Name: Pranjal Shivaji Thorat

Title:

ChatBot

Abstract:

This report outlines the process of creating a chatbot using the Facebook Babi Dataset. The objective is to build an intelligent dialogue agent capable of answering questions based on provided stories. The methodology involves data preprocessing, model creation, training, evaluation, and testing. The code implementation is presented along with insights into chatbot functionality and technology.

Objective:

The primary objective of this project is to develop a chatbot using the Facebook Babi Dataset to demonstrate its capability in understanding and responding to questions based on given stories. The project aims to showcase the use of natural language processing and neural networks in building an effective chatbot.

Introduction:

A chatbot is a computer program designed to facilitate communication between humans and technology. It employs various input methods such as text, voice, and gesture. Chatbots leverage technologies like Natural Language Processing (NLP) and Artificial Intelligence (AI) to provide human-like interactions. This report focuses on building a chatbot using the Facebook Babi Dataset to answer questions posed based on provided stories.

What is a Chatbot?

A Chatbot is a computer program that facilitates technological and human communication through various input methods, such as text, voice and gesture. Chatbots serve the purpose of digital assistants, virtual assistants, AI assistants and much more. The recent technological advancement like Natural Language Processing(NLP), Artificial Intelligence(AI), Data Mining and Machine Learning(ML) have resulted in the creation of advanced AI Chatbots that are useful to businesses and individuals alike.

Chatbot Functionality

Chatbot is used by enterprises to communicate within their business, with customers regarding the services rendered and so on. The Chatbot understands text by using Natural Language Processing (NLP). Natural Language Understanding (NLU) is used by chatbots to understand the language, which is combined with algorithms to give a suitable response to the supplied query. The next level in the delivery of the natural and personalized experience is achieved by Natural Language Generation (NLG).

Types of Technology for Chatbots

The technology driving today's chatbot is linguistics and machine learning. The linguistic chatbots are also known as rule based chatbots and are structured in a way that responses to queries are done in meaningful ways. These chatbots are basic and close to interactive questioning. Machine learning (AI chatbots) are complex chatbots which are data driven and use NLU to personalize answers.

How are chatbots trained?

To train AI bots, it is paramount that a huge amount of training data is fed into the system to get sophisticated services. A hybrid approach is the best solution to enterprises looking for complex chatbots. The queries which cannot be answered by AI bots can be taken care of by linguistic chatbots. The data resulting from these basic bots can then be further applied to train AI bots, resulting in the hybrid bot system.

The Facebook bAbI dataset

The bAbI project was conducted by Facebook AI research team in 2015 to solve the problem of automatic text understanding and reasoning in an intelligent dialogue agent. To make the conversation with the interface as human as possible the team developed proxy tasks that evaluate reading comprehension via question answering. The tasks are designed to measure directly how well language models can exploit wider linguistic context. For our project, the subset of Babi Data Set from Facebook Research is used.

Methodology:

The methodology involves several key steps:

1. Data Exploration: The Facebook Babi Dataset, containing stories, questions, and answers, is examined. Train and test data are prepared for further processing.

2. Vocabulary Setup: A vocabulary dictionary is established to hold unique words from the stories and questions.

3. Vectorization: The Keras library is utilized to tokenize and pad sequences, converting words into integers for model input.

4. Model Creation: A neural network model consisting of encoders and decoders is built using the Keras Sequential model. The model architecture is designed to understand and respond to questions.

5. Model Evaluation: The model's accuracy is assessed using training and testing data. The training process is monitored across epochs to visualize accuracy improvements.

6. Test Results: The trained model's performance is demonstrated by providing test stories and questions, with predictions and probabilities generated for each answer.

Code:

The Python code implementation follows these steps, utilizing libraries like Keras for model building, tokenization, and sequence padding. The code includes functions for data vectorization, model creation, and evaluation. It also showcases how to predict answers based on test stories and questions.

Conclusion:

In conclusion, this project successfully demonstrates the creation of a chatbot using the Facebook Babi Dataset. The chatbot's ability to comprehend and respond to questions based on stories is showcased. The utilization of neural networks and natural language processing technologies proves essential in achieving accurate predictions. The Facebook Babi Dataset serves as a valuable resource for training and evaluating chatbot models, highlighting the potential of AI-driven dialogue agents.

This report presents a comprehensive overview of building a chatbot and showcases the effectiveness of neural networks in achieving human-like interactions.

ChatBot

Step 1: Import required libraries and read the data files.

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 [5]:
```
with open("test_qa.txt", "rb") as fp:
    test_data = pickle.load(fp)
```

In [6]:
```
" ".join(train_data[0][2])
```

Out[6]:
```
'n o'
```

In [7]:
```
test_data
```

Out[7]:
```
[(['Mary',
   'got',
   'the',
   'milk',
   'there',
   '.',
   'John',
   'moved',
   'to',
   'the',
   'bedroom',
   '.'],
  ['Is', 'John', 'in', 'the', 'kitchen', '?'],
  'no'),
 (['Mary',
   'got',
   'the',
   'milk',
   'there',
```

Step 2: Data Exploration

In [8]:
```
len(test_data)
```

Out[8]:
```
1000
```

In [9]:
```
len(train_data)
```

Out[9]:
```
10000
```

Step 3: Setting up vocabulary of all words.

In [10]:
```
vocal = set()
all_data =train_data +test_data
```

In [11]:
```
for story,question,ans in all_data :
    vocal = vocal.union(set(story))
    vocal = vocal.union(set(question))
```

In [12]:
```
vocal.add("yes")
vocal.add("no")
```

In [13]:
```
vocal
```

Out[13]:
```
{'.',
 '?',
 '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 [14]:
```
len(vocal)
```

Out[14]:
```
37
```

In [15]:
```
vocal_len = len(vocal)+1
```

In [16]:
```
max_story_len = max([len(data[0]) for data in all_data])
max_story_len
```

Out[16]:
```
156
```

In [17]:
```
max_ques_len = max([len(data[1]) for data in all_data])
max_ques_len
```

Out[17]: 6

In [18]:
```
#pip install keras ==2.8.0
#pip install tensarflow ==2.8.0
from keras.preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer
```

In [19]:
```
tokenizer = Tokenizer(filters =[])
tokenizer.fit_on_texts(vocal)
```

In [20]:
```
tokenizer.word_index
```

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

In [21]:
```
train_story_text = []
train_ques_text = []
train_ans_text = []

for story, question, ans in train_data:
    train_story_text.append(story)
    train_ques_text.append(question)
    train_ans_text.append(ans)
```

In [22]:
```
train_story_seq = tokenizer.texts_to_sequences(train_story_text)
train_story_seq
```

Out[22]:
```
[[4, 32, 25, 31, 1, 23, 33, 11, 25, 31, 10, 23],
 [4,
  32,
  25,
  31,
  1,
  23,
  33,
  11,
  25,
  31,
  10,
  23,
  4,
  29,
  21,
  25,
  31,
  10,
```

In [23]:
```
len(train_story_text)
```

Out[23]:
```
10000
```

In [24]:
```
len(train_story_seq)
```

Out[24]:
```
10000
```

In [25]:
```
train_story_seq
```

Out[25]:
```
[[4, 32, 25, 31, 1, 23, 33, 11, 25, 31, 10, 23],
 [4,
  32,
  25,
  31,
  1,
  23,
  33,
  11,
  25,
  31,
  10,
  23,
  4,
  29,
  21,
  25,
  31,
  10,
```

In [26]:
```
train_story_text
```

Out[26]:
```
[['Mary',
  'moved',
  'to',
  'the',
  'bathroom',
  '.',
  'Sandra',
  'journeyed',
  'to',
  'the',
  'bedroom',
  '.'],
 ['Mary',
  'moved',
  'to',
  'the',
  'bathroom',
  '.',
  'Sandra',
```

Step 4: Vectorizing the data

In [27]:
```
def vectorize_stories(data, word_index=tokenizer.word_index,
                      max_story_len=max_story_len,max_ques_len=max_ques_len ):

    # X = STORIES
    X = []
    # Xq = QUERY/QUESTION
    Xq = []
    # Y = CORRECT ANSWER
    Y = []

    for story, query, answer in data:

        # Convert words to their corresponding indices in the word_index
        x = [word_index[word.lower()] for word in story]
        xq = [word_index[word.lower()] for word in query]

        # Create a one-hot encoded vector for the correct answer
        y = np.zeros(len(word_index) + 1)
        y[word_index[answer]] = 1

        # Append to respective lists
        X.append(x)
        Xq.append(xq)
        Y.append(y)

    # Pad sequences to a common length
    return (
        pad_sequences(X, maxlen=max_story_len),
        pad_sequences(Xq, maxlen=max_ques_len),
        np.array(Y)
    )
```

In [28]:
```
inputs_train ,queries_train,answers_train =  vectorize_stories(train_data)
```

In [29]:
```
inputs_test ,queries_test,answers_test =  vectorize_stories(test_data)
```

In [30]:
```
queries_test
```

Out[30]:
```
array([[34, 15, 26, 31,  3, 30],
       [34, 15, 26, 31,  3, 30],
       [34, 15, 26, 31, 19, 30],
       ...,
       [34,  4, 26, 31, 10, 30],
       [34, 33, 26, 31, 19, 30],
       [34,  4, 26, 31, 19, 30]])
```

In [31]:
```
answers_test
```

Out[31]:
```
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 [32]:
```
queries_train
```

Out[32]:
```
array([[34, 33, 26, 31, 20, 30],
       [34,  6, 26, 31,  1, 30],
       [34,  6, 26, 31, 12, 30],
       ...,
       [34, 33, 26, 31, 20, 30],
       [34,  4, 26, 31,  3, 30],
       [34,  4, 26, 31, 10, 30]])
```

In [33]:
```
answers_train
```

Out[33]:
```
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 [34]:
```
inputs_train
```

Out[34]:
```
array([[ 0,  0,  0, ..., 31, 10, 23],
       [ 0,  0,  0, ..., 31, 20, 23],
       [ 0,  0,  0, ..., 31,  1, 23],
       ...,
       [ 0,  0,  0, ..., 31, 10, 23],
       [ 0,  0,  0, ...,  9, 37, 23],
       [ 0,  0,  0, ..., 28, 37, 23]])
```

In [35]:
```
inputs_test
```

Out[35]:
```
array([[ 0,  0,  0, ..., 31, 10, 23],
       [ 0,  0,  0, ..., 31, 19, 23],
       [ 0,  0,  0, ..., 31, 19, 23],
       ...,
       [ 0,  0,  0, ..., 31, 28, 23],
       [ 0,  0,  0, ..., 31, 19, 23],
       [ 0,  0,  0, ..., 28, 37, 23]])
```

In [36]:
```
tokenizer.word_index["yes"]
```
Out[36]:
```
24
```

In [37]:
```
tokenizer.word_index["no"]
```
Out[37]:
```
7
```

Step 5: Creating the model

In [38]:
```
from keras.models import Sequential, Model
from keras.layers.embeddings import Embedding
from keras.layers import Input, Activation, Dense, Permute, Dropout
from keras.layers import Add, Dot, Concatenate
from keras.layers import LSTM

# Define input shapes
input_sequence = Input(shape=(max_story_len,))
question = Input(shape=(max_ques_len,))

# Define input_encoder_n
input_encoder_n = Sequential()
input_encoder_n.add(Embedding(input_dim=vocal_len, output_dim=64))
input_encoder_n.add(Dropout(0.3))

# Define input_encoder_c
input_encoder_c = Sequential()
input_encoder_c.add(Embedding(input_dim=vocal_len, output_dim=max_ques_len))
input_encoder_c.add(Dropout(0.3))

# Define question_encoder
question_encoder = Sequential()
question_encoder.add(Embedding(input_dim=vocal_len, output_dim=64, input_lengt
question_encoder.add(Dropout(0.3))

# Apply encoders to inputs
input_encoded_n = input_encoder_n(input_sequence)
input_encoded_c = input_encoder_c(input_sequence)
question_encoded = question_encoder(question)

# Calculate the match between input_encoded_n and question_encoded
match = Dot(axes=(2, 2))([input_encoded_n, question_encoded])
match = Activation('softmax')(match)

# Calculate response by adding match and input_encoded_c
response = Add()([match, input_encoded_c])
response = Permute((2, 1))(response)

# Concatenate response and question_encoded
answer = Concatenate(axis=-1)([response, question_encoded])

# Define further layers (e.g., LSTM, Dense) as needed
answer = LSTM(32)(answer)
answer = Dropout(0.5)(answer)
answer = Dense(vocal_len, activation='softmax')(answer)

# Create the model
chatbot_model = Model(inputs=[input_sequence, question], outputs=answer)

# Compile the model
chatbot_model.compile(optimizer='adam', loss='categorical_crossentropy', metri

# Print model summary
```

```
chatbot_model.summary()
```

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

 input_2 (InputLayer)           [(None, 6)]          0           []

 sequential (Sequential)        (None, None, 64)     2432        ['input_1[0]
[0]']

 sequential_2 (Sequential)      (None, 6, 64)        2432        ['input_2[0]
[0]']

 dot (Dot)                      (None, 156, 6)       0           ['sequential
[0][0]',
                                                                  'sequential
_2[0][0]']

 activation (Activation)        (None, 156, 6)       0           ['dot[0]
[0]']

 sequential_1 (Sequential)      (None, None, 6)      228         ['input_1[0]
[0]']

 add (Add)                      (None, 156, 6)       0           ['activation
[0][0]',
                                                                  'sequential
_1[0][0]']

 permute (Permute)              (None, 6, 156)       0           ['add[0]
[0]']

 concatenate (Concatenate)      (None, 6, 220)       0           ['permute[0]
[0]',
                                                                  'sequential
_2[0][0]']

 lstm (LSTM)                    (None, 32)           32384       ['concatenat
e[0][0]']

 dropout_3 (Dropout)            (None, 32)           0           ['lstm[0]
[0]']

 dense (Dense)                  (None, 38)           1254        ['dropout_3
[0][0]']

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

In [39]:
```
history = chatbot_model.fit(
    [inputs_train, queries_train],
    answers_train,
    batch_size=32,
    epochs=20,
    validation_data=([inputs_test, queries_test], answers_test)
)
```

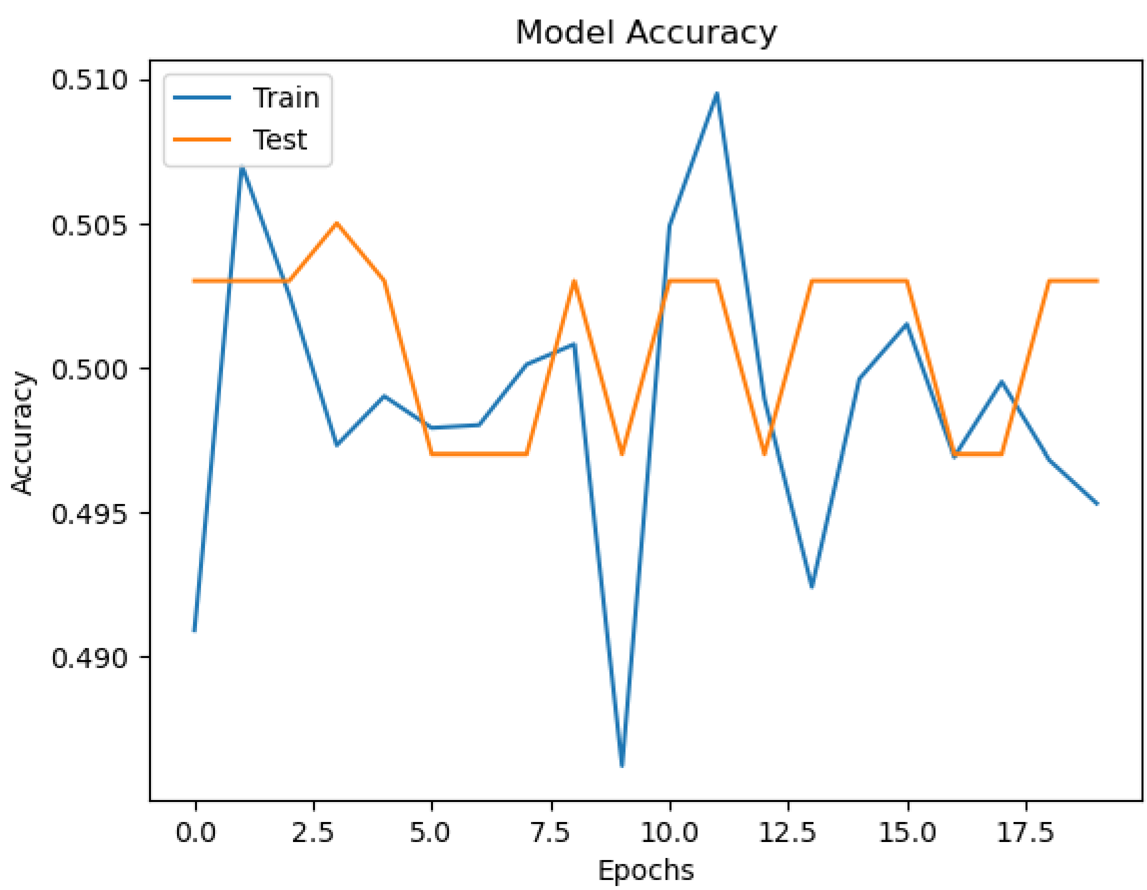
```
Epoch 1/20
313/313 [==============================] - 7s 15ms/step - loss: 0.9957 - accu
racy: 0.4909 - val_loss: 0.7013 - val_accuracy: 0.5030
Epoch 2/20
313/313 [==============================] - 4s 13ms/step - loss: 0.7205 - accu
racy: 0.5070 - val_loss: 0.6951 - val_accuracy: 0.5030
Epoch 3/20
313/313 [==============================] - 5s 16ms/step - loss: 0.7060 - accu
racy: 0.5025 - val_loss: 0.6958 - val_accuracy: 0.5030
Epoch 4/20
313/313 [==============================] - 5s 17ms/step - loss: 0.7002 - accu
racy: 0.4973 - val_loss: 0.6936 - val_accuracy: 0.5050
Epoch 5/20
313/313 [==============================] - 5s 15ms/step - loss: 0.6985 - accu
racy: 0.4990 - val_loss: 0.6934 - val_accuracy: 0.5030
Epoch 6/20
313/313 [==============================] - 4s 14ms/step - loss: 0.6971 - accu
racy: 0.4979 - val_loss: 0.6969 - val_accuracy: 0.4970
Epoch 7/20
313/313 [==============================] - 5s 16ms/step - loss: 0.6968 - accu
racy: 0.4980 - val_loss: 0.6973 - val_accuracy: 0.4970
Epoch 8/20
313/313 [==============================] - 4s 13ms/step - loss: 0.6961 - accu
racy: 0.5001 - val_loss: 0.6942 - val_accuracy: 0.4970
Epoch 9/20
313/313 [==============================] - 6s 20ms/step - loss: 0.6961 - accu
racy: 0.5008 - val_loss: 0.6936 - val_accuracy: 0.5030
Epoch 10/20
313/313 [==============================] - 4s 14ms/step - loss: 0.6965 - accu
racy: 0.4862 - val_loss: 0.6940 - val_accuracy: 0.4970
Epoch 11/20
313/313 [==============================] - 7s 24ms/step - loss: 0.6952 - accu
racy: 0.5049 - val_loss: 0.6932 - val_accuracy: 0.5030
Epoch 12/20
313/313 [==============================] - 6s 18ms/step - loss: 0.6949 - accu
racy: 0.5095 - val_loss: 0.6932 - val_accuracy: 0.5030
Epoch 13/20
313/313 [==============================] - 6s 18ms/step - loss: 0.6958 - accu
racy: 0.4989 - val_loss: 0.6933 - val_accuracy: 0.4970
Epoch 14/20
313/313 [==============================] - 4s 13ms/step - loss: 0.6955 - accu
racy: 0.4924 - val_loss: 0.6932 - val_accuracy: 0.5030
Epoch 15/20
313/313 [==============================] - 4s 12ms/step - loss: 0.6954 - accu
racy: 0.4996 - val_loss: 0.6969 - val_accuracy: 0.5030
Epoch 16/20
313/313 [==============================] - 5s 15ms/step - loss: 0.6953 - accu
racy: 0.5015 - val_loss: 0.6937 - val_accuracy: 0.5030
Epoch 17/20
313/313 [==============================] - 4s 13ms/step - loss: 0.6949 - accu
racy: 0.4969 - val_loss: 0.6954 - val_accuracy: 0.4970
Epoch 18/20
313/313 [==============================] - 4s 12ms/step - loss: 0.6956 - accu
racy: 0.4995 - val_loss: 0.6956 - val_accuracy: 0.4970
Epoch 19/20
313/313 [==============================] - 5s 16ms/step - loss: 0.6956 - accu
racy: 0.4968 - val_loss: 0.6940 - val_accuracy: 0.5030
```

```
Epoch 20/20
313/313 [==============================] - 4s 14ms/step - loss: 0.6955 - accu
racy: 0.4953 - val_loss: 0.6936 - val_accuracy: 0.5030
```

Step 6: Evaluating the Model

In [40]:
```
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")
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

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



In [41]:
```
chatbot_model.save("chatbot_model")
```

```
WARNING:absl:Found untraced functions such as lstm_cell_layer_call_fn, lstm_c
ell_layer_call_and_return_conditional_losses while saving (showing 2 of 2). T
hese functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: chatbot_model\assets

INFO:tensorflow:Assets written to: chatbot_model\assets
WARNING:absl:<keras.layers.recurrent.LSTMCell object at 0x0000012F22FE50D0> h
as the same name 'LSTMCell' as a built-in Keras object. Consider renaming <cl
ass 'keras.layers.recurrent.LSTMCell'> to avoid naming conflicts when loading
with `tf.keras.models.load_model`. If renaming is not possible, pass the obje
ct in the `custom_objects` parameter of the load function.
```

In [42]:
```
chatbot_model.load_weights("chatbot_model")
```

Out[42]:
```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x12f33f60d
60>
```

Step 7: Test Results

In [43]:
```
pred_results = chatbot_model.predict([inputs_test, queries_test])
```

In [44]:
```
test_data[0][0]
```

Out[44]:
```
['Mary',
 'got',
 'the',
 'milk',
 'there',
 '.',
 'John',
 'moved',
 'to',
 'the',
 'bedroom',
 '.']
```

In [45]:
```
story =' '.join(word for word in test_data[0][0])
print(story)
```

```
Mary got the milk there . John moved to the bedroom .
```

In [46]:
```
query = ' '.join(word for word in test_data[0][1])
print(query)
```

```
Is John in the kitchen ?
```

In [47]:
```
print("True Test Answer from Data is:",test_data[0][2])
```

```
True Test Answer from Data is: no
```

In [48]:
```
val_max = np.argmax(pred_results[0])
```

In [49]:
```
for key, val in tokenizer.word_index.items():
    if val == val_max:
        k = key
```

In [50]:
```
print("Predicted answer is: ", k)
print("Probability of certainty was: ", pred_results[0][val_max])
```

```
Predicted answer is:  no
Probability of certainty was:  0.51080245
```