ChatBot-Project

By:- Mohd Nabeel Shamsi

STEP-1 (Importing Libraries)

In [24]:
```
import pickle
import numpy as np
```

Step-2(Loading Datasets)

In [25]:
```
with open("train_qa-DS.txt", "rb") as fp:
    train_data = pickle.load(fp)
```

In [26]:
```
train_data[10]
```

Out[26]:
```
(['Sandra',
  'went',
  'back',
  'to',
  'the',
  'hallway',
  '.',
  'Sandra',
  'moved',
  'to',
  'the',
  'office',
  '.'],
 ['Is', 'Sandra', 'in', 'the', 'office', '?'],
 'yes')
```

In [27]:
```
with open("test_qa-DS.txt", "rb") as fp:
    test_data = pickle.load(fp)
```

In [28]:
```
test_data[10]
```

Out[28]:
```
(['John',
  'moved',
  'to',
  'the',
  'hallway',
  '.',
  'Sandra',
  'went',
  'to',
  'the',
  'bedroom',
  '.'],
 ['Is', 'John', 'in', 'the', 'hallway', '?'],
 'yes')
```

In [29]:
```
type(test_data)
```

Out[29]:
```
list
```

In [30]:
```
type(train_data)
```

Out[30]:
```
list
```

In [31]:
```
len(train_data)
```

Out[31]:
```
10000
```

In [32]:
```
len(test_data)
```

Out[32]:
```
1000
```

In [33]:
```
train_data[0]
```

Out[33]:
```
(['Mary',
  'moved',
  'to',
  'the',
  'bathroom',
  '.',
  'Sandra',
  'journeyed',
  'to',
  'the',
  'bedroom',
  '.'],
 ['Is', 'Sandra', 'in', 'the', 'hallway', '?'],
 'no')
```

In [34]:
```
#Story
' '.join(train_data[0][0])
```

Out[34]:
```
'Mary moved to the bathroom . Sandra journeyed to the bedroom .'
```

In [35]:
```
#My_question
' '.join(train_data[0][1])
```

Out[35]:
```
'Is Sandra in the hallway ?'
```

In [36]:
```
#My_answer
train_data[0][2]
```

Out[36]:
```
'no'
```

In [37]:
```
#Setting up Vocabulary
vocab = set()
```

In [38]:
```
all_data = test_data + train_data
```

In [39]:
```
all_data[10]
```

Out[39]:
```
(['John',
  'moved',
  'to',
  'the',
  'hallway',
  '.',
  'Sandra',
  'went',
  'to',
  'the',
  'bedroom',
  '.'],
 ['Is', 'John', 'in', 'the', 'hallway', '?'],
 'yes')
```

In [40]:
```
for story, question, answer in all_data:
    vocab = vocab.union(set(story))
    vocab = vocab.union(set(question))
```

In [41]:
```
vocab.add('yes')
vocab.add('no')
```

In [42]:
```
vocab
```

Out[42]:
```
{'.',
 '?',
 'Daniel',
 'Is',
 'John',
 'Mary',
 'Sandra',
 'apple',
 'back',
 'bathroom',
 'bedroom',
 'discarded',
 'down',
 'dropped',
 'football',
 'garden',
 'got',
 'grabbed',
 'hallway',
 'in',
 'journeyed',
 'kitchen',
 'left',
 'milk',
 'moved',
 'no',
 'office',
 'picked',
 'put',
 'the',
 'there',
 'to',
 'took',
 'travelled',
 'up',
 'went',
 'yes'}
```

In [43]:
```
len(vocab)
```

Out[43]:
```
37
```

In [44]:
```
vocab_len = len(vocab) + 1
```

In [45]:
```
#Maximum Length of Story
max_story_len = max([len(data[0]) for data in all_data])
max_story_len
```

Out[45]:
```
156
```

In [46]:
```
#Maximum Length of Question
max_ques_len = max([len(data[1]) for data in all_data])
max_ques_len
```

Out[46]:
```
6
```

Step-3 (Processing the Datasets)

Importing relevant libraries

In [47]:
```
import tensorflow as tf
import keras
```

In [48]:
```
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

In [49]:
```
from tensorflow.keras.preprocessing.text import Tokenizer
```

Performing Tokenization

In [50]:
```
tokenizer = Tokenizer(filters = [])
```

In [51]:
```
tokenizer.fit_on_texts(vocab)
```

In [52]:
```
tokenizer.word_index
```

Out[52]:
```
{'office': 1,
 'journeyed': 2,
 'football': 3,
 'yes': 4,
 'apple': 5,
 'dropped': 6,
 'picked': 7,
 'milk': 8,
 'is': 9,
 'left': 10,
 'travelled': 11,
 'in': 12,
 'bathroom': 13,
 'discarded': 14,
 'went': 15,
 'sandra': 16,
 'there': 17,
 'garden': 18,
 'back': 19,
 'the': 20,
 'got': 21,
 'grabbed': 22,
 'up': 23,
 'put': 24,
 'daniel': 25,
 'hallway': 26,
 'mary': 27,
 'bedroom': 28,
 '?': 29,
 'john': 30,
 '.': 31,
 'took': 32,
 'down': 33,
 'no': 34,
 'to': 35,
 'moved': 36,
 'kitchen': 37}
```

In [53]:
```
train_story_text = []
train_question_text = []
train_answers = []

for story, question, answer in train_data:
    train_story_text.append(story)
    train_question_text.append(question)
```

In [54]:
```
train_story_seq = tokenizer.texts_to_sequences(train_story_text)
```

In [55]:
```
len(train_story_text)
```

Out[55]:
```
10000
```

In [56]:
```
len(train_story_seq)
```

Out[56]:
```
10000
```

In [57]:
```
train_story_seq[10]
```

Out[57]:
```
[16, 15, 19, 35, 20, 26, 31, 16, 36, 35, 20, 1, 31]
```

In [58]:
```
#Defining a Function
```

In [169…
```
def vectorize_stories(data, word_index = tokenizer.word_index,
                      max_story_len = max_story_len, max_ques_len = max_ques_len):



    X = [] #stories
    Xq = [] #query/question
    Y = [] #correct_answer


    for story, query, answer in data:

        x = [word_index[word.lower()] for word in story]
        xq = [word_index[word.lower()] for word in query]
        y = np.zeros(len(word_index) +1)
        y[word_index[answer]] = 1

        X.append(x)
        Xq.append(xq)
        Y.append(y)

    return(pad_sequences(X, maxlen = max_story_len) ,
           pad_sequences(Xq, maxlen = max_ques_len) ,
           np.array(Y) )
```

In [170…
```
inputs_train, queries_train, answers_train = vectorize_stories(train_data)
```

In [171…
```
inputs_test, queries_test, answers_test = vectorize_stories(test_data)
```

In [172…
```
inputs_train
```

Out[172]:
```
array([[ 0,  0,  0, ..., 29, 24, 20],
       [ 0,  0,  0, ..., 29,  4, 20],
       [ 0,  0,  0, ..., 29, 31, 20],
       ...,
       [ 0,  0,  0, ..., 29, 24, 20],
       [ 0,  0,  0, ..., 10,  5, 20],
       [ 0,  0,  0, ..., 30,  5, 20]])
```

In [173…
```
queries_test
```

Out[173]:
```
array([[37, 28, 13, 29,  1,  8],
       [37, 28, 13, 29,  1,  8],
       [37, 28, 13, 29, 27,  8],
       ...,
       [37, 26, 13, 29, 24,  8],
       [37, 36, 13, 29, 27,  8],
       [37, 26, 13, 29, 27,  8]])
```

In [174…
```
answers_test
```

Out[174]:
```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

In [175…
```
tokenizer.word_index['yes']
```

Out[175]:
```
16
```

In [176…
```
tokenizer.word_index['no']
```

Out[176]:
```
19
```

Step-4(Creating the Model)

In [177…
```
from tensorflow.keras.models import Sequential, Model
```

In [178…
```
import keras.layers
```

In [179…
```
from keras.layers import Input, Activation, Dense, Permute, Dropout, add, dot, concatenate, LSTM
```

In [180…
```
from keras.layers import Embedding
```

In [181…
```
input_sequence = Input((max_story_len,))
question = Input((max_ques_len,))
```

In [182…
```
#To build end-to-end network
```

In [183…
```
#Input Encoder n
input_encoder_n = Sequential()
input_encoder_n.add(Embedding(input_dim = vocab_len, output_dim = 64))
input_encoder_n.add(Dropout(0.3))
```

In [184…
```
#Input_encoder_c
input_encoder_c = Sequential()
input_encoder_c.add(Embedding(input_dim = vocab_len, output_dim = max_ques_len))
input_encoder_c.add(Dropout(0.3))
```

In [185…
```
#Question encoder
question_encoder = Sequential()
question_encoder.add(Embedding(input_dim = vocab_len, output_dim = 64, input_length = max_ques_len))
question_encoder.add(Dropout(0.3))
```

In [186…
```
#Encode the Sequences
input_encoded_n = input_encoder_n(input_sequence)
input_encoded_c = input_encoder_c(input_sequence)
question_encoded = question_encoder(question)
```

In [187…
```
#Using Dot-Product
match = dot([input_encoded_n,question_encoded], axes = (2,2))
match = Activation('softmax')(match)
```

In [188…
```
response = add([match, input_encoded_c])
response = Permute((2,1))(response)
```

In [189…
```
#Applying Concatenate
answer = concatenate([response, question_encoded ])
```

In [190…
```
answer
```

Out[190]:
```
<KerasTensor: shape=(None, 6, 220) dtype=float32 (created by layer 'concatenate_1')>
```

In [191…
```
#Applying LSTM
answer = LSTM(32)(answer)
```

In [192…
```
#Regularize with Dropout
answer = Dropout(0.5)(answer)
answer = Dense(vocab_len)(answer)
```

In [193…
```
answer = Activation('softmax')(answer)
```

Step- 5(Building The Model)

In [194…
```
model = Model([input_sequence, question], answer )
model.compile(optimizer ='rmsprop', loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

In [195…
```
model.summary()
```

```
Model: "model_1"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_3 (InputLayer)           [(None, 156)]        0           []

 input_4 (InputLayer)           [(None, 6)]          0           []

 sequential_4 (Sequential)      (None, None, 64)     2432        ['input_3[0][0]']

 sequential_6 (Sequential)      (None, 6, 64)        2432        ['input_4[0][0]']

 dot_2 (Dot)                    (None, 156, 6)       0           ['sequential_4[0][0]',
                                                                  'sequential_6[0][0]']

 activation_2 (Activation)      (None, 156, 6)       0           ['dot_2[0][0]']

 sequential_5 (Sequential)      (None, None, 6)      228         ['input_3[0][0]']

 add_1 (Add)                    (None, 156, 6)       0           ['activation_2[0][0]',
                                                                  'sequential_5[0][0]']

 permute_1 (Permute)            (None, 6, 156)       0           ['add_1[0][0]']

 concatenate_1 (Concatenate)    (None, 6, 220)       0           ['permute_1[0][0]',
                                                                  'sequential_6[0][0]']

 lstm_1 (LSTM)                  (None, 32)           32384       ['concatenate_1[0][0]']

 dropout_7 (Dropout)            (None, 32)           0           ['lstm_1[0][0]']

 dense_1 (Dense)                (None, 38)           1254        ['dropout_7[0][0]']

 activation_3 (Activation)      (None, 38)           0           ['dense_1[0][0]']

==================================================================================================
Total params: 38,730
Trainable params: 38,730
Non-trainable params: 0
__________________________________________________________________________________________________
```

In [196…
```
history = model.fit([inputs_train, queries_train], answers_train,
                    batch_size = 32,  epochs = 20,
                    validation_data = ([inputs_test, queries_test], answers_test)
                    )
```

```
Epoch 1/20
313/313 [==============================] - 7s 15ms/step - loss: 0.8856 - accuracy: 0.4952 - val_loss: 0.6946 - val_accuracy: 0.5
030
Epoch 2/20
313/313 [==============================] - 4s 13ms/step - loss: 0.7054 - accuracy: 0.4974 - val_loss: 0.6946 - val_accuracy: 0.4
970
Epoch 3/20
313/313 [==============================] - 3s 11ms/step - loss: 0.6973 - accuracy: 0.5038 - val_loss: 0.6968 - val_accuracy: 0.4
970
Epoch 4/20
313/313 [==============================] - 3s 11ms/step - loss: 0.6960 - accuracy: 0.5067 - val_loss: 0.6932 - val_accuracy: 0.5
000
Epoch 5/20
313/313 [==============================] - 4s 12ms/step - loss: 0.6959 - accuracy: 0.4983 - val_loss: 0.6932 - val_accuracy: 0.4
970
Epoch 6/20
313/313 [==============================] - 4s 12ms/step - loss: 0.6957 - accuracy: 0.4927 - val_loss: 0.6937 - val_accuracy: 0.4
970
Epoch 7/20
313/313 [==============================] - 4s 12ms/step - loss: 0.6951 - accuracy: 0.4973 - val_loss: 0.6934 - val_accuracy: 0.5
030
Epoch 8/20
313/313 [==============================] - 3s 11ms/step - loss: 0.6953 - accuracy: 0.5025 - val_loss: 0.6962 - val_accuracy: 0.4
970
Epoch 9/20
313/313 [==============================] - 4s 11ms/step - loss: 0.6950 - accuracy: 0.5015 - val_loss: 0.6942 - val_accuracy: 0.5
030
Epoch 10/20
313/313 [==============================] - 4s 12ms/step - loss: 0.6955 - accuracy: 0.4990 - val_loss: 0.6966 - val_accuracy: 0.4
970
Epoch 11/20
313/313 [==============================] - 4s 12ms/step - loss: 0.6950 - accuracy: 0.5002 - val_loss: 0.6933 - val_accuracy: 0.5
030
Epoch 12/20
313/313 [==============================] - 3s 11ms/step - loss: 0.6949 - accuracy: 0.5006 - val_loss: 0.6946 - val_accuracy: 0.5
030
Epoch 13/20
313/313 [==============================] - 4s 11ms/step - loss: 0.6956 - accuracy: 0.4951 - val_loss: 0.6935 - val_accuracy: 0.4
970
Epoch 14/20
313/313 [==============================] - 4s 12ms/step - loss: 0.6948 - accuracy: 0.5064 - val_loss: 0.6947 - val_accuracy: 0.4
970
Epoch 15/20
313/313 [==============================] - 4s 12ms/step - loss: 0.6944 - accuracy: 0.5085 - val_loss: 0.6932 - val_accuracy: 0.4
970
Epoch 16/20
313/313 [==============================] - 4s 11ms/step - loss: 0.6956 - accuracy: 0.4939 - val_loss: 0.6940 - val_accuracy: 0.4
970
Epoch 17/20
313/313 [==============================] - 3s 11ms/step - loss: 0.6951 - accuracy: 0.4933 - val_loss: 0.6952 - val_accuracy: 0.4
970
Epoch 18/20
313/313 [==============================] - 4s 12ms/step - loss: 0.6950 - accuracy: 0.5029 - val_loss: 0.6932 - val_accuracy: 0.4
970
Epoch 19/20
313/313 [==============================] - 4s 12ms/step - loss: 0.6948 - accuracy: 0.4998 - val_loss: 0.6932 - val_accuracy: 0.4
720
Epoch 20/20
313/313 [==============================] - 4s 12ms/step - loss: 0.6948 - accuracy: 0.5021 - val_loss: 0.6961 - val_accuracy: 0.5
030
```

In [197…
```
#Plotting the model
```

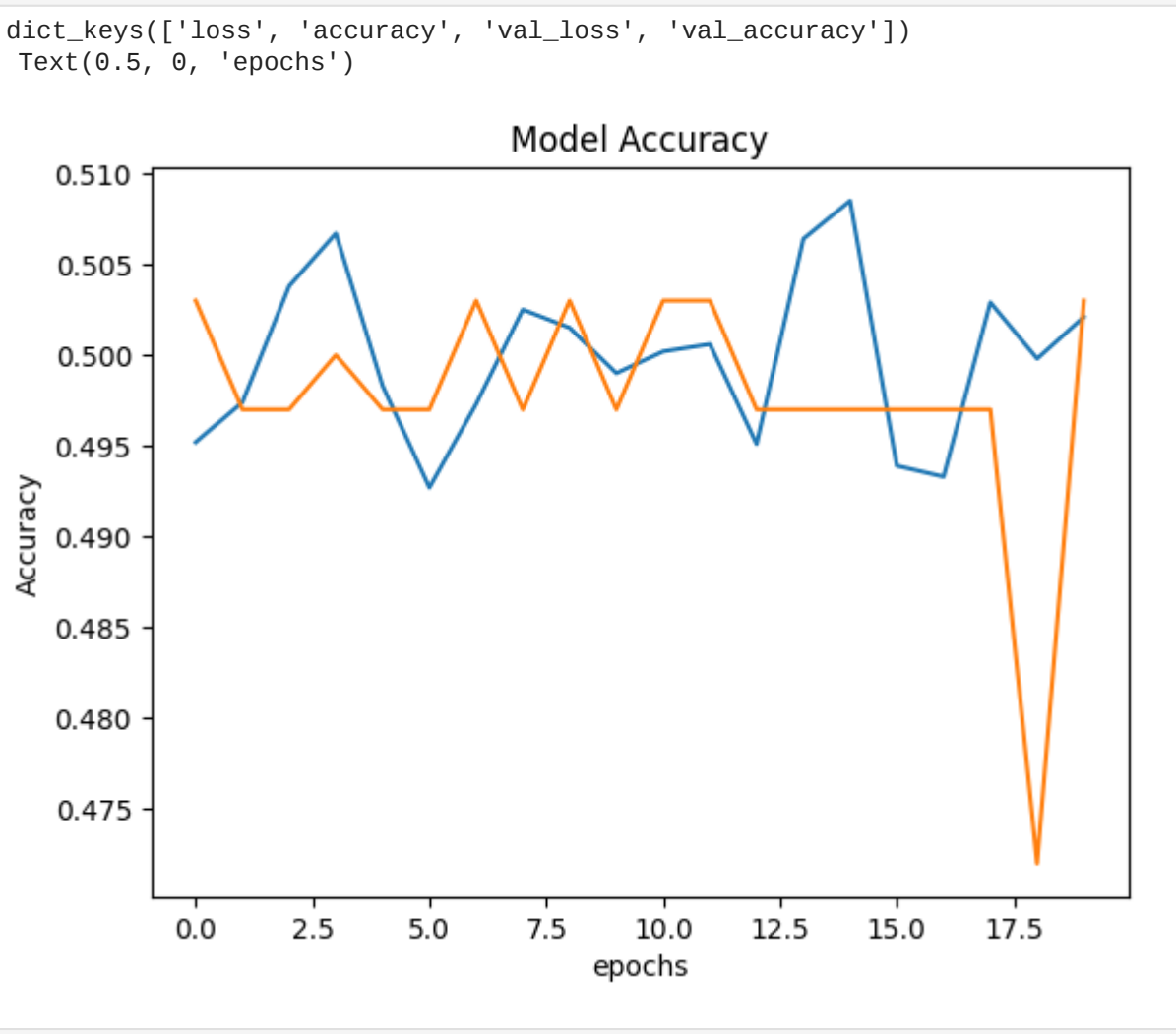
In [199…
```
import matplotlib.pyplot as plt
print(history.history.keys())
plt.plot(history.history['accuracy'])

plt.plot(history.history['val_accuracy'])


plt.title("Model Accuracy")
plt.ylabel("Accuracy")
plt.xlabel("epochs")
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

Out[199]:
```
Text(0.5, 0, 'epochs')
```



In [200…
```
#Save the model
model.save("ChatBot_model")
```

```
WARNING:absl:Found untraced functions such as _update_step_xla, lstm_cell_1_layer_call_fn, lstm_cell_1_layer_call_and_return_con
ditional_losses while saving (showing 3 of 3). These functions will not be directly callable after loading.
INFO:tensorflow:Assets written to: ChatBot_model\assets
INFO:tensorflow:Assets written to: ChatBot_model\assets
```

In [201…
```
#Evaluation on the Test set
model.load_weights("ChatBot_model")
```

Out[201]:
```
<tensorflow.python.checkpoint.checkpoint.CheckpointLoadStatus at 0x22d49113040>
```

In [202…
```
pred_results = model.predict(([inputs_test, queries_test]))
```

```
32/32 [==============================] - 1s 6ms/step
```

In [203…
```
test_data[0][0]
```

Out[203]:
```
['Mary',
 'got',
 'the',
 'milk',
 'there',
 '.',
 'John',
 'moved',
 'to',
 'the',
 'bedroom',
 '.']
```

In [204…
```
story = ' '.join(word for word in test_data[0][0])
```

In [205…
```
story
```

Out[205]:
```
'Mary got the milk there . John moved to the bedroom .'
```

In [206…
```
query = ' '.join(word for word in test_data[0][1])
```

In [207…
```
query
```

Out[207]:
```
'Is John in the kitchen ?'
```

In [208…
```
test_data[0][2]
```

Out[208]:
```
'no'
```

In [209…
```
story = ' '.join(word for word in test_data[10][0])
```

In [210…
```
story
```

Out[210]:
```
'John moved to the hallway . Sandra went to the bedroom .'
```

In [211…
```
query = ' '.join(word for word in test_data[10][1])
```

In [212…
```
query
```

Out[212]:
```
'Is John in the hallway ?'
```

In [213…
```
test_data[10][2]
```

Out[213]:
```
'yes'
```

In [214…
```
story = ' '.join(word for word in test_data[15][0])
story
```

Out[214]:
```
'John journeyed to the hallway . John got the apple there .'
```

In [215…
```
query = ' '.join(word for word in test_data[15][1])
query
```

Out[215]:
```
'Is John in the hallway ?'
```

In [216…
```
test_data[15][2]
```

Out[216]:
```
'yes'
```

In [217…
```
story = ' '.join(word for word in test_data[23][0])
story

query = ' '.join(word for word in test_data[23][1])
query

test_data[23][2]
```

Out[217]:
```
'no'
```

In [218…
```
#Generating predictions from model
val_max = np.argmax(pred_results[23])

for key, val in tokenizer.word_index.items():
    if val == val_max:
        k = key

print("Predicted Answer is", k)
print("Probability of Certainity", pred_results[23][val_max])
```

```
Predicted Answer is no
Probability of Certainity 0.5409763
```

In [219…
```
val_max = np.argmax(pred_results[30])

for key, val in tokenizer.word_index.items():
    if val == val_max:
        k = key

print("Predicted Answer is", k)
print("Probability of Certainity", pred_results[30][val_max])
```

```
Predicted Answer is no
Probability of Certainity 0.5412144
```

Step-6(Creating story and evaluating)

In [276…
```
vocab
```

Out[276]:
```
{'.',
 '?',
 'Daniel',
 'Is',
 'John',
 'Mary',
 'Sandra',
 'apple',
 'back',
 'bathroom',
 'bedroom',
 'discarded',
 'down',
 'dropped',
 'football',
 'garden',
 'got',
 'grabbed',
 'hallway',
 'in',
 'journeyed',
 'kitchen',
 'left',
 'milk',
 'moved',
 'no',
 'office',
 'picked',
 'put',
 'the',
 'there',
 'to',
 'took',
 'travelled',
 'up',
 'went',
 'yes'}
```

In [277…
```
story = "Sandra dropped milk in kitchen . Mary picked football . Daniel went to office "
story.split()
```

Out[277]:
```
['Sandra',
 'dropped',
 'milk',
 'in',
 'kitchen',
 '.',
 'Mary',
 'picked',
 'football',
 '.',
 'Daniel',
 'went',
 'to',
 'office']
```

In [278…
```
my_question = " Is Daniel in the office ? "
my_question.split()
```

Out[278]:
```
['Is', 'Daniel', 'in', 'the', 'office', '?']
```

In [284…
```
mydata = [(story.split(),my_question.split(), 'yes')]
```

In [285…
```
my_story, my_ques, my_ans = vectorize_stories(mydata)
```

In [286…
```
pred_results = model.predict((my_story, my_ques))
```

```
1/1 [==============================] - 0s 40ms/step
```

In [287…
```
val_max = np.argmax(pred_results[0])

for key, val in tokenizer.word_index.items():
    if val == val_max:
        k = key

print("Predicted Answer is", k)
print("Probability of Certainity", pred_results[0][val_max])
```

```
Predicted Answer is no
Probability of Certainity 0.54136765
```

#END OF PROJECT