuffling Randomly and Converting into Numpy Array for Faster Processing......")

random**.**shuffle(training)

training **=** np**.**array(training)

print("Creating Train and Test Lists.....")

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

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

print("Building Neural Network for Out Chatbot to be Contextual....")

print("Resetting graph data....")

tf**.**reset\_default\_graph()

Shuffling Randomly and Converting into Numpy Array for Faster Processing......

Creating Train and Test Lists.....

Building Neural Network for Out Chatbot to be Contextual....

Resetting graph data....

In [8]:

net **=** tflearn**.**input\_data(shape**=**[**None**, len(train\_x[0])])

net **=** tflearn**.**fully\_connected(net, 8)

net **=** tflearn**.**fully\_connected(net, 8)

net **=** tflearn**.**fully\_connected(net, len(train\_y[0]), activation**=**'softmax')

net **=** tflearn**.**regression(net)

print("Training....")

Training....

In [ ]:

model **=** tflearn**.**DNN(net, tensorboard\_dir**=**'tflearn\_logs')

In [ ]:

print("Training the Model.......")

model**.**fit(train\_x, train\_y, n\_epoch**=**1000, batch\_size**=**8, show\_metric**=True**)

print("Saving the Model.......")

model**.**save('model.tflearn')

Training Step: 3999 | total loss: **0.06984** | time: 0.011s

| Adam | epoch: 1000 | loss: 0.06984 - acc: 0.9976 -- iter: 24/27

Training Step: 4000 | total loss: **0.07164** | time: 0.014s

| Adam | epoch: 1000 | loss: 0.07164 - acc: 0.9978 -- iter: 27/27

--

Saving the Model.......

INFO:tensorflow:E:\FreeBirdsCrew\Chatbot\model.tflearn is not in all\_model\_checkpoint\_paths. Manually adding it.

In [ ]:

print("Pickle is also Saved..........")

pickle**.**dump( {'words':words, 'classes':classes, 'train\_x':train\_x, 'train\_y':train\_y}, open( "training\_data", "wb" ) )

Pickle is also Saved..........

In [ ]:

print("Loading Pickle.....")

data **=** pickle**.**load( open( "training\_data", "rb" ) )

words **=** data['words']

classes **=** data['classes']

train\_x **=** data['train\_x']

train\_y **=** data['train\_y']

**with** open('intents.json') **as** json\_data:

intents **=** json**.**load(json\_data)

print("Loading the Model......")

*# load our saved model*

model**.**load('./model.tflearn')

Loading Pickle.....

Loading the Model......

INFO:tensorflow:Restoring parameters from E:\FreeBirdsCrew\Chatbot\model.tflearn

In [ ]:

**def** clean\_up\_sentence(sentence):

*# It Tokenize or Break it into the constituents parts of Sentense.*

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

*# Stemming means to find the root of the word.*

sentence\_words **=** [stemmer**.**stem(word**.**lower()) **for** word **in** sentence\_words]

**return** sentence\_words

*# Return the Array of Bag of Words: True or False and 0 or 1 for each word of bag that exists in the Sentence*

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

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))

ERROR\_THRESHOLD **=** 0.25

print("ERROR\_THRESHOLD = 0.25")

**def** classify(sentence):

*# Prediction or To Get the Posibility or Probability from the Model*

results **=** model**.**predict([bow(sentence, words)])[0]

*# Exclude those results which are Below Threshold*

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

*# Sorting is Done because heigher Confidence Answer comes first.*

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

return\_list **=** []

**for** r **in** results:

return\_list**.**append((classes[r[0]], r[1])) *#Tuppl -> Intent and Probability*

**return** return\_list

**def** response(sentence, userID**=**'123', show\_details**=False**):

results **=** classify(sentence)

*# That Means if Classification is Done then Find the Matching Tag.*

**if** results:

*# Long Loop to get the Result.*

**while** results:

**for** i **in** intents['intents']:

*# Tag Finding*

**if** i['tag'] **==** results[0][0]:

*# Random Response from High Order Probabilities*

**return** print(random**.**choice(i['responses']))

results**.**pop(0)

ERROR\_THRESHOLD = 0.25

In [ ]:

das

In [ ]:

**while** **True**:

input\_data **=** input("You- ")

answer **=** response(input\_data)

answer

You- Hi

Hello, thanks for visiting

You- open today?

Our hours are 9am-9pm every day

You- mopeds

Hi there, how can I help?

You- rent?

Hi there, how can I help?

You- rent mopeds?

Hello, thanks for visiting

You- rental

Good to see you again

You- Rental Mopends?

Hi there, how can I help?

You- how Much?

Good to see you again

You- much?

Bye! Come back again soon.

You- much

See you later, thanks for visiting

You- s

See you later, thanks for visiting

You- d

See you later, thanks for visiting

You- f

Bye! Come back again soon.

You- g

Have a nice day

You- h

See you later, thanks for visiting

You- j

Have a nice day

You- e

Have a nice day

You- r

See you later, thanks for visiting

You- s

Have a nice day

You- ad

Have a nice day

You- a

Have a nice day

You- da

Have a nice day

You-

Bye! Come back again soon.

You- a

Have a nice day

You- hi

Hello, thanks for visiting

You- mopads rent

Good to see you again

You- rent of mopads?

Good to see you again

You- yea

See you later, thanks for visiting