Build a Chatbot with Neural Network

We've discovered how to build a chatbot with cosine similarity. Now, let's explore how we might build one with neural network!

We will create our training data, train a neural network with them, then use the trained model to make our chatbot.

First, we will install required libraries. Uncomment the few blocks below only if you do not have the libraries installed.

In [1]:
```
#!pip install numpy scipy
#!pip install scikit-learn
#!pip install pillow
#!pip install h5py
```

In [2]:
```
#!pip install tensorflow
```

In [3]:
```
#!pip install tensorflow-gpu
```

In [4]:
```
#!pip install keras
```

1. Install Libraries

Firstly, we will install libraries needed for this neural network powered chatbot. Keras is a machine learning library which utilizes tensorflow (another lower level machine learning library) at the backend. This makes it easier for us to deploy deep neural network for this purpose.

In [5]:
```
from keras.models import Sequential
from keras.losses import categorical_crossentropy
from keras.optimizers import SGD
from keras.layers import Dense

from numpy import argmax
import numpy as np
import re
```

```
C:\Users\chtan\anaconda3\lib\site-packages\scipy\__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (detected version 1.26.1
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

2. Input training data

We will first include the following training data for our chatbot:

1. X represent the different possible inputs that users might enter
2. Y represent the intent of the inputs

In [6]:
```
X = ['Hi',
     'Hello',
     'How are you?',
     'I am making',
     'making',
     'working',
     'studying',
     'see you later',
     'bye',
     'goodbye']
```

In [7]:
```
print(len(X))
```

```
10
```

In [8]:
```
Y = ['greeting',
     'greeting',
     'greeting',
     'busy',
     'busy',
     'busy',
     'busy',
     'bye',
     'bye',
     'bye']
```

In [9]:
```
print(len(Y))
```

```
10
```

Notice that there are several different sentences that have similar intent. Here, we are only having 3 intents, but you can add as many as you want for your project!

This is the way our chatbot will work:

1. From the input sentence, we will identify the intent using our trained AI model.
2. For each intent, we have a prepared response.

For example, if we identify that the intent of the input is for a greeting, we might ask the chatbot to reply with a greeting as well, something like 'hi' or 'how are you doing?'

We will use machine learning to create a model that can classify input sentence into different intents. We make it as follows:

1. We create a training data (X and Y above) which contains a list of sentences and their intents.
2. Use the training data to train a classifier.
3. Vectorize input sentences and use classifier to determine intent.

3. Text processing

As usual, we will start with text processing. Do you remember the process?

3.1 Remove non alphanumeric characters

In [10]:
```
def remove_non_alpha_numeric_characters(sentence):
    new_sentence = ''
    for alphabet in sentence:
        if alphabet.isalpha() or alphabet == ' ':
            new_sentence += alphabet
    return new_sentence
```

In [11]:
```
def preprocess_data(X):
    X = [data_point.lower() for data_point in X]
    X = [remove_non_alpha_numeric_characters(
        sentence) for sentence in X]
    X = [data_point.strip() for data_point in X]
    X = [re.sub(' +', ' ',
                data_point) for data_point in X]
    return X
```

In [12]:
```
X = preprocess_data(X)

vocabulary = set()
for data_point in X:
    for word in data_point.split(' '):
        vocabulary.add(word)

vocabulary = list(vocabulary)
```

Create document vectors

In [13]:
```
X_encoded = []

def encode_sentence(sentence):
    sentence = preprocess_data([sentence])[0]
    sentence_encoded = [0] * len(vocabulary)
    for i in range(len(vocabulary)):
        if vocabulary[i] in sentence.split(' '):
            sentence_encoded[i] = 1
    return sentence_encoded

X_encoded = [encode_sentence(sentence) for sentence in X]
```

In [14]:
```
classes = list(set(Y))

Y_encoded = []
for data_point in Y:
    data_point_encoded = [0] * len(classes)
    for i in range(len(classes)):
        if classes[i] == data_point:
            data_point_encoded[i] = 1
    Y_encoded.append(data_point_encoded)
```

4. Create training data and test data

In [15]:
```
X_train = X_encoded
y_train = Y_encoded
X_test = X_encoded
y_test = Y_encoded
```

Print and check the data you are using for training and test data

In [16]:
```
print (y_test)
```

```
[[0, 1, 0], [0, 1, 0], [0, 1, 0], [0, 0, 1], [0, 0, 1], [0, 0, 1], [0, 0, 1], [1, 0,
0], [1, 0, 0], [1, 0, 0]]
```

In [17]:
```
print(X_train)
```

```
[[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0], [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0], [0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0], [1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0,
0], [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0], [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
1], [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
0]]
```

In [18]:
```
print(len(X_train))
```

```
10
```

In [19]:
```
y_train
```

Out[19]:
```
[[0, 1, 0],
 [0, 1, 0],
 [0, 1, 0],
 [0, 0, 1],
 [0, 0, 1],
 [0, 0, 1],
 [0, 0, 1],
 [1, 0, 0],
 [1, 0, 0],
 [1, 0, 0]]
```

What does y_train represent? Do you understand the array shown above?

5. Model training

Now we will use the training data to train our neural network.

In [20]:
```
model = Sequential()
model.add(Dense(units=64, activation='sigmoid',
                input_dim=len(X_train[0])))
model.add(Dense(units=len(y_train[0]), activation='softmax'))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                960

 dense_1 (Dense)             (None, 3)                 195

=================================================================
Total params: 1155 (4.51 KB)
Trainable params: 1155 (4.51 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [21]:

```
model.compile(loss=categorical_crossentropy,
              optimizer=SGD(lr=0.01,
                            momentum=0.9, nesterov=True))
model.fit(np.array(X_train), np.array(y_train), epochs=100, batch_size=16)
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use t
he legacy optimizer, e.g.,tf.keras.optimizers.legacy.SGD.
```

```
Epoch 1/100
1/1 [==============================] - 0s 271ms/step - loss: 1.2013
Epoch 2/100
1/1 [==============================] - 0s 3ms/step - loss: 1.1756
Epoch 3/100
1/1 [==============================] - 0s 3ms/step - loss: 1.1471
Epoch 4/100
1/1 [==============================] - 0s 3ms/step - loss: 1.1214
Epoch 5/100
1/1 [==============================] - 0s 4ms/step - loss: 1.1020
Epoch 6/100
1/1 [==============================] - 0s 3ms/step - loss: 1.0898
Epoch 7/100
1/1 [==============================] - 0s 3ms/step - loss: 1.0841
Epoch 8/100
1/1 [==============================] - 0s 3ms/step - loss: 1.0830
Epoch 9/100
1/1 [==============================] - 0s 3ms/step - loss: 1.0843
Epoch 10/100
1/1 [==============================] - 0s 3ms/step - loss: 1.0865
Epoch 11/100
1/1 [==============================] - 0s 3ms/step - loss: 1.0882
Epoch 12/100
1/1 [==============================] - 0s 3ms/step - loss: 1.0887
Epoch 13/100
1/1 [==============================] - 0s 3ms/step - loss: 1.0877
Epoch 14/100
1/1 [==============================] - 0s 2ms/step - loss: 1.0853
Epoch 15/100
1/1 [==============================] - 0s 3ms/step - loss: 1.0818
Epoch 16/100
1/1 [==============================] - 0s 4ms/step - loss: 1.0777
Epoch 17/100
1/1 [==============================] - 0s 3ms/step - loss: 1.0732
Epoch 18/100
1/1 [==============================] - 0s 3ms/step - loss: 1.0689
Epoch 19/100
1/1 [==============================] - 0s 3ms/step - loss: 1.0651
Epoch 20/100
1/1 [==============================] - 0s 3ms/step - loss: 1.0618
Epoch 21/100
1/1 [==============================] - 0s 3ms/step - loss: 1.0591
Epoch 22/100
1/1 [==============================] - 0s 4ms/step - loss: 1.0570
Epoch 23/100
1/1 [==============================] - 0s 3ms/step - loss: 1.0551
Epoch 24/100
1/1 [==============================] - 0s 3ms/step - loss: 1.0534
Epoch 25/100
1/1 [==============================] - 0s 3ms/step - loss: 1.0517
Epoch 26/100
1/1 [==============================] - 0s 4ms/step - loss: 1.0499
Epoch 27/100
1/1 [==============================] - 0s 3ms/step - loss: 1.0479
Epoch 28/100
1/1 [==============================] - 0s 3ms/step - loss: 1.0457
Epoch 29/100
1/1 [==============================] - 0s 3ms/step - loss: 1.0434
Epoch 30/100
1/1 [==============================] - 0s 3ms/step - loss: 1.0411
```

```
Epoch 31/100
1/1 [==============================] - 0s 3ms/step - loss: 1.0387
Epoch 32/100
1/1 [==============================] - 0s 3ms/step - loss: 1.0363
Epoch 33/100
1/1 [==============================] - 0s 4ms/step - loss: 1.0340
Epoch 34/100
1/1 [==============================] - 0s 4ms/step - loss: 1.0317
Epoch 35/100
1/1 [==============================] - 0s 3ms/step - loss: 1.0295
Epoch 36/100
1/1 [==============================] - 0s 3ms/step - loss: 1.0273
Epoch 37/100
1/1 [==============================] - 0s 3ms/step - loss: 1.0252
Epoch 38/100
1/1 [==============================] - 0s 3ms/step - loss: 1.0231
Epoch 39/100
1/1 [==============================] - 0s 3ms/step - loss: 1.0210
Epoch 40/100
1/1 [==============================] - 0s 4ms/step - loss: 1.0188
Epoch 41/100
1/1 [==============================] - 0s 3ms/step - loss: 1.0167
Epoch 42/100
1/1 [==============================] - 0s 4ms/step - loss: 1.0145
Epoch 43/100
1/1 [==============================] - 0s 2ms/step - loss: 1.0124
Epoch 44/100
1/1 [==============================] - 0s 4ms/step - loss: 1.0102
Epoch 45/100
1/1 [==============================] - 0s 3ms/step - loss: 1.0080
Epoch 46/100
1/1 [==============================] - 0s 3ms/step - loss: 1.0058
Epoch 47/100
1/1 [==============================] - 0s 3ms/step - loss: 1.0036
Epoch 48/100
1/1 [==============================] - 0s 3ms/step - loss: 1.0015
Epoch 49/100
1/1 [==============================] - 0s 3ms/step - loss: 0.9993
Epoch 50/100
1/1 [==============================] - 0s 3ms/step - loss: 0.9971
Epoch 51/100
1/1 [==============================] - 0s 4ms/step - loss: 0.9950
Epoch 52/100
1/1 [==============================] - 0s 3ms/step - loss: 0.9928
Epoch 53/100
1/1 [==============================] - 0s 3ms/step - loss: 0.9906
Epoch 54/100
1/1 [==============================] - 0s 3ms/step - loss: 0.9885
Epoch 55/100
1/1 [==============================] - 0s 3ms/step - loss: 0.9863
Epoch 56/100
1/1 [==============================] - 0s 3ms/step - loss: 0.9841
Epoch 57/100
1/1 [==============================] - 0s 2ms/step - loss: 0.9819
Epoch 58/100
1/1 [==============================] - 0s 3ms/step - loss: 0.9798
Epoch 59/100
1/1 [==============================] - 0s 3ms/step - loss: 0.9776
Epoch 60/100
1/1 [==============================] - 0s 3ms/step - loss: 0.9754
```

```
Epoch 61/100
1/1 [==============================] - 0s 2ms/step - loss: 0.9732
Epoch 62/100
1/1 [==============================] - 0s 3ms/step - loss: 0.9710
Epoch 63/100
1/1 [==============================] - 0s 3ms/step - loss: 0.9688
Epoch 64/100
1/1 [==============================] - 0s 2ms/step - loss: 0.9667
Epoch 65/100
1/1 [==============================] - 0s 3ms/step - loss: 0.9645
Epoch 66/100
1/1 [==============================] - 0s 4ms/step - loss: 0.9623
Epoch 67/100
1/1 [==============================] - 0s 3ms/step - loss: 0.9601
Epoch 68/100
1/1 [==============================] - 0s 2ms/step - loss: 0.9579
Epoch 69/100
1/1 [==============================] - 0s 4ms/step - loss: 0.9557
Epoch 70/100
1/1 [==============================] - 0s 2ms/step - loss: 0.9535
Epoch 71/100
1/1 [==============================] - 0s 3ms/step - loss: 0.9513
Epoch 72/100
1/1 [==============================] - 0s 13ms/step - loss: 0.9491
Epoch 73/100
1/1 [==============================] - 0s 3ms/step - loss: 0.9469
Epoch 74/100
1/1 [==============================] - 0s 3ms/step - loss: 0.9446
Epoch 75/100
1/1 [==============================] - 0s 3ms/step - loss: 0.9424
Epoch 76/100
1/1 [==============================] - 0s 4ms/step - loss: 0.9402
Epoch 77/100
1/1 [==============================] - 0s 3ms/step - loss: 0.9380
Epoch 78/100
1/1 [==============================] - 0s 3ms/step - loss: 0.9358
Epoch 79/100
1/1 [==============================] - 0s 5ms/step - loss: 0.9335
Epoch 80/100
1/1 [==============================] - 0s 3ms/step - loss: 0.9313
Epoch 81/100
1/1 [==============================] - 0s 3ms/step - loss: 0.9291
Epoch 82/100
1/1 [==============================] - 0s 4ms/step - loss: 0.9268
Epoch 83/100
1/1 [==============================] - 0s 3ms/step - loss: 0.9246
Epoch 84/100
1/1 [==============================] - 0s 3ms/step - loss: 0.9223
Epoch 85/100
1/1 [==============================] - 0s 3ms/step - loss: 0.9201
Epoch 86/100
1/1 [==============================] - 0s 6ms/step - loss: 0.9178
Epoch 87/100
1/1 [==============================] - 0s 3ms/step - loss: 0.9156
Epoch 88/100
1/1 [==============================] - 0s 3ms/step - loss: 0.9133
Epoch 89/100
1/1 [==============================] - 0s 4ms/step - loss: 0.9110
Epoch 90/100
1/1 [==============================] - 0s 4ms/step - loss: 0.9087
```

```
Epoch 91/100
1/1 [==============================] - 0s 2ms/step - loss: 0.9065
Epoch 92/100
1/1 [==============================] - 0s 3ms/step - loss: 0.9042
Epoch 93/100
1/1 [==============================] - 0s 3ms/step - loss: 0.9019
Epoch 94/100
1/1 [==============================] - 0s 13ms/step - loss: 0.8996
Epoch 95/100
1/1 [==============================] - 0s 3ms/step - loss: 0.8973
Epoch 96/100
1/1 [==============================] - 0s 3ms/step - loss: 0.8950
Epoch 97/100
1/1 [==============================] - 0s 3ms/step - loss: 0.8927
Epoch 98/100
1/1 [==============================] - 0s 4ms/step - loss: 0.8904
Epoch 99/100
1/1 [==============================] - 0s 3ms/step - loss: 0.8881
Epoch 100/100
1/1 [==============================] - 0s 3ms/step - loss: 0.8858
```

Out[21]: <keras.src.callbacks.History at 0x26e41fc6eb0>

List down predictions

In [22]:
```
predictions = [argmax(pred) for pred in model.predict(np.array(X_test))]
```

```
1/1 [==============================] - 0s 54ms/step
```

Model Evaluation

Let's evaluate our model now. We will compare the prediction made by the model and our test data:

In [23]:
```
correct = 0
for i in range(len(predictions)):
    if predictions[i] == argmax(y_test[i]):
        correct += 1

print ("Correct:", correct)
print ("Total:", len(predictions))
```

```
Correct: 7
Total: 10
```

Testing the chatbot

Let's test the chatbot now! We will input a sentence, and then see what class is predicted by the neural network:

In [24]:
```
while True:
    print ("Enter a sentence")
```

```
    sentence = input()
    if sentence.lower() == "exit":
        break
    prediction= model.predict(np.array([encode_sentence(sentence)]))
    print (classes[argmax(prediction)])
```

```
Enter a sentence
hi
1/1 [==============================] - 0s 15ms/step
busy
Enter a sentence
hello
1/1 [==============================] - 0s 14ms/step
greeting
Enter a sentence
how are you doing?
1/1 [==============================] - 0s 14ms/step
greeting
Enter a sentence
thanks
1/1 [==============================] - 0s 14ms/step
busy
Enter a sentence
bye
1/1 [==============================] - 0s 14ms/step
busy
Enter a sentence
good bye
1/1 [==============================] - 0s 15ms/step
busy
Enter a sentence
exit
```

Realize that you can't stop the chatbot? You'll have to add the exit command later (see the previous notebook to find out how to do it.

For now, simply press the stop button (interrupt button) above to stop the chatbot.

Try it! press the stop button, and try typing something onto the box.

Challenge

We have successfully use neural network to map our input to conversation intent. Your challenge is to link the conversation intent to a particular response that the chatbot will say. For example, if the conversation intent is 'greeting', get your chatbot to say a greeting as well!

In [25]:
```
while True:
    print ("Enter a sentence")
    sentence = input()
    prediction= model.predict(np.array([encode_sentence(sentence)]))
    #print (classes[argmax(prediction)])
    if sentence.lower() == "exit":
        break
    if (classes[argmax(prediction)] == 'bye'):
        print ('Goodbye! See you next time!')
```

```
    elif (classes[argmax(prediction)] == 'busy'):
        print('ah... all the best!')
    elif (classes[argmax(prediction)] == 'greeting'):
        print ('Hi!')
```

```
Enter a sentence
hi
1/1 [==============================] - 0s 15ms/step
ah... all the best!
Enter a sentence
hello
1/1 [==============================] - 0s 15ms/step
Hi!
Enter a sentence
how are you doing?
1/1 [==============================] - 0s 14ms/step
Hi!
Enter a sentence
what's the time now?
1/1 [==============================] - 0s 14ms/step
ah... all the best!
Enter a sentence
exit
1/1 [==============================] - 0s 14ms/step
```

Great job! You've successfully created a simple chatbot with neural network! How might you improve the chatbot?

You can improve the chatbot by:

- Adding more training data
- Adding more intent
- Focusing on a particular topic and train the chatbot with many training data in that topic

Resource:

https://blog.eduonix.com/internet-of-things/simple-nlp-based-chatbot-python/