The BABI Chatbot

- INTERNSHIP PROJECT
- **Project report by:**

B. Shiva Krishna

INDEX

TABLE OF CONTENTS

1. ABSTRACT

The Project report aims to develop bAbI chatbot of Facebook AI Research which is organized towards the goal of automatic text understanding and reasoning. Chatbots are frequently used to improve the IT service management experience, which delves towards self-service and automating processes offered to internal staff.

A chatbot is generally created using LSTM -A RNN(Recurrent Neural Network) model. RNN is a special kind of neural network that is designed to effectively deal with Sequential Data. LSTMs (Long-Short Term Memory) are variants of RNN that solve Long Term Memory of the former. They mitigate the problems of exploding and vanishing gradients.

2. OBJECTIVES

The main aim of the project is to build a chatbot. Chatbots boost operational efficiency and bring cost savings to businesses while offering convenience and added services to internal employees and external customers.

Basically chatbots are LSTM models that have a more complex cell structure than normal recurrent neuron. These cells allow them to better regulate how to learn or forget from different input sources.

Chatbots allow businesses to connect with customers in a personal way without the expense of human representatives. For example, many of questions or issues customers have are common and easily answered. That’s why companies create FAQs and troubleshooting guides. They provide a personal alternative to a written FAQ or guide and can even answer customers simple queries. These days chatbots have become more popular as a time and money saver for businesses and an added convenience for customers.

3. INTRODUCTION

The origin of the chatbot arguably lies with Alan Turing’s 1950s vision of intelligent machines. Artificial Intelligence, the foundation for chatbots, has progressed since that time to include supercomputers such as IBM Watson.

The original chatbot was the phone tree to wind their customers through an automated customer service model. Enhancements in technology and the growing sophistication of AI, ML and NLP evolved this model into pop-up, live, onscreen chats. And the evolutionary journey has started.

Creating a chatbot is similar to creating a mobile application and requires a messaging platform or service for delivery.

4. METHODOLOGY

The Facebook bAbI has certain tasks. The aim is that each task tests a unique aspect of text and reasoning, and hence test different capabilities of learning models.

The data-set comes already separated into **training data (10k instances)** and **test data (1k instances)**, where each instance has a **fact**, a **question**, and a yes/no **answer** to that question.

Training Set Size: For each task, there are 1000 questions for training, and 1000 for testing.

The file format for each task is as follows:

ID text ID text ID text ID question[tab]answer[tab]supporting fact IDS. ...

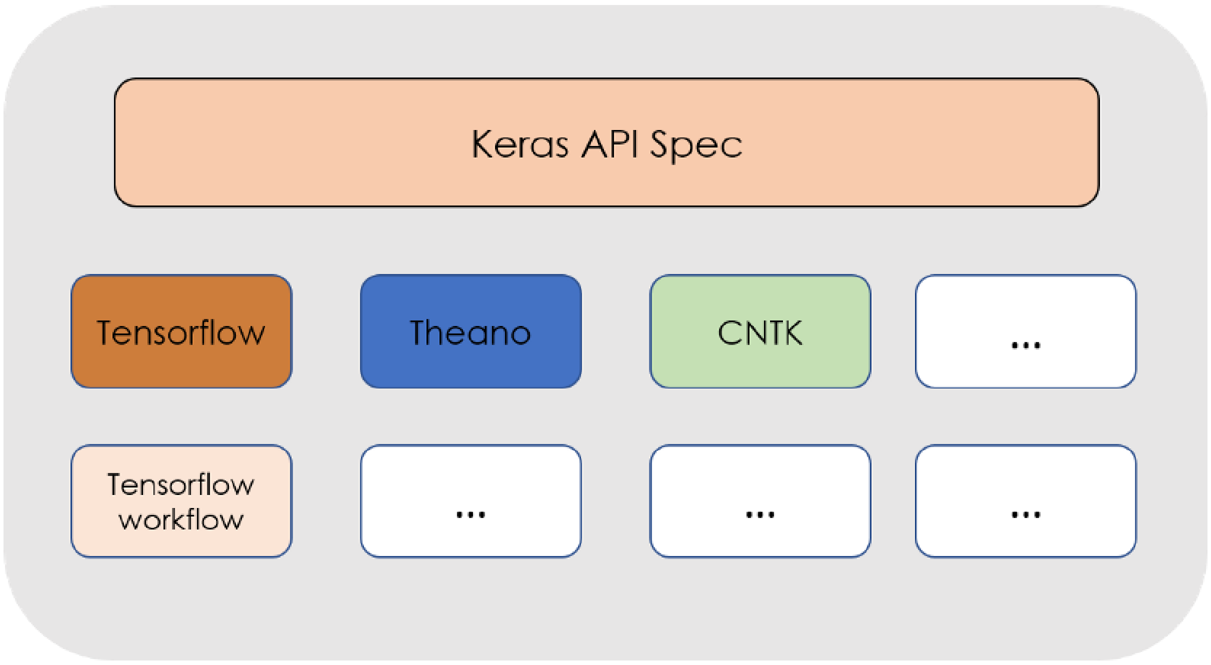
Example :
1 Mary moved to the bathroom.
2 John went to the hallway.
3 Where is Mary? bathroom
4 Daniel went back to the hallway.
5 Sandra moved to the garden.
6 Where is Daniel? hallway

Here lines 1,2,4,5 represent story, 3,6 represent questions related to that story and their respected answers. The aim of chatbot is to predict the correct answer for such questions in the test data set.

In order to create and implement chatbot. Python offers a wide range of tools.

1. **Keras** is an open source, high level library for developing neural network models built and owned by Google. Its core principle is to make the process of building a neural network, training it, and then make predictions.

Basically, keras is just an interface that can run on top of different Deep Learning Frameworks like TensorFlow, Theano etc.



The **steps for creating a Keras model** are the following:

Step 1: First we must define a network model, which most of the time will be the ***Sequential* model:** the network will be defined as a sequence of layers, each with its own customisable size and activation function.

Step 2: After creating the structure of the network in this manner we have to **compile it**, which transforms the simple sequence of layers that we have previously defined, into a complex group of matrix operations that dictate how the network behaves.

Step 3: Once this is done, we can *train* or ***fit*** the network, which is done using the back-propagation algorithm.

Building the model:

The first step to creating the network is to create what in Keras is known as ***placeholders*** for the inputs, which in this case are the **stories** and the **questions**. In an easy manner, these placeholders are containers where batches of training data will be placed before being fed to the model.

The next step is to create **Embeddings.** An embedding turns an integer number (in this case the index of a word) into a ***d*** dimensional vector, where **context is taken into account**. Word embeddings are **widely used in NLP** .

Once the embeddings are created for input sentences and questions. We need to compute the attention by doing the **dot product** between the embedding of the questions and one of the embeddings of the stories, and then doing a softmax.

Then the output ***o*** is calculated by adding the match matrix with the second input vector sequence, and then the **response** is calculated using this output and encoded question.

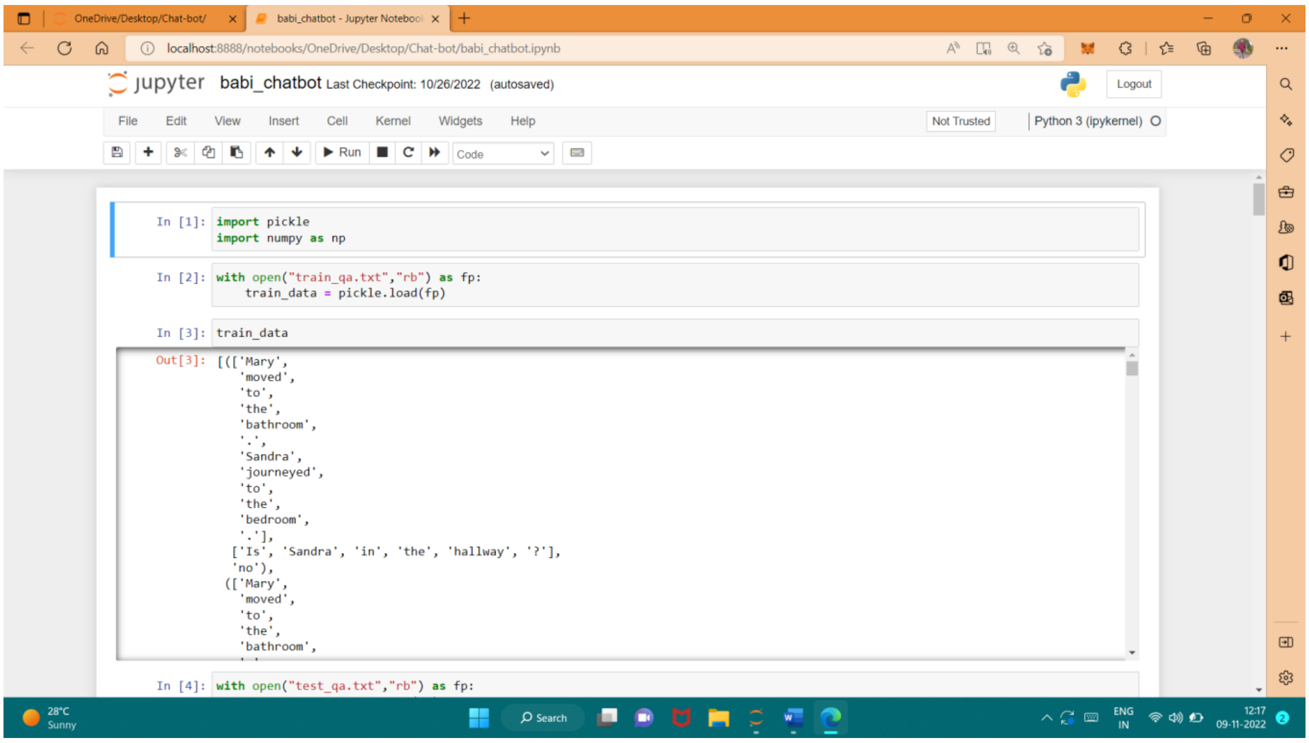
Lastly, once this is done, we add the rest of the layers of the model, adding an **LSTM** layer, a dropout layer and a final softmax to compute the output.

Now it's **time to train the model**, here we need to define the **inputs to the training**, (the input stories, questions and answers), the **batch size** that we will be feeding the model with (that is, how many inputs at once), and the **number of epochs** that we will train the model for (that is, how many times the model will go through the training data in order to update the weights). Here I have used 150 epochs and obtained an accuracy of around 86%.

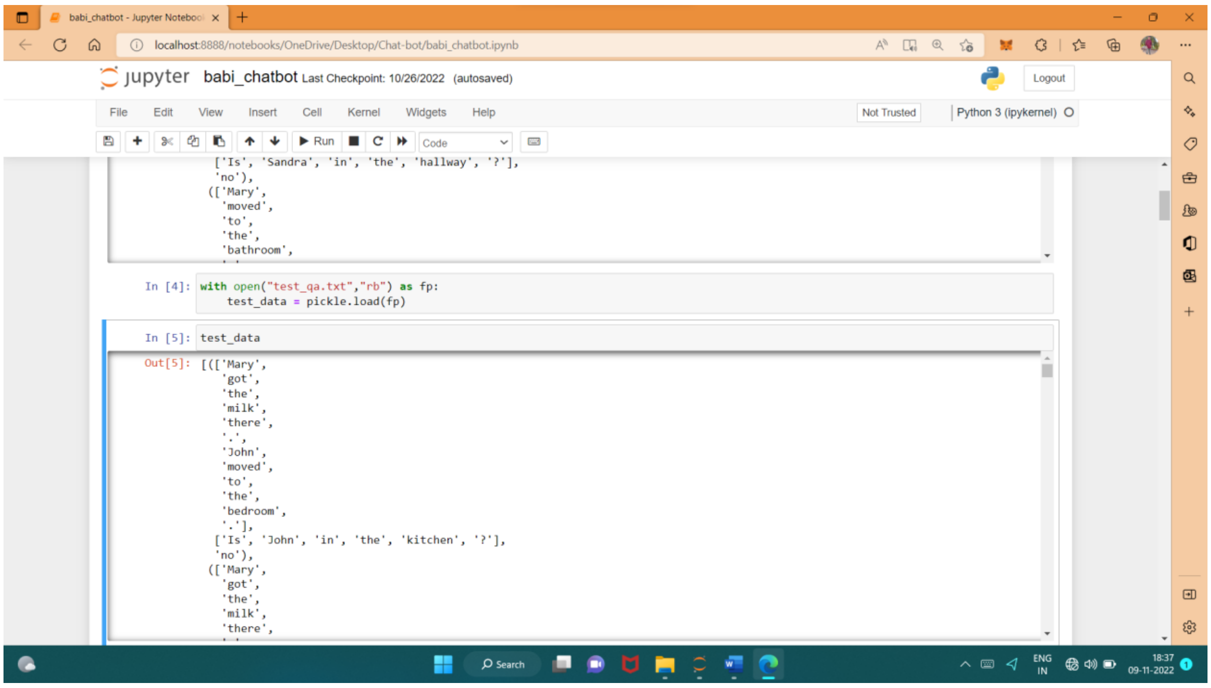
5. CODE

```
## import libraries
import pickle
import numpy as np
from keras_preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer
from keras.models import Sequential, Model
from keras.layers import Embedding
from keras.layers import Input, Activation, Dense, Permute, Dropout, add, dot, concatenate, LSTM
import matplotlib.pyplot as plt

## import dataset
with open("train_qa.txt","rb") as fp:
    train_data = pickle.load(fp)
print(train_data)
with open("test_qa.txt","rb") as fp:
    test_data = pickle.load(fp)
print(test_data)

## analysis of data set
## 1. type of data set
print(type(train_data))
print(type(test_data))
## 2. size
print(len(train_data))
print(len(test_data))

train_data[0] # gives story qsn and answer

' '.join(train_data[0][0]) ## story

' '.join(train_data[0][1]) ## question

train_data[0][2] ## answer

## setup Vocabulary
vocab = set()
all_data = test_data+train_data
print(all_data)
print(type(all_data))

for story, question, answer in all_data:
    vocab = vocab.union(set(story))
    vocab = vocab.union(set(question))
vocab.add("yes")
vocab.add("no")
print(vocab)
print(len(vocab))

vocab_len = len(vocab)+1 ## for extra space to hold 0 for keras pad sequence
print(vocab_len)
max_story_len = max([len(data[0]) for data in all_data])
print(max_story_len)
```

```
max_qsn_len = max([len(data[1]) for data in all_data])
print(max_qsn_len)

## vectorize data set ==== covert to numerical from
tokenizer = Tokenizer(filters=[])
tokenizer.fit_on_texts(vocab)
print(tokenizer.word_index)

train_story_text =[]
train_qsn_text = []
train_ans = []
for story,question,answer in train_data:
    train_story_text.append(story)
    train_qsn_text.append(question)

train_story_seq = tokenizer.texts_to_sequences(train_story_text)
print(len(train_story_seq))

def vectorize_stories(data,word_index=tokenizer.word_index,
                      max_story_len = max_story_len,max_qsn_len = max_qsn_len):
    X =[]# story
    Xq = []## query qsn
    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_qsn_len),
           np.array(Y))
print(type(train_data))

input_train, queries_train, answers_train = vectorize_stories(train_data)
input_test, queries_test, answers_test = vectorize_stories(test_data)

print(input_train)
print(answers_train)
print(tokenizer.word_index['yes'])
print(tokenizer.word_index['no'])

input_sequence = Input((max_story_len,))
question = Input((max_qsn_len,))

## Input Encoder m
input_encoder_m = Sequential()
input_encoder_m.add(Embedding(input_dim = vocab_len,output_dim = 64))
input_encoder_m.add(Dropout(0.3))

## Input Encoder c
input_encoder_c = Sequential()
input_encoder_c.add(Embedding(input_dim = vocab_len,output_dim = max_qsn_len))
input_encoder_c.add(Dropout(0.3))

## question Encoder
question_encoder = Sequential()
question_encoder.add(Embedding(input_dim = vocab_len,output_dim = 64, input_length =
max_qsn_len))
question_encoder.add(Dropout(0.3))

## Encode the sequences
input_encoded_m = input_encoder_m(input_sequence)
input_encoded_c = input_encoder_c(input_sequence)
question_encoded = question_encoder(question)

match = dot([input_encoded_m,question_encoded],axes=(2,2))
match = Activation('softmax')(match)
response = add([match,input_encoded_c])
response = Permute((2,1))(response)
## Concatenate
answer = concatenate([response,question_encoded])

## apply LSTM == Long Short Term Memory special kind of RNN
answer = LSTM(32)(answer)
answer = Dropout(0.5)(answer)
answer = Dense(vocab_len)(answer)
answer = Activation('softmax')(answer)

## build final model
model = Model([input_sequence,question],answer)
model.compile(optimizer='rmsprop',loss='categorical_crossentropy',metrics=['accuracy'])
print(model.summary())


## tarin the model
history = model.fit([input_train,queries_train],answers_train,batch_size=32,epochs=150,
                validation_data = ([input_test,queries_test],answers_test))

## Accuracy Graph
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")
plt.show()

## saving the model
model.save("my_babi_chatbot_model")

## generate predicted results
pred_results = model.predict(([input_test,queries_test]))

story = ' '.join(word for word in test_data[15][0])
print(story)
query = ' '.join(word for word in test_data[15][1])
print(query)
print(test_data[15][2])

## generate predictions from model
val_max = np.argmax(pred_results[15])
for key,val in tokenizer.word_index.items():
    if val == val_max:
        k=key
print("Predicted Answer is ",k)
print("Probability of Certainity is ",pred_results[15][val_max])
```


OneDrive/Desktop/Chat-bot/
babi_chatbot - Jupyter Notebook
localhost:8888/notebooks/OneDrive/Desktop/Chat-bot/babi_chatbot.ipynb
jupyter
babi_chatbot Last Checkpoint: 10/26/2022 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code
28°C Sunny
Search
ENG IN
12:17 09-11-2022

```
In [1]: import pickle
        import numpy as np

In [2]: with open("train_qa.txt","rb") as fp:
            train_data = pickle.load(fp)

In [3]: train_data

Out[3]: [(['Mary',
           'moved',
           'to',
           'the',
           'bathroom',
           '.',
           'Sandra',
           'journeyed',
           'to',
           'the',
           'bedroom',
           '.'],
          ['Is', 'Sandra', 'in', 'the', 'hallway', '?'],
          'no'),
         (['Mary',
           'moved',
           'to',
           'the',
           'bathroom',

In [4]: with open("test_qa.txt","rb") as fp:
```

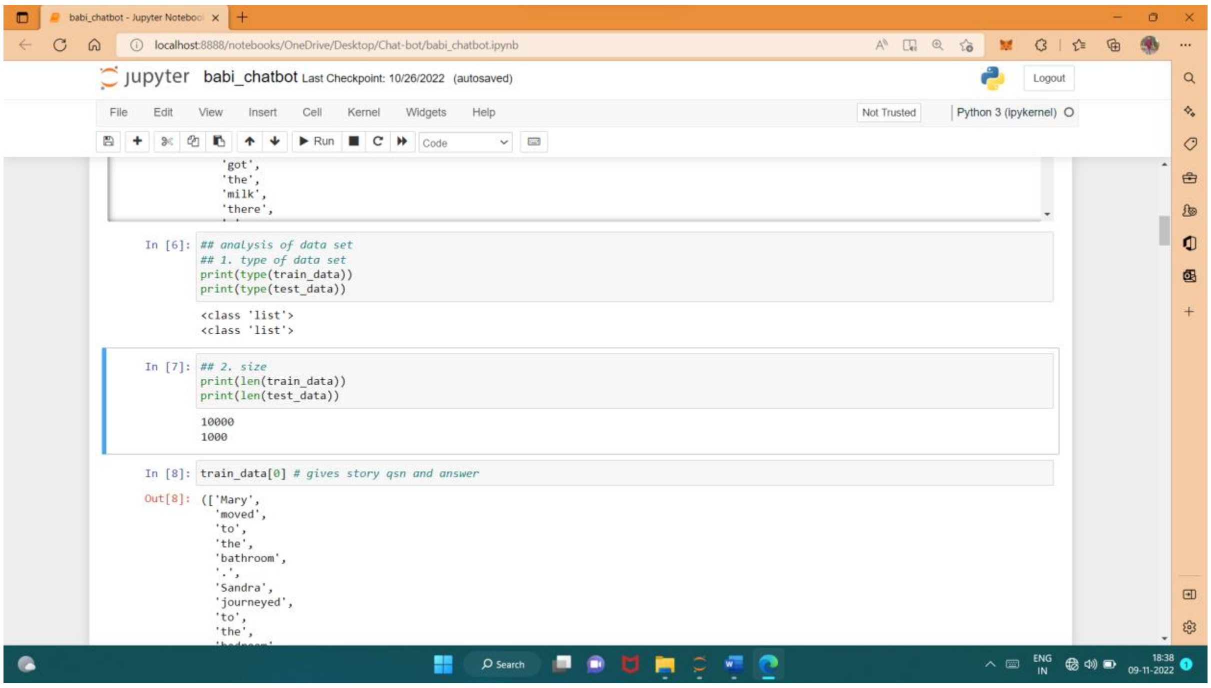


```
                ['Is', 'Sandra', 'in', 'the', 'hallway', '?'],
                'no'),
               (['Mary',
                 'moved',
                 'to',
                 'the',
                 'bathroom',
```

```
In [4]: with open("test_qa.txt","rb") as fp:
            test_data = pickle.load(fp)
```

```
In [5]: test_data
```

```
Out[5]: [(['Mary',
           'got',
           'the',
           'milk',
           'there',
           '.',
           'John',
           'moved',
           'to',
           'the',
           'bedroom',
           '.'],
          ['Is', 'John', 'in', 'the', 'kitchen', '?'],
          'no'),
         (['Mary',
           'got',
           'the',
           'milk',
           'there',
```



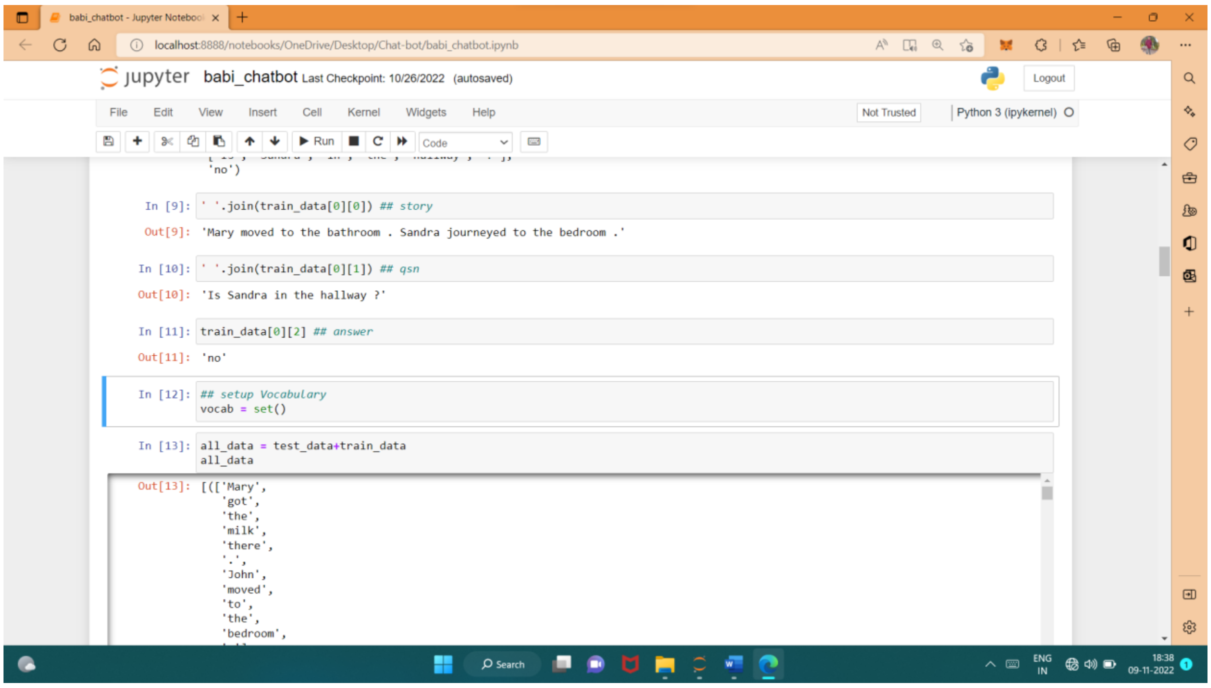
```
           'got',
           'the',
           'milk',
           'there',
```

```
In [6]: ## analysis of data set
        ## 1. type of data set
        print(type(train_data))
        print(type(test_data))

        <class 'list'>
        <class 'list'>
```

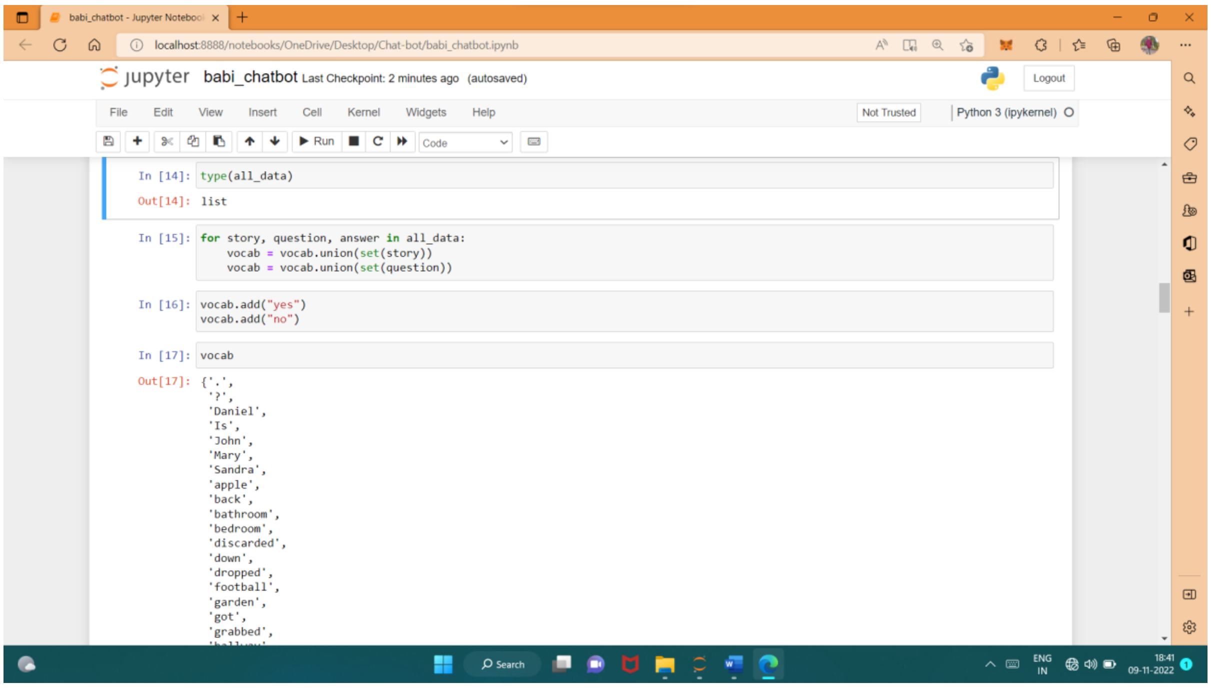
```
In [7]: ## 2. size
        print(len(train_data))
        print(len(test_data))

        10000
        1000
```

```
In [8]: train_data[0] # gives story qsn and answer
```

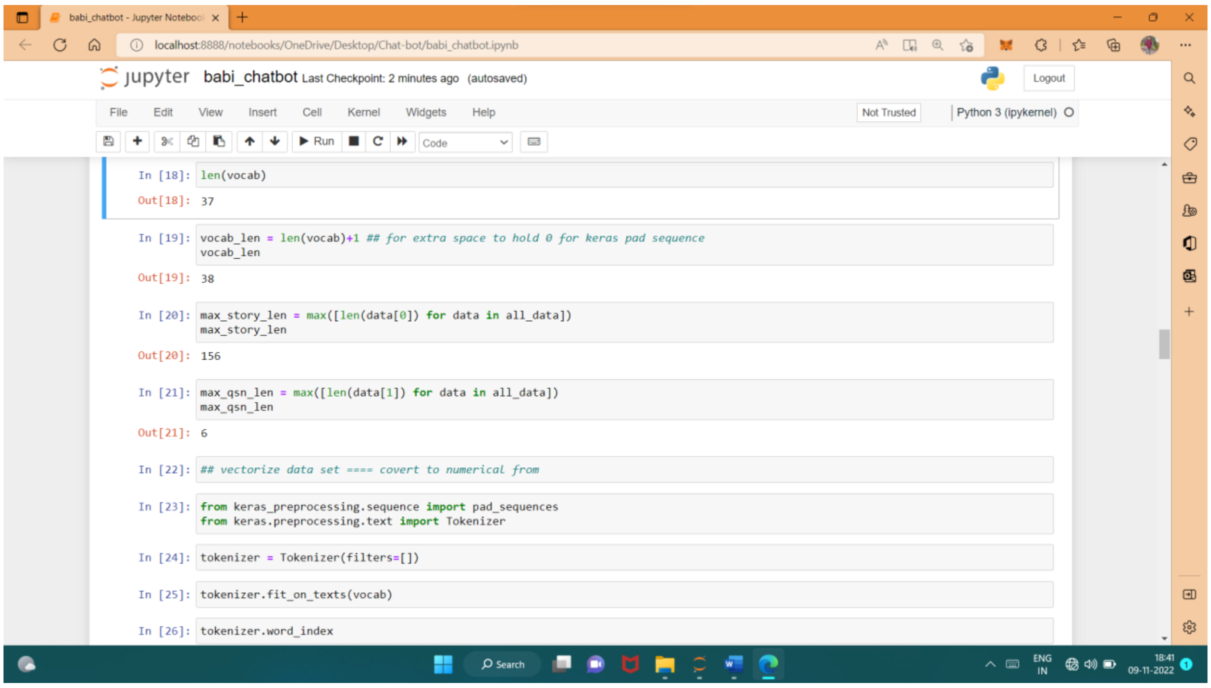
```
Out[8]: (['Mary',
          'moved',
          'to',
          'the',
          'bathroom',
          '.',
          'Sandra',
          'journeyed',
          'to',
          'the',
```



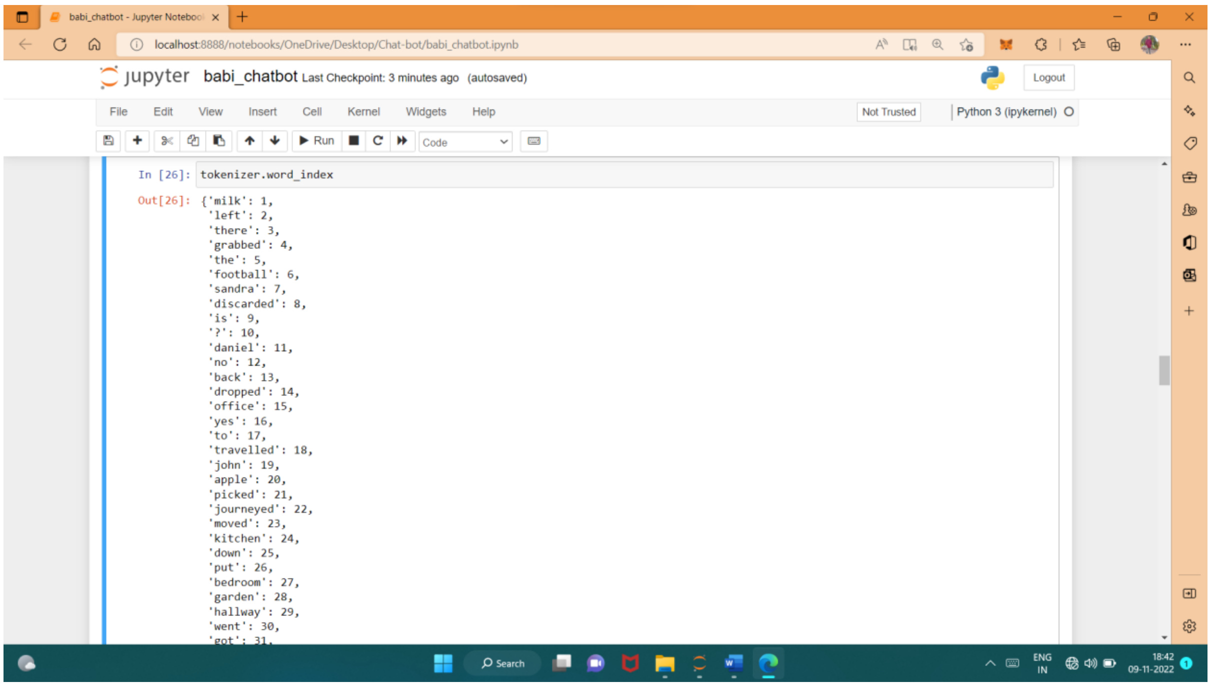
```
 'no')

In [9]: ' '.join(train_data[0][0]) ## story
Out[9]: 'Mary moved to the bathroom . Sandra journeyed to the bedroom .'

In [10]: ' '.join(train_data[0][1]) ## qsn
Out[10]: 'Is Sandra in the hallway ?'

In [11]: train_data[0][2] ## answer
Out[11]: 'no'

In [12]: ## setup Vocabulary
         vocab = set()

In [13]: all_data = test_data+train_data
         all_data

Out[13]: [(['Mary',
            'got',
            'the',
            'milk',
            'there',
            '.',
            'John',
            'moved',
            'to',
            'the',
            'bedroom',
```



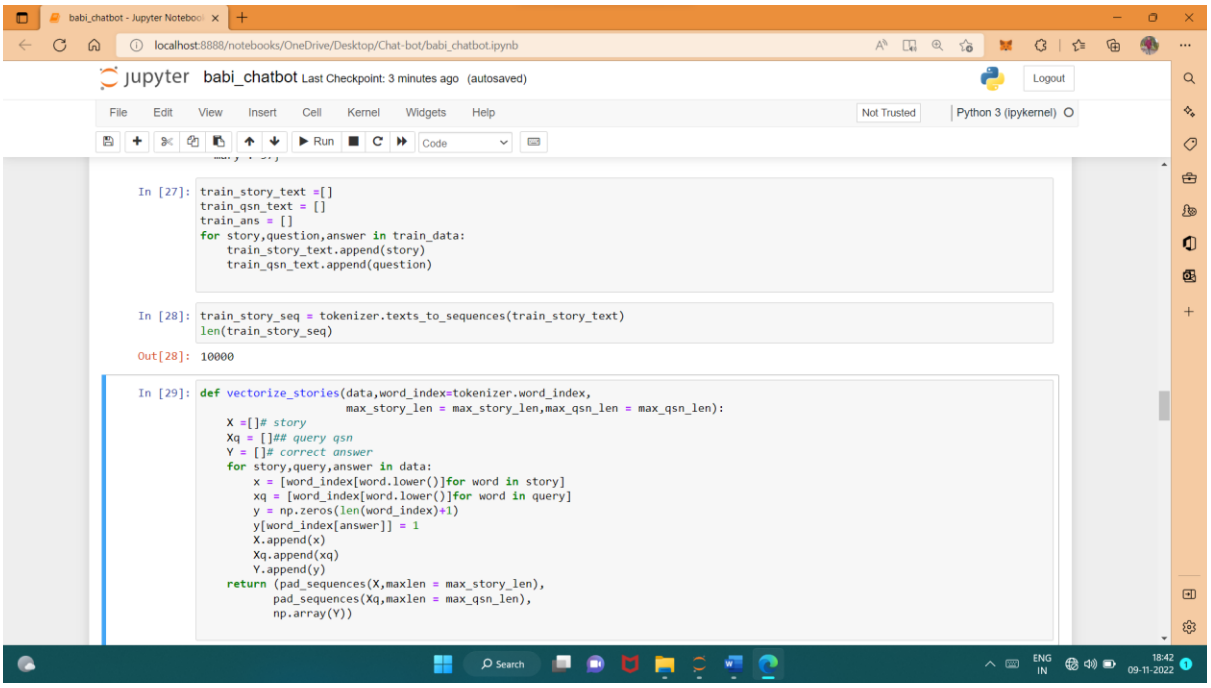
```
In [14]: type(all_data)
Out[14]: list

In [15]: for story, question, answer in all_data:
             vocab = vocab.union(set(story))
             vocab = vocab.union(set(question))

In [16]: vocab.add("yes")
         vocab.add("no")

In [17]: vocab

Out[17]: {'.',
          '?',
          'Daniel',
          'Is',
          'John',
          'Mary',
          'Sandra',
          'apple',
          'back',
          'bathroom',
          'bedroom',
          'discarded',
          'down',
          'dropped',
          'football',
          'garden',
          'got',
          'grabbed',
```



```
In [18]: len(vocab)
Out[18]: 37

In [19]: vocab_len = len(vocab)+1 ## for extra space to hold 0 for keras pad sequence
         vocab_len
Out[19]: 38

In [20]: max_story_len = max([len(data[0]) for data in all_data])
         max_story_len
Out[20]: 156

In [21]: max_qsn_len = max([len(data[1]) for data in all_data])
         max_qsn_len
Out[21]: 6

In [22]: ## vectorize data set ==== covert to numerical from

In [23]: from keras_preprocessing.sequence import pad_sequences
         from keras.preprocessing.text import Tokenizer

In [24]: tokenizer = Tokenizer(filters=[])

In [25]: tokenizer.fit_on_texts(vocab)

In [26]: tokenizer.word_index
```

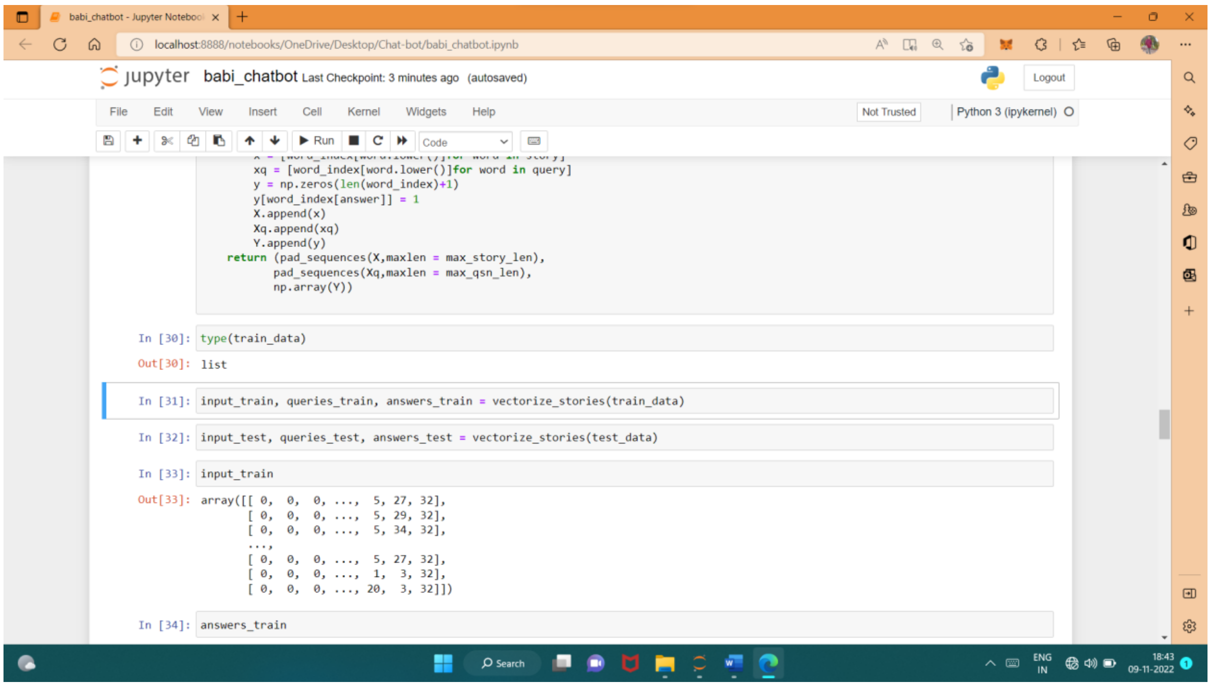


```
In [26]: tokenizer.word_index
Out[26]: {'milk': 1,
          'left': 2,
          'there': 3,
          'grabbed': 4,
          'the': 5,
          'football': 6,
          'sandra': 7,
          'discarded': 8,
          'is': 9,
          '?': 10,
          'daniel': 11,
          'no': 12,
  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

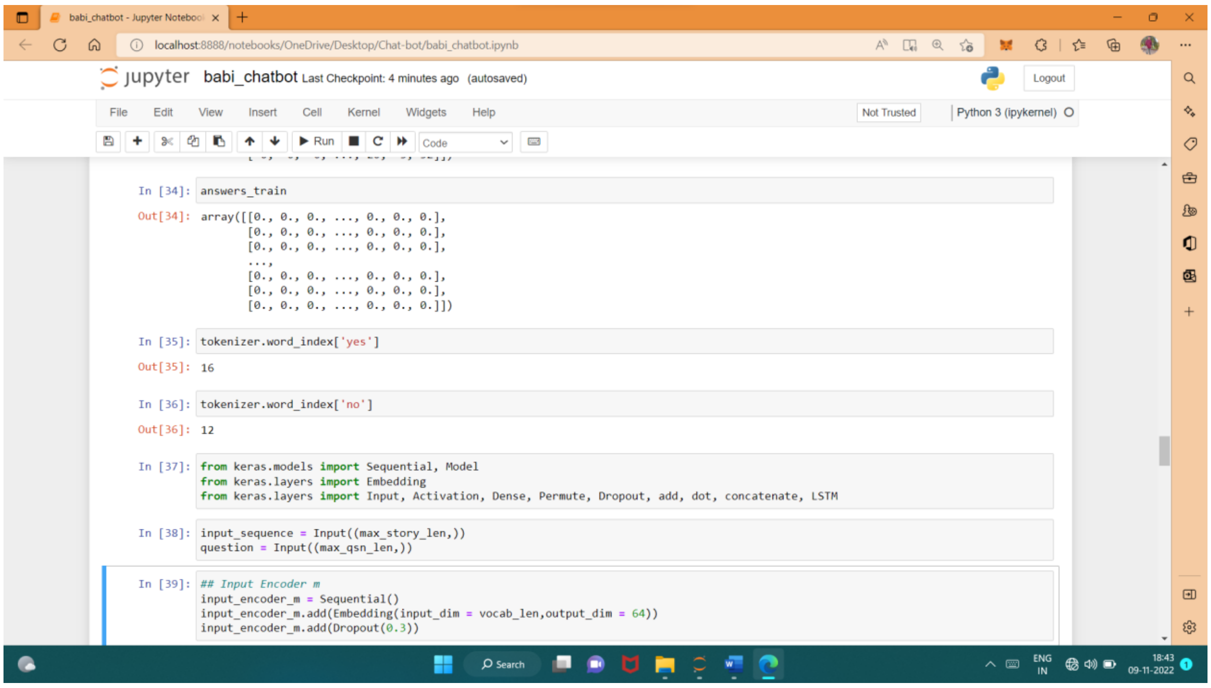

babi_chatbot - Jupyter Notebook
localhost:8888/notebooks/OneDrive/Desktop/Chat-bot/babi_chatbot.ipynb
jupyter babi_chatbot Last Checkpoint: 3 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code

```
In [27]: train_story_text =[]
         train_qsn_text = []
         train_ans = []
         for story,question,answer in train_data:
             train_story_text.append(story)
             train_qsn_text.append(question)

In [28]: train_story_seq = tokenizer.texts_to_sequences(train_story_text)
         len(train_story_seq)

Out[28]: 10000

In [29]: def vectorize_stories(data,word_index=tokenizer.word_index,
                               max_story_len = max_story_len,max_qsn_len = max_qsn_len):
             X =[]# story
             Xq = []## query qsn
             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_qsn_len),
                    np.array(Y))
```

Search
ENG
IN
18:42
09-11-2022

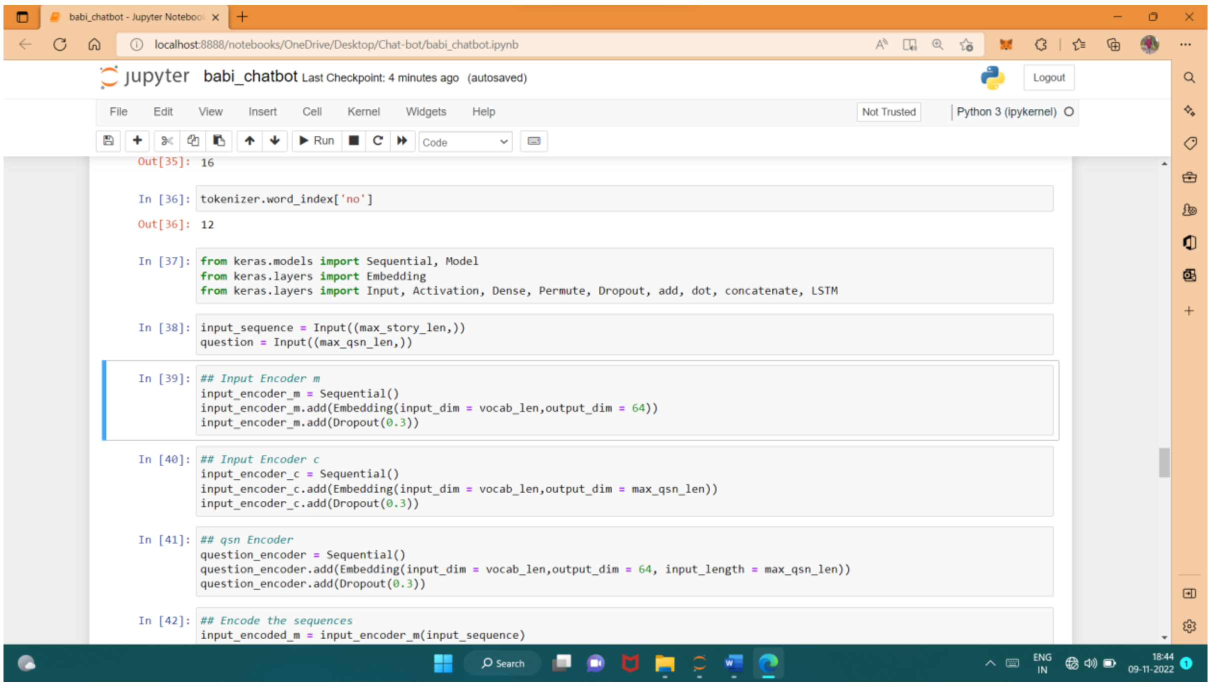

babi_chatbot - Jupyter Notebook
localhost:8888/notebooks/OneDrive/Desktop/Chat-bot/babi_chatbot.ipynb
jupyter babi_chatbot Last Checkpoint: 3 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code

```
                 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_qsn_len),
                    np.array(Y))

In [30]: type(train_data)

Out[30]: list

In [31]: input_train, queries_train, answers_train = vectorize_stories(train_data)

In [32]: input_test, queries_test, answers_test = vectorize_stories(test_data)

In [33]: input_train

Out[33]: array([[ 0,  0,  0, ...,  5, 27, 32],
                [ 0,  0,  0, ...,  5, 29, 32],
                [ 0,  0,  0, ...,  5, 34, 32],
                ...,
                [ 0,  0,  0, ...,  5, 27, 32],
                [ 0,  0,  0, ...,  1,  3, 32],
                [ 0,  0,  0, ..., 20,  3, 32]])

In [34]: answers_train
```

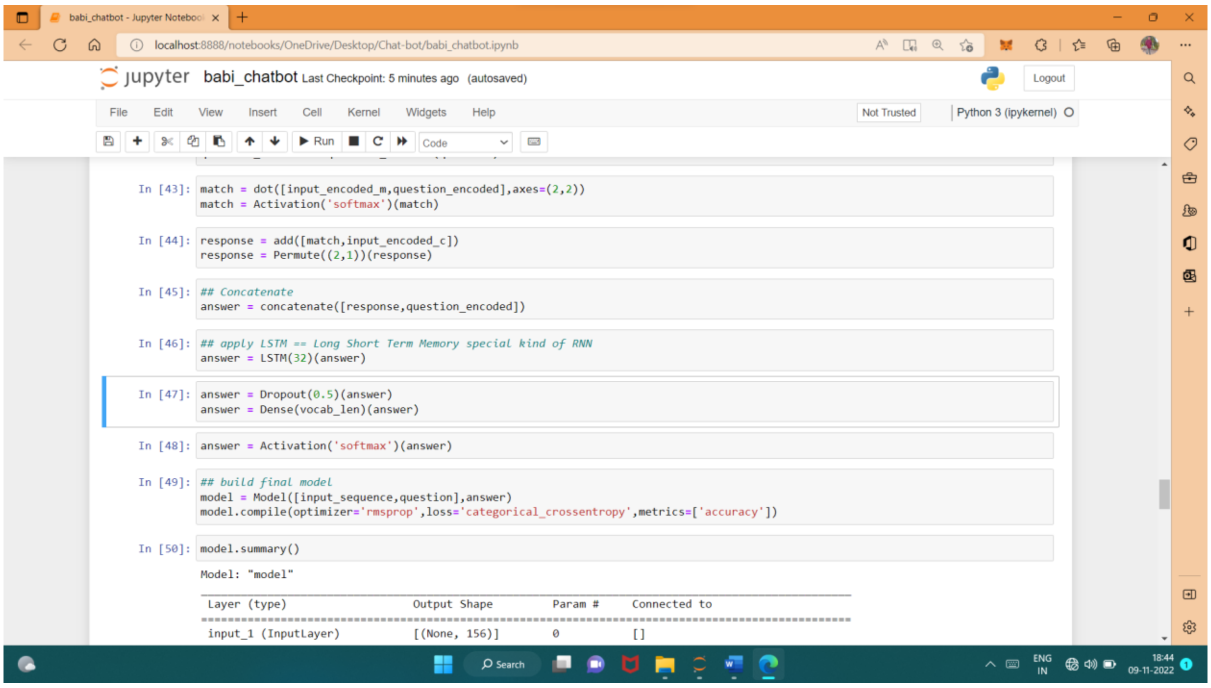

babi_chatbot - Jupyter Notebook
localhost:8888/notebooks/OneDrive/Desktop/Chat-bot/babi_chatbot.ipynb
jupyter babi_chatbot Last Checkpoint: 4 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code

```
In [34]: answers_train

Out[34]: 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 [35]: tokenizer.word_index['yes']

Out[35]: 16

In [36]: tokenizer.word_index['no']

Out[36]: 12

In [37]: from keras.models import Sequential, Model
         from keras.layers import Embedding
         from keras.layers import Input, Activation, Dense, Permute, Dropout, add, dot, concatenate, LSTM

In [38]: input_sequence = Input((max_story_len,))
         question = Input((max_qsn_len,))

In [39]: ## Input Encoder m
         input_encoder_m = Sequential()
         input_encoder_m.add(Embedding(input_dim = vocab_len,output_dim = 64))
         input_encoder_m.add(Dropout(0.3))
```

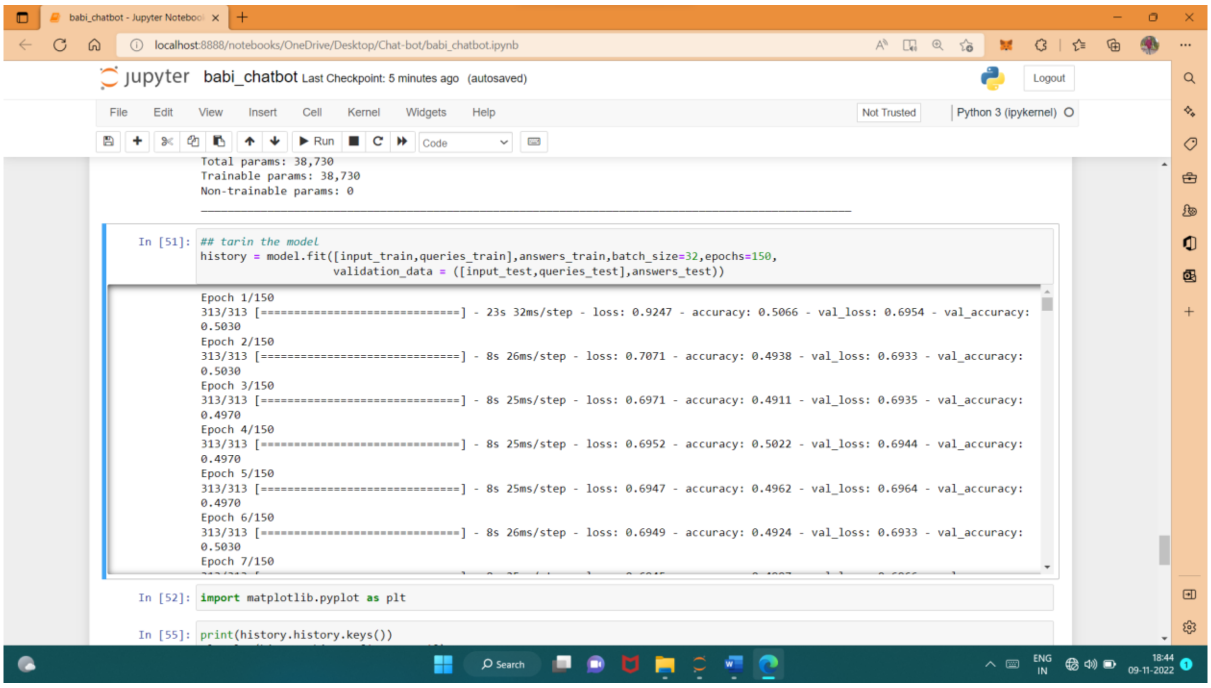
Search
ENG IN
18:43
09-11-2022


babi_chatbot - Jupyter Notebook
localhost:8888/notebooks/OneDrive/Desktop/Chat-bot/babi_chatbot.ipynb
jupyter babi_chatbot Last Checkpoint: 4 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code

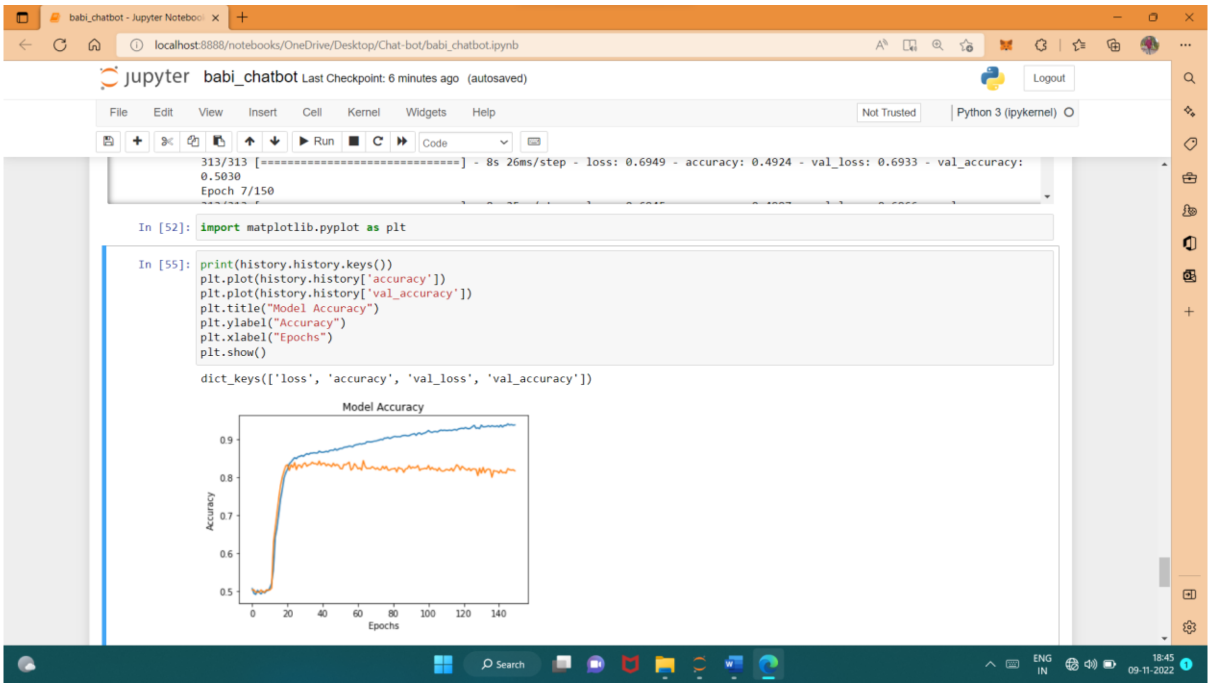
```
Out[35]: 16

In [36]: tokenizer.word_index['no']

Out[36]: 12

In [37]: from keras.models import Sequential, Model
         from keras.layers import Embedding
         from keras.layers import Input, Activation, Dense, Permute, Dropout, add, dot, concatenate, LSTM

In [38]: input_sequence = Input((max_story_len,))
         question = Input((max_qsn_len,))

In [39]: ## Input Encoder m
         input_encoder_m = Sequential()
         input_encoder_m.add(Embedding(input_dim = vocab_len,output_dim = 64))
         input_encoder_m.add(Dropout(0.3))

In [40]: ## Input Encoder c
         input_encoder_c = Sequential()
         input_encoder_c.add(Embedding(input_dim = vocab_len,output_dim = max_qsn_len))
         input_encoder_c.add(Dropout(0.3))

In [41]: ## qsn Encoder
         question_encoder = Sequential()
         question_encoder.add(Embedding(input_dim = vocab_len,output_dim = 64, input_length = max_qsn_len))
         question_encoder.add(Dropout(0.3))

In [42]: ## Encode the sequences
         input_encoded_m = input_encoder_m(input_sequence)
```



```
In [43]: match = dot([input_encoded_m,question_encoded],axes=(2,2))
         match = Activation('softmax')(match)

In [44]: response = add([match,input_encoded_c])
         response = Permute((2,1))(response)

In [45]: ## Concatenate
         answer = concatenate([response,question_encoded])

In [46]: ## apply LSTM == Long Short Term Memory special kind of RNN
         answer = LSTM(32)(answer)

In [47]: answer = Dropout(0.5)(answer)
         answer = Dense(vocab_len)(answer)

In [48]: answer = Activation('softmax')(answer)

In [49]: ## build final model
         model = Model([input_sequence,question],answer)
         model.compile(optimizer='rmsprop',loss='categorical_crossentropy',metrics=['accuracy'])

In [50]: model.summary()

         Model: "model"
         __________________________________________________________________________________________
          Layer (type)                   Output Shape         Param #     Connected to
         ==========================================================================================
          input_1 (InputLayer)           [(None, 156)]        0           []
```



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

In [51]: ## tarin the model
         history = model.fit([input_train,queries_train],answers_train,batch_size=32,epochs=150,
                             validation_data = ([input_test,queries_test],answers_test))

         Epoch 1/150
         313/313 [==============================] - 23s 32ms/step - loss: 0.9247 - accuracy: 0.5066 - val_loss: 0.6954 - val_accuracy:
         0.5030
         Epoch 2/150
         313/313 [==============================] - 8s 26ms/step - loss: 0.7071 - accuracy: 0.4938 - val_loss: 0.6933 - val_accuracy:
         0.5030
         Epoch 3/150
         313/313 [==============================] - 8s 25ms/step - loss: 0.6971 - accuracy: 0.4911 - val_loss: 0.6935 - val_accuracy:
         0.4970
         Epoch 4/150
         313/313 [==============================] - 8s 25ms/step - loss: 0.6952 - accuracy: 0.5022 - val_loss: 0.6944 - val_accuracy:
         0.4970
         Epoch 5/150
         313/313 [==============================] - 8s 25ms/step - loss: 0.6947 - accuracy: 0.4962 - val_loss: 0.6964 - val_accuracy:
         0.4970
         Epoch 6/150
         313/313 [==============================] - 8s 26ms/step - loss: 0.6949 - accuracy: 0.4924 - val_loss: 0.6933 - val_accuracy:
         0.5030
         Epoch 7/150

In [52]: import matplotlib.pyplot as plt

In [55]: print(history.history.keys())
```

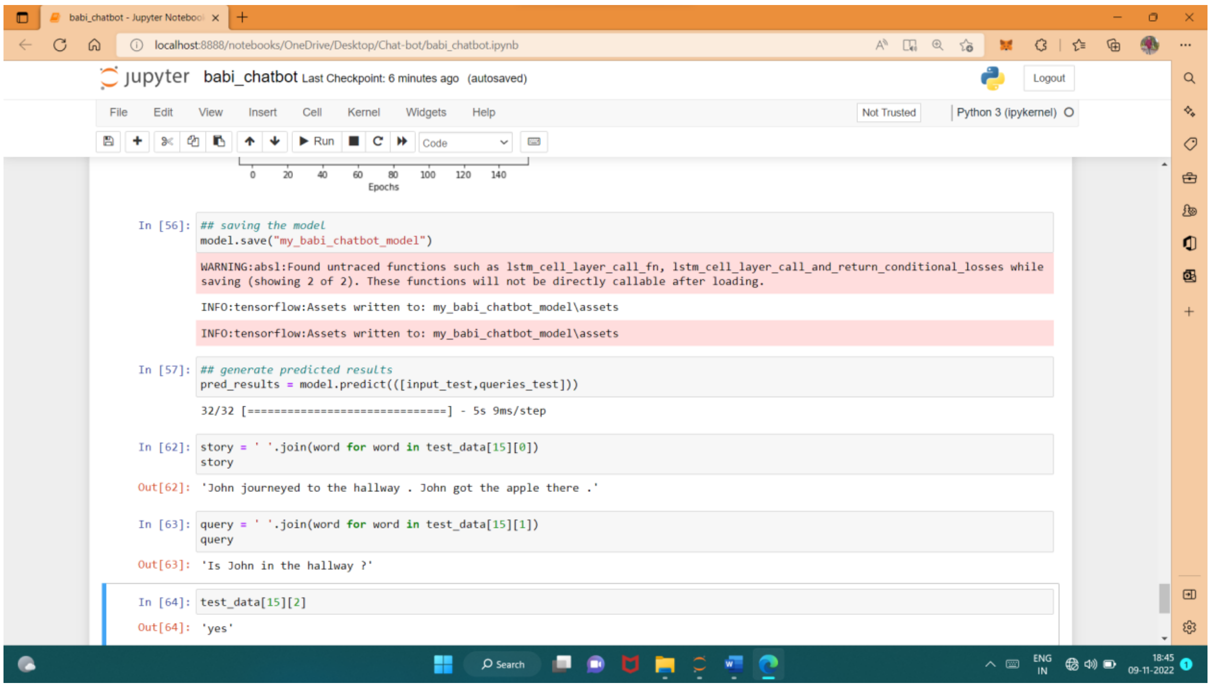

babi_chatbot - Jupyter Notebook
localhost:8888/notebooks/OneDrive/Desktop/Chat-bot/babi_chatbot.ipynb
jupyter babi_chatbot Last Checkpoint: 6 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code

```
313/313 [==============================] - 8s 26ms/step - loss: 0.6949 - accuracy: 0.4924 - val_loss: 0.6933 - val_accuracy:
0.5030
Epoch 7/150
```

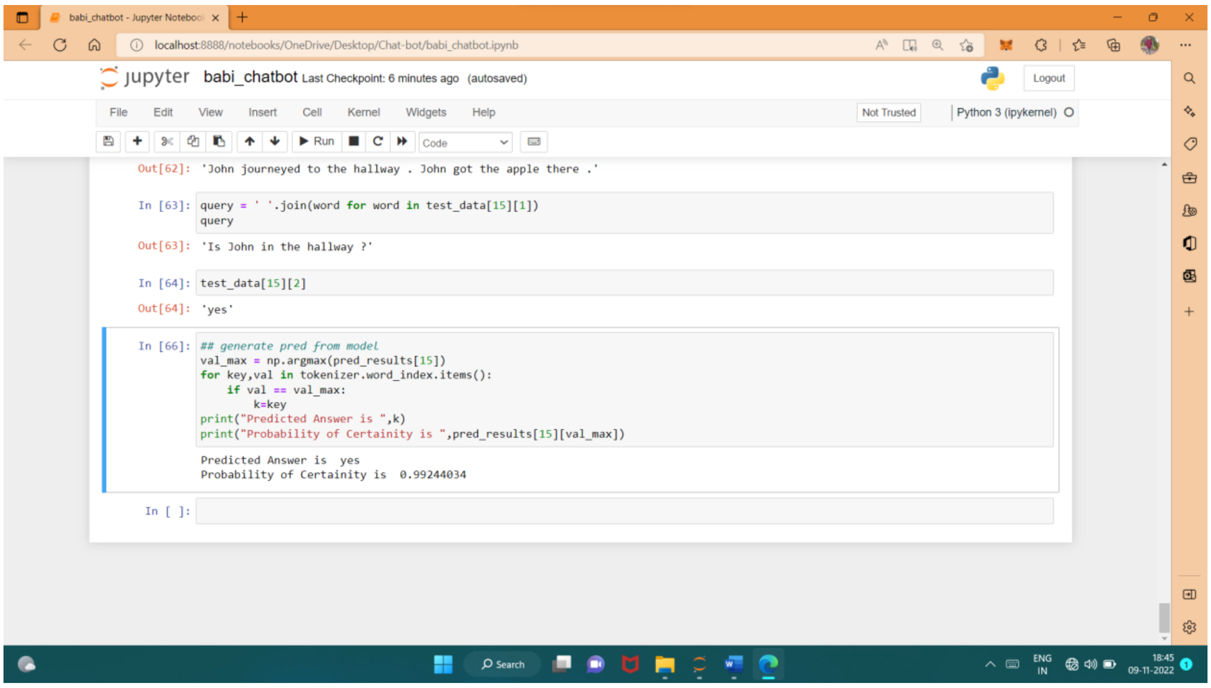
```
In [52]: import matplotlib.pyplot as plt
```

```
In [55]: 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")
         plt.show()

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

Model Accuracy
Accuracy
Epochs
0.5
0.6
0.7
0.8
0.9
0
20
40
60
80
100
120
140
Search
ENG IN
18:45
09-11-2022


babi_chatbot - Jupyter Notebook
localhost:8888/notebooks/OneDrive/Desktop/Chat-bot/babi_chatbot.ipynb
jupyter babi_chatbot Last Checkpoint: 6 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code
0
20
40
60
80
100
120
140
Epochs

```
In [56]: ## saving the model
         model.save("my_babi_chatbot_model")

         WARNING:absl:Found untraced functions such as lstm_cell_layer_call_fn, lstm_cell_layer_call_and_return_conditional_losses while
         saving (showing 2 of 2). These functions will not be directly callable after loading.

         INFO:tensorflow:Assets written to: my_babi_chatbot_model\assets

         INFO:tensorflow:Assets written to: my_babi_chatbot_model\assets

In [57]: ## generate predicted results
         pred_results = model.predict(([input_test,queries_test]))

         32/32 [==============================] - 5s 9ms/step

In [62]: story = ' '.join(word for word in test_data[15][0])
         story

Out[62]: 'John journeyed to the hallway . John got the apple there .'

In [63]: query = ' '.join(word for word in test_data[15][1])
         query

Out[63]: 'Is John in the hallway ?'

In [64]: test_data[15][2]

Out[64]: 'yes'
```


babi_chatbot - Jupyter Notebook
localhost:8888/notebooks/OneDrive/Desktop/Chat-bot/babi_chatbot.ipynb
jupyter babi_chatbot Last Checkpoint: 6 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code

```
Out[62]: 'John journeyed to the hallway . John got the apple there .'

In [63]: query = ' '.join(word for word in test_data[15][1])
         query

Out[63]: 'Is John in the hallway ?'

In [64]: test_data[15][2]

Out[64]: 'yes'

In [66]: ## generate pred from model
         val_max = np.argmax(pred_results[15])
         for key,val in tokenizer.word_index.items():
             if val == val_max:
                 k=key
         print("Predicted Answer is ",k)
         print("Probability of Certainity is ",pred_results[15][val_max])

         Predicted Answer is  yes
         Probability of Certainity is  0.99244034

In [ ]:
```

Find complete code here:
https://drive.google.com/drive/folders/1ZrHin8w2wzT1HF5SWOqjO8EgagFKdY2f?usp=sharing

6. CONCLUSION

Chatbots are conventional tools that performs routine tasks efficiently. People like them because they help them get through some tasks quickly so that they can focus their attention on high-level, strategic, and engaging activities that require human capabilities that cannot be replicated by machines.

In the near future, when AI is combined with the development of 5G technology, businesses, employees, and consumers are likely to enjoy enhanced chatbot features such as faster recommendations and predictions, and easy access to high-definition video conferencing from within a conversation. These and other possibilities are in the investigative stages and will evolve quickly as internet connectivity, AI, NLP, and ML advance.

Eventually, every person can have a fully functional personal assistant right in their pocket, making our world a more efficient and connected place to live and work.