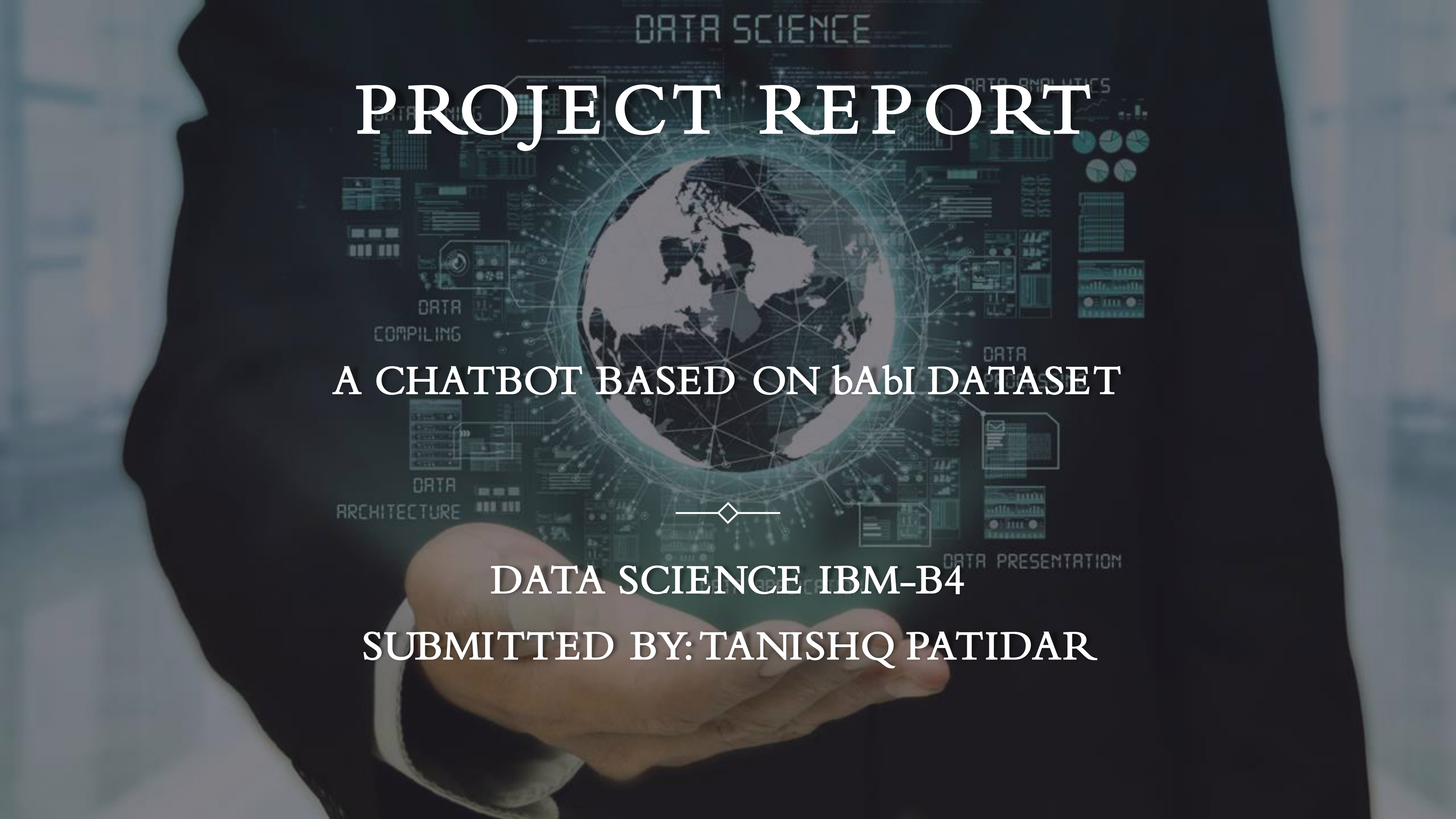


PROJECT REPORT

A CHATBOT BASED ON bAbI DATASET

DATA SCIENCE IBM-B4

SUBMITTED BY: TANISHQ PATIDAR

Table of Contents

ABSTRACT

An abstract is an outline/brief summary of this project report.

The aim is that each task tests a unique aspect of text and reasoning, and hence test different capabilities of learning models. More tasks are planned in the future to capture more aspects. The purpose is to investigate how long-term and short-term memory can be used in a chatbot to simulate a consistent persona for question answering and to enable long term question asking via user modeling. This is done by implementing and testing a chatbot with user specific and agent specific memories, where long-term memory data is mainly used through rule-based methods, such as templates, and short-term memory is used in a generative model. 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. We will be developing a simple chatbot that can answer questions based on a "story" given to it.

INTRODUCTION

Chatbots, also known as conversational agents, are designed with the help of AI (Artificial Intelligence) software. They simulate a conversation (or a chat) with users in a natural language via messaging applications, websites, mobile apps, or phone.

Chatbots represent a potential shift in how people interact with data and services online. While there is currently a surge of interest in chatbot design and development, we lack knowledge about why people use chatbots. Chatbots are computer programs designed to hold conversations with user using natural language. Some of them have human identities and personalities to make the conversation more natural. Nearly half of the online interactions between 2007 and 2015 involved a chatbot. They have been documented for use in variety of contexts, including education and commerce. There are two primary ways chatbots are offered to visitors:

- ❖ Web-based applications
- ❖ Standalone applications

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 services a chatbot can deliver are diverse. Important life-saving health messages, to check the weather forecast or to purchase a new pair of shoes, and anything else in between. Consumers spend lots of time using messaging applications (more than they spend on social media). Therefore, messaging applications are currently the most popular way companies deliver chatbot experiences to consumers.

METHODOLOGY

A chatbot is a computer program designed to simulate conversation with human users, especially over the internet.

When a question is presented to a chatbot, a series or complex algorithms process the received input, understand what the user is asking, and based on that, determines the answer suitable to the question.

The main technology that lies behind chatbots is NLP and Machine Learning.

Artificial Neural Network algorithms by design, try to process information the same way as our brain. Artificial neural networks are a collection of nodes called artificial neurons. The artificial neurons are interconnected and communicate with each other. These neurons are organized into multiple layers starting from input layer which receives external data followed by zero or more in-between hidden layers and an output layer which produces the result. There exist multiple connections between neurons of same or different layers and each connection is assigned a weight that represents its importance.

The output is calculated from the input using weighted connections which are calculated from repeated iterations while training the data. Based on the flow of data though the network, Artificial neural networks can be classified into Feed-forward and Feedback. In Feed-Forward networks the flow of information is unidirectional. Feed-back neural network are recurrent networks where feedback loops are allowed. In Feed-back networks, the signal can travel in both directions. Sequence to Sequence (Seq2Seq) is a type of recurrent neural network and is one of the most popular network model for designing machine translation and dialogue systems. Seq2Seq model consists of two recurrent neural networks, an encoder and a decoder. Since recurrent neural networks have the problem of vanishing gradient, much more powerful variants such as Long Short Term Memory (LSTM) or Gated Recurrent Units (GRU) are used. The encoder network processes the input sentence (user query) by breaking down the sentence into a hidden feature vector consisting of only the important words. The decoder takes as input the hidden vector generated by the encoder. Along with its own hidden states, current word and the hidden vector generated by encoder, the decoder tries to produce the next hidden vector and finally predicts the next word. Thereby, the Seq2Seq model is able to understand the context of the conversation by taking two inputs (one from the user and the other from the previous output of the model) at each point of time.

Natural Language Processing provides chatbots the ability to read, understand and derive meaning from human languages. Natural language processing is a collective name for a combination of steps to be followed to convert the customer's text or speech into a structured data that could be used to select the related response. Some of the steps include segmentation, tokenization, lemmatization, identifying stop words, dependency parsing, named entity recognition and coreference resolution.

Natural Language Processing provides chatbots the ability to read, understand and derive meaning from human languages. Natural language processing is a collective name for a combination of steps to be followed to convert the customer's text or speech into a structured data that could be used to select the related response. Some of the steps include segmentation, tokenization, lemmatization, identifying stop words, dependency parsing, named entity recognition and coreference resolution.

Training Set Size: For each task, there are 1000 questions for training, and 1000 for testing. However, we emphasize that the goal is to use as little data as possible to do well on the tasks (i.e. if you can use less than 1000 that's even better) — and without resorting to engineering task-specific tricks that will not generalize to other tasks, as they may not be of much use subsequently. Note that the aim during evaluation is to use the _same_ learner across all tasks to evaluate its skills and capabilities.

Each task tests a unique aspect of dialog. Tasks are designed to complement the set of 20 bAbI tasks for story understanding of the previous section. For each task, there are 1000 dialogs for training, 1000 for development and 1000 for testing. For tasks 1-5, we also include a second test set (with suffix -OOV.txt) that contains dialogs including entities not present in training and development sets.

The file format for each task is as follows:

ID user_utterance [tab] bot_utterance …

The IDs for a given dialog start at 1 and increase. When the IDs in a file reset back to 1 you can consider the following sentences as a new dialog. When the bot speaks two times in a row, we used the special token "" to fill in for the missing user utterance. See more details in the README included with the dataset. The goal of the tasks is to predict the bot utterances, that can be sentences or API calls (sentences starting with the special token "api_call").

CODE

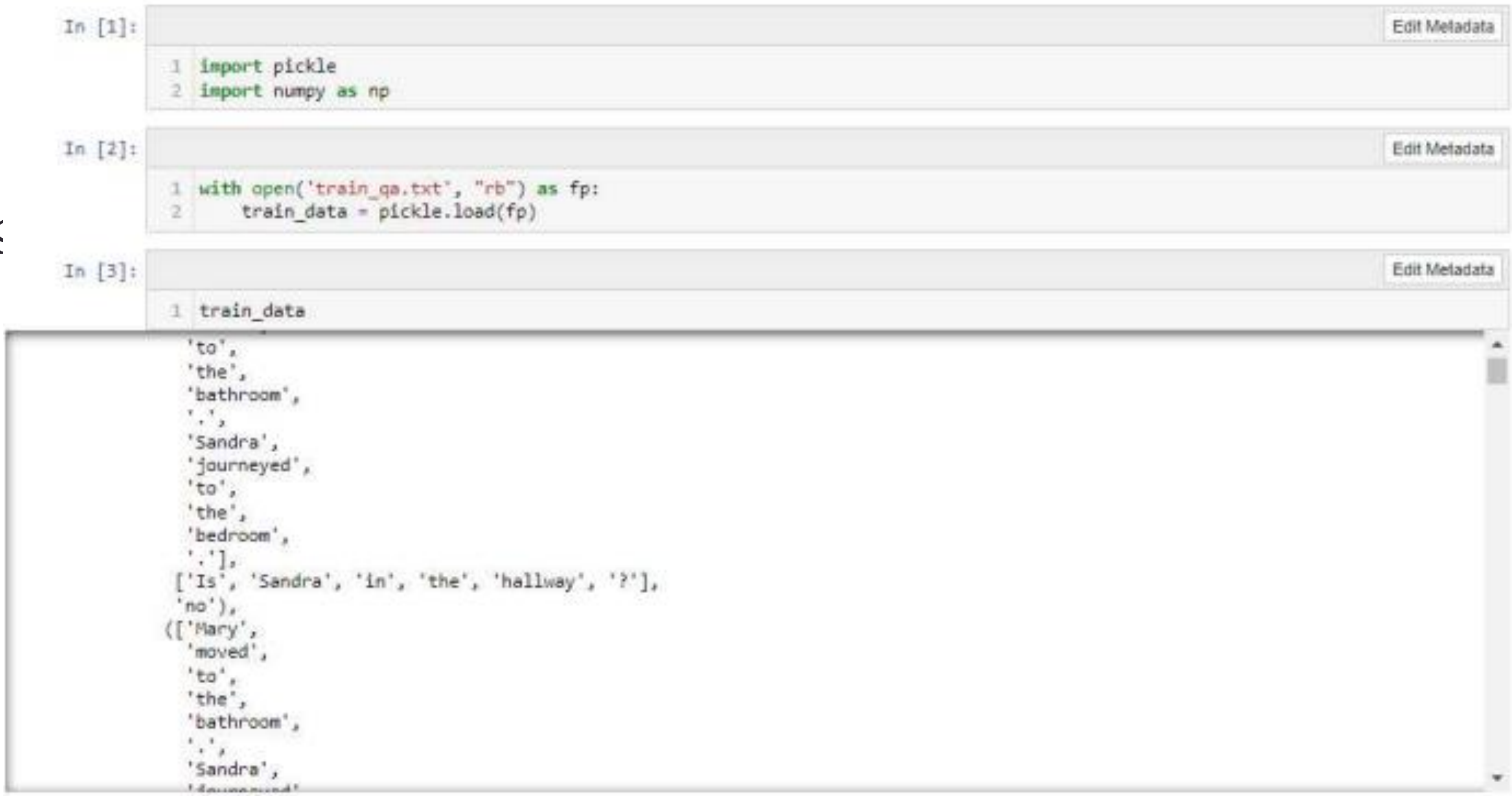
```
import pickle

import numpy as np

with open('train_qa.txt', "rb") as fp

train_data = pickle.load(fp)

train_data
```



len(train_data)

with open('test_qa.txt', "rb") as fp:

test_data = pickle.load(fp)

In [4]: Edit Metadata

```
len(train_data)
```

Out[4]: 10000

In [5]: Edit Metadata

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

test_data

len(test_data)

In [6]:

```
test_data
```

Out[6]:

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

In [7]:

```
len(test_data)
```

Out[7]:

```
1000
```

train_data[0][0]

train_data[0][1]

train_data[0][2]

In [8]:

```
train_data[0][0]
```

Out[8]:

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

In [9]:

```
train_data[0][1]
```

Out[9]:

```
['Is', 'Sandra', 'in', 'the', 'hallway', '?']
```

In [10]:

```
train_data[0][2]
```

Out[10]:

```
'no'
```

vocab = set()

all_data = test_data + train_data

all_data

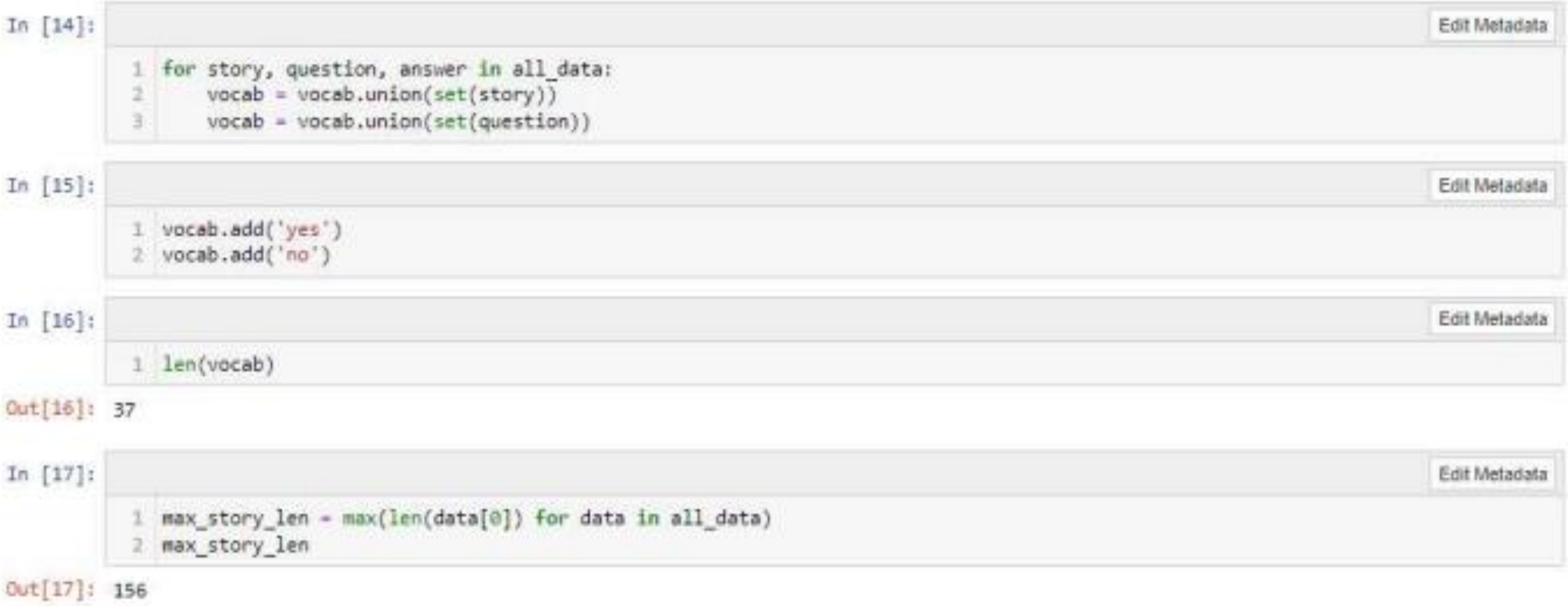
```
In [11]:                                                        Edit Metadata
vocab = set()

In [12]:                                                        Edit Metadata
all_data = test_data + train_data

In [13]:                                                        Edit Metadata
all_data

Out[13]: [(['Mary',
           'got',
           'the',
           'milk',
           'there',
           '.',
           'John',
           'moved',
           'to',
           'the',
           'bedroom',
           '.'],
          ['Is', 'John', 'in', 'the', 'kitchen', '?'],
          'no'),
         (['Mary',
           'got',
           'the',
           'milk',
           'there',
```

```
for story, question, answer in all_data:
    vocab = vocab.union(set(story))
    vocab = vocab.union(set(question))
vocab.add('yes')
vocab.add('no')
len(vocab)
max_story_len = max(len(data[0]) for data in all_data)
max_story_len
```



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

max_ques_len

from keras_preprocessing.sequence import pad_sequences

from keras_preprocessing.text import Tokenizer

tokenizer = Tokenizer(filters = [])

tokenizer.fit_on_texts(vocab)
```

```
In [18]:                                                                 Edit Metadata
max_ques_len = max(len(data[1]) for data in all_data)
max_ques_len
Out[18]: 6

In [20]:                                                                 Edit Metadata
from keras_preprocessing.sequence import pad_sequences
from keras_preprocessing.text import Tokenizer

In [21]:                                                                 Edit Metadata
tokenizer = Tokenizer(filters = [])

In [22]:                                                                 Edit Metadata
tokenizer.fit_on_texts(vocab)
```

tokenizer.word_index

In [23]:

Edit Metadata

```
tokenizer.word_index
```

Out[23]:

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

```
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)

train_story_seq = tokenizer.texts_to_sequences(train_story_text)
```

In [24]: Edit Metadata


```
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 [25]: Edit Metadata


```
train_story_seq = tokenizer.texts_to_sequences(train_story_text)
```

train_story_seq

In [26]: Edit Metadata

```
train_story_seq
```

Out[26]:

```
[[10, 16, 31, 8, 17, 15, 14, 6, 31, 8, 30, 15],
 [10,
  16,
  31,
  8,
  17,
  15,
  14,
  6,
  31,
  8,
  30,
  15,
  10,
  23,
  21,
  31,
  8,
  30,
```

```
def vectorize_stories(data, word_index=tokenizer.word_index
max_story_len=max_story_len, max_ques_len=max_ques_len):
 X = []
 Xq = []
Y = []
for story, ques, ans in data:
 x = [word_index[word.lower()] for word in story]
 xq = [word_index[word.lower()] for word in ques]
 y = np.zeros(len(word_index) + 1)
 y[word_index[ans]] = 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) )
inputs_train, queries_train, answers_train = vectorize_stories(train_data)
```

In [27]: Edit Metadata

```python
def vectorize_stories(data, word_index=tokenizer.word_index,
                      max_story_len=max_story_len, max_ques_len=max_ques_len):
    X = []
    Xq = []
    Y = []

    for story, ques, ans in data:
        x = [word_index[word.lower()] for word in story]
        xq = [word_index[word.lower()] for word in ques]
        y = np.zeros(len(word_index) + 1)
        y[word_index[ans]] = 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 [28]: Edit Metadata

```python
inputs_train, queries_train, answers_train = vectorize_stories(train_data)
```

inputs_train

queries_train

In [29]: Edit Metadata

```
inputs_train
```

Out[29]:
```
array([[ 0,  0,  0, ...,  8, 30, 15],
       [ 0,  0,  0, ...,  8, 35, 15],
       [ 0,  0,  0, ...,  8, 17, 15],
       ...,
       [ 0,  0,  0, ...,  8, 30, 15],
       [ 0,  0,  0, ..., 29,  1, 15],
       [ 0,  0,  0, ..., 33,  1, 15]])
```

In [30]: Edit Metadata

```
queries_train
```

Out[30]:
```
array([[22, 14, 20,  8, 35, 32],
       [22,  7, 20,  8, 17, 32],
       [22,  7, 20,  8,  4, 32],
       ...,
       [22, 14, 20,  8, 35, 32],
       [22, 10, 20,  8, 24, 32],
       [22, 10, 20,  8, 30, 32]])
```

answers_train

inputs_test, queries_test, answers_test = vectorize_stories(test_data)

inputs_test

```
In [31]: answers_train
Out[31]: array([[0., 0., 0., ..., 0., 0., 1.],
                [0., 0., 0., ..., 0., 0., 1.],
                [0., 0., 0., ..., 0., 0., 1.],
                ...,
                [0., 0., 0., ..., 0., 0., 1.],
                [0., 0., 0., ..., 0., 0., 1.],
                [0., 0., 0., ..., 0., 0., 0.]])

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

In [33]: inputs_test
Out[33]: array([[ 0,  0,  0, ...,  8, 30, 15],
                [ 0,  0,  0, ...,  8, 26, 15],
                [ 0,  0,  0, ...,  8, 26, 15],
                ...,
                [ 0,  0,  0, ...,  8, 33, 15],
                [ 0,  0,  0, ...,  8, 26, 15],
                [ 0,  0,  0, ..., 33,  1, 15]])
```

queries_test

answers_test

vocab_len = len(vocab) + 1

```
In [34]:
queries_test

Out[34]: array([[22, 34, 20,  8, 24, 32],
       [22, 34, 20,  8, 24, 32],
       [22, 34, 20,  8, 26, 32],
       ...,
       [22, 10, 20,  8, 30, 32],
       [22, 14, 20,  8, 26, 32],
       [22, 10, 20,  8, 26, 32]])

In [35]:
answers_test

Out[35]: array([[0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])

In [36]:
vocab_len = len(vocab) + 1
```

from tensorflow.python.keras.models import Sequential, Model

from tensorflow.python.keras.layers.embeddings import Embedding

from tensorflow.python.keras.layers import Input, Activation, Dense, Permute, Dropout, add, dot, concatenate, LSTM

input_sequence = Input((max_story_len,))

question = Input((max_ques_len,))

In [48]: Edit Metadata

```
from tensorflow.python.keras.models import Sequential, Model
from tensorflow.python.keras.layers.embeddings import Embedding
from tensorflow.python.keras.layers import Input, Activation, Dense, Permute, Dropout, add, dot, concatenate, LSTM
```

In [49]: Edit Metadata

```
input_sequence = Input((max_story_len,))
question = Input((max_ques_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_ques_len))
input_encoder_c.add(Dropout(0.3))
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))
input_encoded_m = input_encoder_m(input_sequence)
```

input_encoded_c = input_encoder_c(input_sequence)

question_encoded = question_encoder(question)

In [50]:

Edit Metadata

```
# 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 [51]:

Edit Metadata

```
# 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 [62]:

Edit Metadata

```
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 [98]:

Edit Metadata

```
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)

print('Match shape', match)
```

In [99]: Edit Metadata

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

In [100]: Edit Metadata

```
print('Match shape', match)
```

```
Match shape KerasTensor(type_spec=TensorSpec(shape=(None, 156, 6), dtype=tf.float32, name=None), name='activation_3/Softmax:0', description="created by layer 'activation_3'")
```

```
response = add([match, input_encoded_c])
response = Permute((2, 1))(response)
answer = concatenate([response, question_encoded])
answer = LSTM(32)(answer)
```

In [101]: Edit Metadata

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

In [102]: Edit Metadata

```
answer = concatenate([response, question_encoded])
```

In [103]: Edit Metadata

```
answer = LSTM(32)(answer)
```

```
answer = Dropout(0.5)(answer)

answer = Dense(vocab_len)(answer)

answer = Activation('softmax')(answer)

model = Model([input_sequence, question], answer)

model.compile(optimizer = 'rmsprop', loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

In [104]: Edit Metadata

```
answer = Dropout(0.5)(answer)
answer = Dense(vocab_len)(answer)
```

In [105]: Edit Metadata

```
answer = Activation('softmax')(answer)
```

In [108]: Edit Metadata

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

model.summary()

```
1  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_3 (Sequential)       (None, None, 64)     2432        input_3[0][0]
__________________________________________________________________________________________________
sequential_5 (Sequential)       (None, 6, 64)        2432        input_4[0][0]
__________________________________________________________________________________________________
dot_1 (Dot)                     (None, 156, 6)       0           sequential_3[0][0]
                                                                 sequential_5[0][0]
__________________________________________________________________________________________________
activation_2 (Activation)       (None, 156, 6)       0           dot_1[0][0]
__________________________________________________________________________________________________
sequential_4 (Sequential)       (None, None, 6)      228         input_3[0][0]
__________________________________________________________________________________________________
add_1 (Add)                     (None, 156, 6)       0           activation_2[0][0]
                                                                 sequential_4[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_5[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
__________________________________________________________________________________________________
```

```
history = model.fit([inputs_train, queries_train], answers_train,
        batch_size = 30, epochs = 22
        validation_data = ([inputs_test, queries_test], answers_test))
```

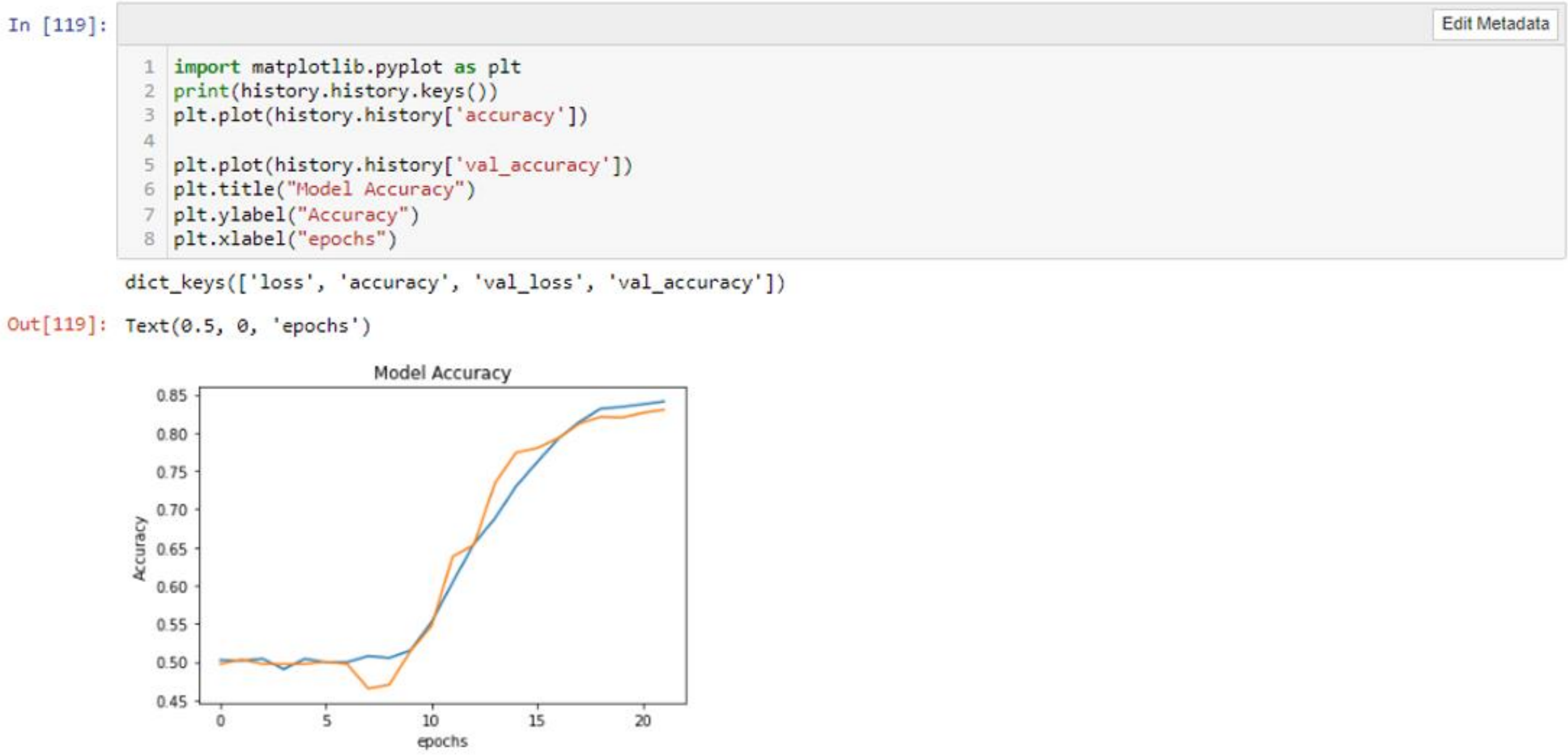
Edit Metadata

```
history = model.fit([inputs_train, queries_train], answers_train,
                    batch_size = 30, epochs = 22,
                    validation_data = ([inputs_test, queries_test], answers_test))
```

```
Epoch 1/22
334/334 [==============================] - 9s 16ms/step - loss: 0.9321 - accuracy: 0.5022 - val_loss: 0.7017 - val_accuracy: 0.
4970
Epoch 2/22
334/334 [==============================] - 5s 14ms/step - loss: 0.7076 - accuracy: 0.5014 - val_loss: 0.6941 - val_accuracy: 0.
5030
Epoch 3/22
334/334 [==============================] - 5s 14ms/step - loss: 0.6974 - accuracy: 0.5041 - val_loss: 0.6937 - val_accuracy: 0.
4970
Epoch 4/22
334/334 [==============================] - 5s 15ms/step - loss: 0.6961 - accuracy: 0.4902 - val_loss: 0.6933 - val_accuracy: 0.
4970
Epoch 5/22
334/334 [==============================] - 5s 15ms/step - loss: 0.6946 - accuracy: 0.5039 - val_loss: 0.6940 - val_accuracy: 0.
4970
Epoch 6/22
334/334 [==============================] - 5s 15ms/step - loss: 0.6953 - accuracy: 0.4994 - val_loss: 0.6933 - val_accuracy: 0.
5000
Epoch 7/22
334/334 [==============================] - 5s 14ms/step - loss: 0.6945 - accuracy: 0.4994 - val_loss: 0.6942 - val_accuracy: 0.
4970
Epoch 8/22
334/334 [==============================] - 5s 14ms/step - loss: 0.6942 - accuracy: 0.5076 - val_loss: 0.6940 - val_accuracy: 0.
4650
Epoch 9/22
334/334 [==============================] - 5s 14ms/step - loss: 0.6943 - accuracy: 0.5051 - val_loss: 0.6950 - val_accuracy: 0.
4700
Epoch 10/22
334/334 [==============================] - 5s 14ms/step - loss: 0.6926 - accuracy: 0.5148 - val_loss: 0.6945 - val_accuracy: 0.
5140
Epoch 11/22
334/334 [==============================] - 5s 15ms/step - loss: 0.6856 - accuracy: 0.5522 - val_loss: 0.6849 - val_accuracy: 0.
5470

Epoch 12/22
334/334 [==============================] - 5s 15ms/step - loss: 0.6610 - accuracy: 0.6042 - val_loss: 0.6421 - val_accuracy: 0.
6380
Epoch 13/22
334/334 [==============================] - 5s 14ms/step - loss: 0.6339 - accuracy: 0.6543 - val_loss: 0.6284 - val_accuracy: 0.
6530
Epoch 14/22
334/334 [==============================] - 5s 14ms/step - loss: 0.5876 - accuracy: 0.6879 - val_loss: 0.5453 - val_accuracy: 0.
7340
Epoch 15/22
334/334 [==============================] - 5s 16ms/step - loss: 0.5481 - accuracy: 0.7299 - val_loss: 0.4970 - val_accuracy: 0.
7740
Epoch 16/22
334/334 [==============================] - 5s 15ms/step - loss: 0.5124 - accuracy: 0.7616 - val_loss: 0.4506 - val_accuracy: 0.
7800
Epoch 17/22
334/334 [==============================] - 5s 15ms/step - loss: 0.4603 - accuracy: 0.7922 - val_loss: 0.4304 - val_accuracy: 0.
7930
Epoch 18/22
334/334 [==============================] - 5s 15ms/step - loss: 0.4259 - accuracy: 0.8144 - val_loss: 0.3893 - val_accuracy: 0.
8120
Epoch 19/22
334/334 [==============================] - 5s 15ms/step - loss: 0.3948 - accuracy: 0.8314 - val_loss: 0.3827 - val_accuracy: 0.
8210
Epoch 20/22
334/334 [==============================] - 5s 15ms/step - loss: 0.3869 - accuracy: 0.8339 - val_loss: 0.3704 - val_accuracy: 0.
8200
Epoch 21/22
334/334 [==============================] - 5s 14ms/step - loss: 0.3731 - accuracy: 0.8374 - val_loss: 0.3602 - val_accuracy: 0.
8260
Epoch 22/22
334/334 [==============================] - 5s 14ms/step - loss: 0.3610 - accuracy: 0.8411 - val_loss: 0.3906 - val_accuracy: 0.
8300
```

```
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")
```



```
#save
model.save("chatbot_model")
#Evaluation on the test set
model.load_weights("chatbot_model")
```

```
In [113]:
#Evaluation on the test set
model.load_weights("chatbot_model")

Out[113]: <tensorflow.python.checkpoint.checkpoint.CheckpointLoadStatus at 0x1cbb2f1bee0>
```

Edit Metadata

pred_results = model.predict(([inputs_test, queries_test]))

pred_results

In [122]:

Edit Metadata

```
1 pred_results = model.predict(([inputs_test, queries_test]))
```

In [123]:

Edit Metadata

```
1 pred_results
```

Out[123]:

```
array([[6.4261076e-11, 5.6199490e-11, 5.3541917e-11, ..., 5.1434908e-11,
        5.8469438e-11, 9.1903089e-11],
       [4.8528729e-12, 3.5212493e-12, 4.0435745e-12, ..., 3.0390636e-12,
        3.2645215e-12, 6.3436491e-12],
       [5.8537925e-11, 8.3605609e-11, 5.6989823e-11, ..., 9.8973850e-11,
        6.2519497e-11, 1.3062955e-10],
       ...,
       [3.1255161e-12, 2.4251042e-12, 2.7196474e-12, ..., 2.0015552e-12,
        2.2352697e-12, 4.0336935e-12],
       [1.1692533e-11, 1.6662461e-11, 1.1266797e-11, ..., 1.9154579e-11,
        1.2570435e-11, 2.7315220e-11],
       [5.0990076e-12, 6.1420422e-12, 4.5439655e-12, ..., 6.8755080e-12,
        5.1779418e-12, 9.5508879e-12]], dtype=float32)
```

```
test_data[0][0]

story = ' '.join(word for word in test_data[100][0])

story
```

```
In [116]:
test_data[0][0]

Out[116]: ['Mary',
 'got',
 'the',
 'milk',
 'there',
 '.',
 'John',
 'moved',
 'to',
 'the',
 'bedroom',
 '.']

In [117]:
story = ' '.join(word for word in test_data[100][0])

In [118]:
story

Out[118]: 'John took the apple there . John went to the bathroom .'
```

```
query = ' '.join(word for word in test_data[100][1])
query
test_data[100][2]
```

```
In [119]:  Edit Metadata
1 query = ' '.join(word for word in test_data[100][1])

In [120]:  Edit Metadata
1 query

Out[120]: 'Is John in the hallway ?'

In [121]:  Edit Metadata
1 test_data[100][2]

Out[121]: 'no'
```

```
val_max = np.argmax(pred_results[37])

for key, val in tokenizer.word_index.items():

    if val == val_max:

        k = key

print(" Predicted Answer is" , k)

print(" Probability of certainty" , pred_results[37][val_max])
```

In [130]: Edit Metadata

```
val_max = np.argmax(pred_results[37])

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

print("Predicted Answer is", k)
print("Probability of certainty", pred_results[37][val_max])
```

```
Predicted Answer is yes
Probability of certainty 0.94423413
```

CONCLUSION

Chatbots boost operational efficiency and bring cost savings to businesses while offering convenience and added services to internal employees and external customers. They allow companies to easily resolve many types of customer queries and issues while reducing the need for human interaction.

With chatbots, a business can scale, personalize, and be proactive all at the same time—which is an important differentiator. For example, when relying solely on human power, a business can serve a limited number of people at one time. To be cost-effective, human-powered businesses are forced to focus on standardized models and are limited in their proactive and personalized outreach capabilities.

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.