CREATING A CHATBOT

ABSTRACT

Chatbots, or conversational interfaces as they are also known, present a new way for individuals to interact with computer systems. Traditionally, to get a question answered by a software program involved using a search engine, or filling out a form. A chatbot allows a user to simply ask questions in the same manner that they would address a human. The most known chatbots currently are voice chatbots: Alexa and Siri. However, chatbots are currently being adopted at a high rate on computer chat platforms.

The technology at the core of the rise of the chatbot is natural language processing ("NLP"). Recent advances in machine learning have greatly improved the accuracy and effectiveness of natural language processing, making chatbots a viable option for many organizations. This improvement in NLP is firing a great deal of additional research which should lead to continued improvement in the effectiveness of chatbots in the years to come.

A simple chatbot can be created by loading an FAQ (frequently asked questions) into chatbot software. The functionality of the chatbot can be improved by integrating it into the organization's enterprise software, allowing more personal questions to be answered, like "What is my balance?", or "What is the status of my order?".

Most commercial chatbots are dependent on platforms created by the technology giants for their natural language processing. These include Amazon Lex, Microsoft Cognitive Services, Google Cloud Natural Language API, Facebook Deep Text, and IBM Watson. Platforms where chatbots are deployed include Facebook Messenger, Skype, and Slack, among many others.

OBJECTIVE

There may be several internal needs that meet several objectives and they must be well defined beforehand.

For example:

Improve user experience:

By adding a new self-service tool

Available 24/7

Make it easier to navigate and search for information on the desktop site

Better manage contact flows:

Reduce the number of calls with low added-value

Reduce the flow of incoming emails from the desktop site

Increase the customers' autonomy on the desktop site on:

Quotation requests

Retrieval of personal information/documents; Increase conversion rate

Improve customer knowledge

Each of these objectives has to be measurable both qualitatively and quantitatively. It is, therefore, necessary to think carefully about the metrics to monitor to ensure that once in production, the solution in place meets these KPIs and to see how it can be improved if necessary.

For example: reducing the volume of incoming emails by 20 to 30%, top 5 themes handled by the bot, number of quotes generated thanks to the chatbot.

INTRODUCTION

Chatbots are not a recent development. They are simulations that can understand human language, process it, and interact back with humans while performing specific tasks. For example, a chatbot can be employed as a helpdesk executive. Joseph created the first chatbot in 1966, named Eliza. It all started when Alan Turing published an article named “Computer Machinery and Intelligence” and raised an intriguing question, “Can machines think?” ever since, we have seen multiple chatbots surpassing their predecessors to be more naturally conversant and technologically advanced. These advancements have led us to an era where conversations with chatbots have become as normal and natural as with another human. Before looking into the AI chatbot, learn the foundations of artificial intelligence.

Today, almost all companies have chatbots to engage their users and serve customers by catering to their queries. We practically will have chatbots everywhere, but this doesn’t necessarily mean that all will be well-functioning. The challenge here is not to develop a chatbot but to develop a well-functioning one.

METHODOLOGY

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

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.

Chatbots have to rely on the ability of the algorithms to detect the complexity of both text and spoken words. Some chatbots perform very well to the point it becomes difficult to differentiate whether the user is a machine or a human.

However, handling complex conversations is a huge challenge; where there is a usage of various figures of speech, it may be difficult for machines to understand.

CODE

```
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:
            test_data = pickle.load(fp)

           'moved',
           'to',
           'the',
           'bathroom',
           '.',
           'Sandra',

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',
```

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

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

In [6]:
```
len(train_data)
```

Out[6]: 10000

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

In [7]:
```
len(test_data)
```

Out[7]: 1000

In [11]:
```
train_data[0][2]
```

Out[11]: 'no'


```
    train_data = pickle.load(fp)
```

In [6]:
```
len(train_data)
```

Out[6]: 10000

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

In [7]:
```
len(test_data)
```

Out[7]: 1000

In [11]:
```
train_data[0][2]
```

Out[11]: 'no'

In [12]:
```
vocab = set()
```

In [13]:
```
all_data = test_data + train_data
```

In [14]:
```
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',
          'hallway',
          'in',
          'journeyed',
          'kitchen',
          'left',
          'milk',
          'moved',
          'no',
          'office',
          'picked',
          'put',
          'the',
          'there',
          'to',
          'took',
          'travelled',
          'up',
          'went',
          'yes'}
```

```
Out[6]: 10000

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

In [7]: len(test_data)

Out[7]: 1000

In [11]: train_data[0][2]

Out[11]: 'no'

In [12]: vocab = set()

In [13]: all_data = test_data + train_data

In [14]: 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 [18]: len(vocab)

Out[18]: 37
```

```
 'travelled',
 'up',
 'went',
 'yes'}
```

```
In [19]: max_story_len = max([len(data[0]) for data in all_data])
         max_story_len
```

```
Out[19]: 156
```

```
In [20]: max_ques_len = max([len(data[1]) for data in all_data])
         max_story_len
```

```
Out[20]: 156
```

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

```
In [25]: tokenizer = Tokenizer(filters = [])
```

```
In [26]: tokenizer.fits_on_texts(vocab)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_25216/1648031707.py in <module>
----> 1 tokenizer.fits_on_texts(vocab)

AttributeError: 'Tokenizer' object has no attribute 'fits_on_texts'
```

```
 'picked',
 'put',
 'the',
 'there',
 'to',
 'took',
 'travelled',
 'up',
 'went',
 'yes'}
```

```
In [19]: max_story_len = max([len(data[0]) for data in all_data])
         max_story_len
```

```
Out[19]: 156
```

```
In [20]: max_ques_len = max([len(data[1]) for data in all_data])
         max_story_len
```

```
Out[20]: 156
```

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

```
In [25]: tokenizer = Tokenizer(filters = [])
```

```
In [27]: tokenizer.fit_on_texts(vocab)
```

```
In [28]: tokenizer.word_index
```

```
Out[28]: {'the': 1,
          'milk': 2,
          'put': 3,
          'moved': 4,
          '.': 5,
          'sandra': 6,
          'dropped': 7,
```

```
'15': 16,
'took': 17,
'football': 18,
'mary': 19,
'kitchen': 20,
'back': 21,
'bedroom': 22,
'garden': 23,
'in': 24,
'down': 25,
'went': 26,
'picked': 27,
'?': 28,
'travelled': 29,
'bathroom': 30,
'yes': 31,
'john': 32,
'there': 33,
'to': 34,
'apple': 35,
'hallway': 36,
'office': 37}
```

In [29]:
```
# train dataset
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 [ ]:
```
train_story_seq = tokenizer.texts
```

In [29]:
```
# train dataset
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 [30]:
```
train_story_seq = tokenizer.texts_to_sequences(train_story_text)
```

In [31]:
```
train_story_seq
```

```
 34,
 1,
 36,
 5],
[19,
 4,
 34,
 1,
 30,
 5,
 6,
 10,
 34,
 1,
 22,
 5,
 19,
 26,
 21,
 34,
```

In [31]:
```
train_story_seq
```

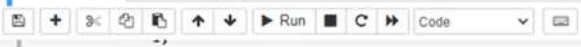
```
 34,
 1,
 36,
 5],
[19,
 4,
 34,
 1,
 30,
 5,
 6,
 10,
 34,
 1,
 22,
 5,
 19,
 26,
 21,
 34,
```

In [ ]:
```
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
```




```
 22,
 5,
 9,
 26,
 21,
 34,
 1,
 36,
 5],
[19,
 4,
 34,
 1,
```

In [ ]:
```
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, )
    )
```

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

```
In [36]: inputs_train
```

```
Out[36]: array([[ 0,  0,  0, ...,  1, 22,  5],
                [ 0,  0,  0, ...,  1, 36,  5],
                [ 0,  0,  0, ...,  1, 30,  5],
                ...,
                [ 0,  0,  0, ...,  1, 22,  5],
                [ 0,  0,  0, ...,  2, 33,  5],
                [ 0,  0,  0, ..., 35, 33,  5]])
```

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

```
In [38]: answers_train
```

```
Out[38]: 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., 1., 0., 0.,
                0., 0., 0., 0.])
```

```
In [39]: inputs_test, queries_test, answers_test = vectorize_stories(test_data)
```

```
In [40]: inputs_test
```

```
Out[40]: array([[ 0,  0,  0, ...,  1, 22,  5],
                [ 0,  0,  0, ...,  1, 23,  5],
                [ 0,  0,  0, ...,  1, 23,  5],
                ...,
                [ 0,  0,  0, ...,  1, 35,  5],
                [ 0,  0,  0, ...,  1, 23,  5],
                [ 0,  0,  0, ..., 35, 33,  5]])
```

```
        Y.append(y)

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

In [35]:
```
inputs_train, queries_train, answers_train = vectorize_stories(train_data)
```

In [38]:
```
answers_train
```

Out[38]:
```
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., 1., 0., 0.,
       0., 0., 0., 0.])
```

In [39]:
```
inputs_test, queries_test, answers_test = vectorize_stories(test_data)
```

In [40]:
```
inputs_test
```

Out[40]:
```
array([[ 0,  0,  0, ...,  1, 22,  5],
       [ 0,  0,  0, ...,  1, 23,  5],
       [ 0,  0,  0, ...,  1, 23,  5],
       ...,
       [ 0,  0,  0, ...,  1, 35,  5],
       [ 0,  0,  0, ...,  1, 23,  5],
       [ 0,  0,  0, ..., 35, 33,  5]])
```

In [43]:
```
from keras.models import Sequential, Model
from keras.layers.embeddings import Embedding
from keras.layers import Input, Activation, Dense, Permute, Dropout, add, dot, concatenate, LSTM
```


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

In [35]:
```
inputs_train, queries_train, answers_train = vectorize_stories(train_data)
```

In [38]:
```
answers_train
```

Out[38]:
```
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., 1., 0., 0.,
       0., 0., 0., 0.])
```

In [39]:
```
inputs_test, queries_test, answers_test = vectorize_stories(test_data)
```


In [45]:
```
vocab_len = len(vocab) + 1
```


In [43]:
```
from keras.models import Sequential, Model
from keras.layers.embeddings import Embedding
from keras.layers import Input, Activation, Dense, Permute, Dropout, add, dot, concatenate, LSTM
```


In [44]:
```
input_sequence = Input((max_story_len,))
question = Input((max_ques_len,))
```

In [46]:
```
# 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 [ ]:
```
# input encoder M
input_encoder_m = Sequential()
input_encoder_m.add(Embedding(input_dim = vocab_len, output_dim = ))
input_encoder_m.add(Dropout(0.3))
```

```
In [35]: inputs_train, queries_train, answers_train = vectorize_stories(train_data)

In [38]: answers_train

Out[38]: 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., 1., 0., 0.,
                0., 0., 0., 0.])

In [39]: inputs_test, queries_test, answers_test = vectorize_stories(test_data)

In [45]: vocab_len = len(vocab) + 1

In [43]: from keras.models import Sequential, Model
         from keras.layers.embeddings import Embedding
         from keras.layers import Input, Activation, Dense, Permute, Dropout, add, dot, concatenate, LSTM

In [44]: input_sequence = Input((max_story_len,))
         question = Input((max_ques_len,))

In [46]: # 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 [47]: # 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 [48]: ques_encoder = Sequential()
         ques_encoder.add(Embedding(input_dim = vocab_len, output_dim = 64, input_length = max_ques_len))
         ques_encoder.add(Dropout(0.3))

In [49]: input_encoded_m = input_encoder_m(input_sequence)
         input_encoded_c = input_encoder_c(question)
         ques_encoded = ques_encoder(question)

In [50]: match = dot([input_encoded_m, input_encoded_c], axes =(2,2))
```

```
In [62]: ques_encoder = Sequential()
         ques_encoder.add(Embedding(input_dim = vocab_len, output_dim = 64, input_length = max_ques_len))
         ques_encoder.add(Dropout(0.3))

In [63]: input_encoded_m = input_encoder_m(input_sequence)
         input_encoded_c = input_encoder_c(question)
         ques_encoded = ques_encoder(question)

In [50]: match = dot([input_encoded_m, input_encoded_c], axes =(2,2))
         match = Activation('softmax')(match)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_25216/3377029801.py in <module>
----> 1 match = dot([input_encoded_m, input_encoded_c], axes =(2,2))
      2 match = Activation('softmax')(match)

C:\ProgramData\Anaconda3\lib\site-packages\keras\layers\merge.py in dot(inputs, axes, normalize, **kwargs)
    986       A tensor, the dot product of the samples from the inputs.
    987   """
--> 988   return Dot(axes=axes, normalize=normalize, **kwargs)(inputs)

C:\ProgramData\Anaconda3\lib\site-packages\keras\utils\traceback_utils.py in error_handler(*args, **kwargs)
     65     except Exception as e:  # pylint: disable=broad-except
     66       filtered_tb = _process_traceback_frames(e.__traceback__)
---> 67       raise e.with_traceback(filtered_tb) from None
     68     finally:
     69       del filtered_tb

C:\ProgramData\Anaconda3\lib\site-packages\keras\layers\merge.py in build(self, input_shape)
    691       axes = self.axes
    692     if shape1[axes[0]] != shape2[axes[1]]:
--> 693       raise ValueError(
    694           'Incompatible input shapes: '
    695           f'axis values {shape1[axes[0]]} (at axis {axes[0]}) != '

ValueError: Incompatible input shapes: axis values 64 (at axis 2) != 6 (at axis 2). Full input shapes: (None, 156, 64), (None,
6, 6)
```

```
input_encoder_m.add(Dropout(0.3))
```

In [61]:
```
# 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]:
```
ques_encoder = Sequential()
ques_encoder.add(Embedding(input_dim = vocab_len, output_dim = 64, input_length = max_ques_len))
ques_encoder.add(Dropout(0.3))
```

In [63]:
```
input_encoded_m = input_encoder_m(input_sequence)
input_encoded_c = input_encoder_c(question)
ques_encoded = ques_encoder(question)
```

In [65]:
```
match = dot([input_encoded_m, input_encoded_c], axes =(2,2))
match = Activation('softmax')(match)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_25216/3377029801.py in <module>
----> 1 match = dot([input_encoded_m, input_encoded_c], axes =(2,2))
      2 match = Activation('softmax')(match)

C:\ProgramData\Anaconda3\lib\site-packages\keras\layers\merge.py in dot(inputs, axes, normalize, **kwargs)
    986       A tensor, the dot product of the samples from the inputs.
    987   """
--> 988   return Dot(axes=axes, normalize=normalize, **kwargs)(inputs)

C:\ProgramData\Anaconda3\lib\site-packages\keras\utils\traceback_utils.py in error_handler(*args, **kwargs)
     65     except Exception as e:  # pylint: disable=broad-except
     66       filtered_tb = _process_traceback_frames(e.__traceback__)
---> 67       raise e.with_traceback(filtered_tb) from None
     68     finally:
     69       del filtered_tb

C:\ProgramData\Anaconda3\lib\site-packages\keras\layers\merge.py in build(self, input_shape)
    691       axes = self.axes
    692     if shape1[axes[0]] != shape2[axes[1]]:
--> 693       raise ValueError(
```


In [65]:
```
match = dot([input_encoded_m, input_encoded_c], axes =(2,2))
match = Activation('softmax')(match)
```

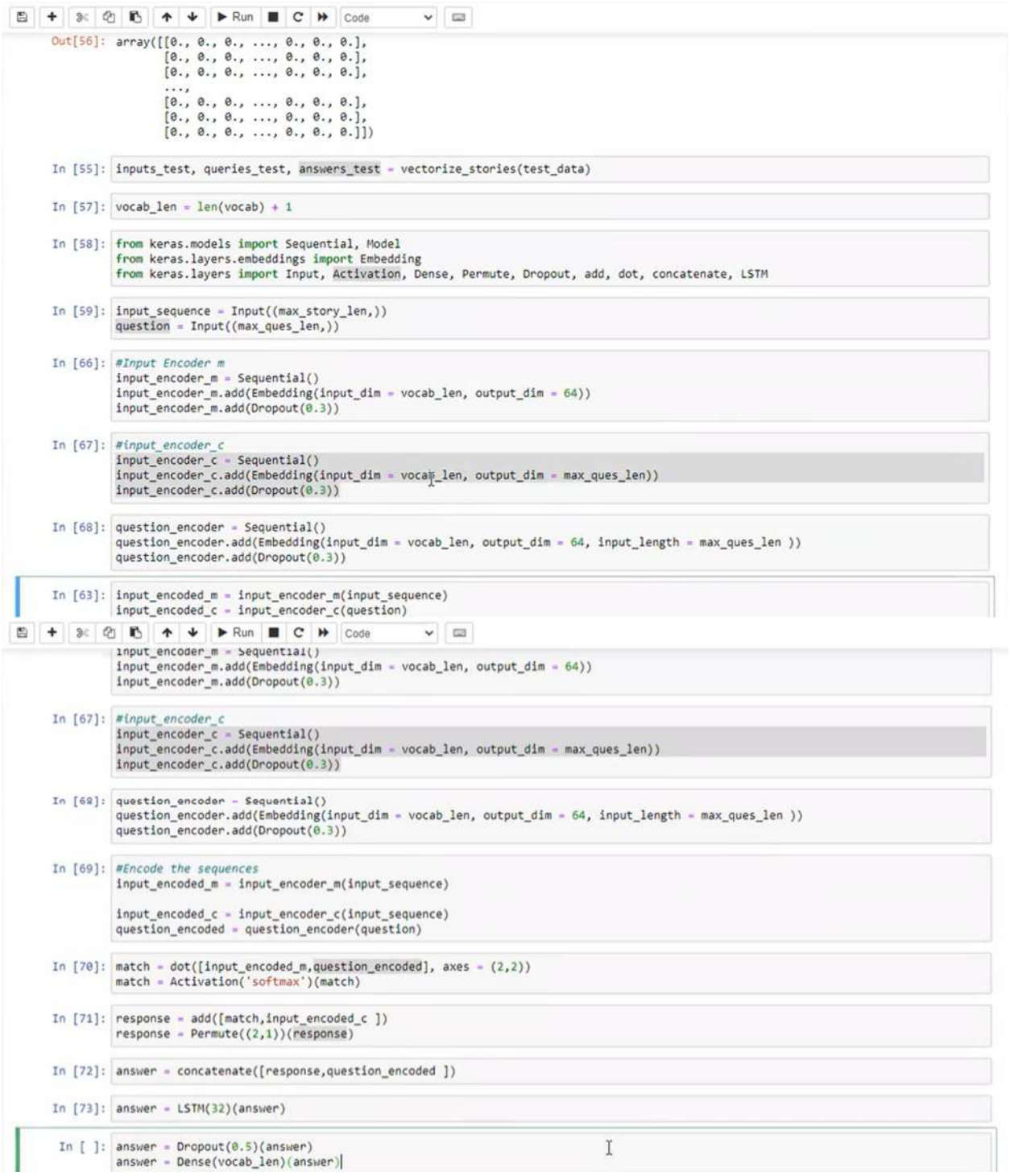
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_25216/3377029801.py in <module>
----> 1 match = dot([input_encoded_m, input_encoded_c], axes =(2,2))
      2 match = Activation('softmax')(match)

C:\ProgramData\Anaconda3\lib\site-packages\keras\layers\merge.py in dot(inputs, axes, normalize, **kwargs)
    986       A tensor, the dot product of the samples from the inputs.
    987   """
--> 988   return Dot(axes=axes, normalize=normalize, **kwargs)(inputs)

C:\ProgramData\Anaconda3\lib\site-packages\keras\utils\traceback_utils.py in error_handler(*args, **kwargs)
     65     except Exception as e:  # pylint: disable=broad-except
     66       filtered_tb = _process_traceback_frames(e.__traceback__)
---> 67       raise e.with_traceback(filtered_tb) from None
     68     finally:
     69       del filtered_tb

C:\ProgramData\Anaconda3\lib\site-packages\keras\layers\merge.py in build(self, input_shape)
    691       axes = self.axes
    692     if shape1[axes[0]] != shape2[axes[1]]:
--> 693       raise ValueError(
    694           'Incompatible input shapes: '
    695           f'axis values {shape1[axes[0]]} (at axis {axes[0]}) != '

ValueError: Incompatible input shapes: axis values 64 (at axis 2) != 6 (at axis 2). Full input shapes: (None, 156, 64), (None,
6, 6)
```



```
Out[56]: 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 [55]: inputs_test, queries_test, answers_test = vectorize_stories(test_data)

In [57]: vocab_len = len(vocab) + 1

In [58]: from keras.models import Sequential, Model
         from keras.layers.embeddings import Embedding
         from keras.layers import Input, Activation, Dense, Permute, Dropout, add, dot, concatenate, LSTM

In [59]: input_sequence = Input((max_story_len,))
         question = Input((max_ques_len,))

In [66]: #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 [67]: #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 [68]: 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 [63]: input_encoded_m = input_encoder_m(input_sequence)
         input_encoded_c = input_encoder_c(question)
```


```
In [67]: #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 [68]: 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 [69]: #Encode the sequences
         input_encoded_m = input_encoder_m(input_sequence)

         input_encoded_c = input_encoder_c(input_sequence)
         question_encoded = question_encoder(question)

In [70]: match = dot([input_encoded_m,question_encoded], axes = (2,2))
         match = Activation('softmax')(match)

In [71]: response = add([match,input_encoded_c ])
         response = Permute((2,1))(response)

In [72]: answer = concatenate([response,question_encoded ])

In [73]: answer = LSTM(32)(answer)

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

```
 activation_1 (Activation)      (None, 38)           0           ['dense[0][0]']

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

In [*]:
```
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 [==============================] - 6s 7ms/step - loss: 0.8866 - accuracy: 0.4925 - val_loss: 0.6978 - val_accuracy: 0.4
970
Epoch 2/22
334/334 [==============================] - 2s 6ms/step - loss: 0.7026 - accuracy: 0.5009 - val_loss: 0.6934 - val_accuracy: 0.4
970
Epoch 3/22
334/334 [==============================] - 2s 6ms/step - loss: 0.6957 - accuracy: 0.4957 - val_loss: 0.6969 - val_accuracy: 0.4
970
Epoch 4/22
334/334 [==============================] - 2s 6ms/step - loss: 0.6950 - accuracy: 0.4989 - val_loss: 0.6933 - val_accuracy: 0.5
030
Epoch 5/22
334/334 [==============================] - 2s 6ms/step - loss: 0.6953 - accuracy: 0.4946 - val_loss: 0.6934 - val_accuracy: 0.4
970
Epoch 6/22
334/334 [==============================] - 2s 6ms/step - loss: 0.6942 - accuracy: 0.5032 - val_loss: 0.6935 - val_accuracy: 0.4
970
Epoch 7/22
334/334 [==============================] - 2s 6ms/step - loss: 0.6948 - accuracy: 0.4994 - val_loss: 0.6933 - val_accuracy: 0.5
030
Epoch 8/22
231/334 [===================>.........] - ETA: 0s - loss: 0.6946 - accuracy: 0.5004
```

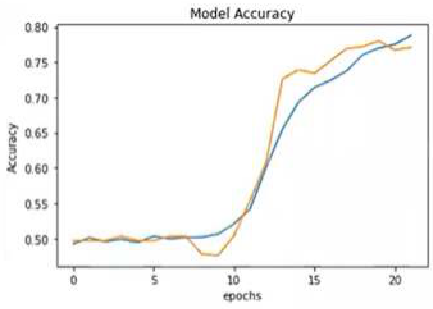
```
710
```

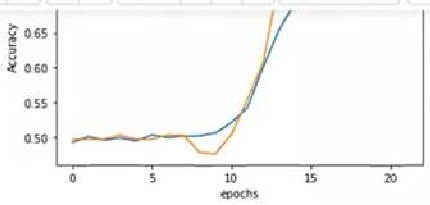
In [79]:
```
import matplotlib.pyplot as plt
print(history.history.keys())
plt.plot(history.history['accuracy'])

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[79]:
```
Text(0.5, 0, 'epochs')
```





In [80]:
```
#save
model.save("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: chatbot_model\assets

INFO:tensorflow:Assets written to: chatbot_model\assets
WARNING:absl:<keras.layers.recurrent.LSTMCell object at 0x000001390E1B10D0> has the same name 'LSTMCell' as a built-in Keras object. Consider renaming <class 'keras.layers.recurrent.LSTMCell'> to avoid naming conflicts when loading with `tf.keras.models.load_model`. If renaming is not possible, pass the object in the `custom_objects` parameter of the load function.
```

In [81]:
```
#Evaluation on the Test set
model.load_weights("chatbot_model")
```

Out[81]:
```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x1391d1c7790>
```

In [82]:
```
pred_results = model.predict(([inputs_test, queries_test]))
```

```
...,
[6.94561422e-12, 9.14323912e-12, 6.71424982e-12, ...,
 5.95381947e-12, 4.77562304e-12, 9.89238552e-12],
[2.06024347e-12, 1.51144028e-12, 1.78321680e-12, ...,
 1.26160627e-12, 1.82409816e-12, 1.47113485e-12],
[1.33736416e-11, 9.01981614e-12, 1.07816438e-11, ...,
 8.19514161e-12, 1.05681080e-11, 1.14234707e-11]], dtype=float32)
```

In [84]:
```
test_data[0][0]
```

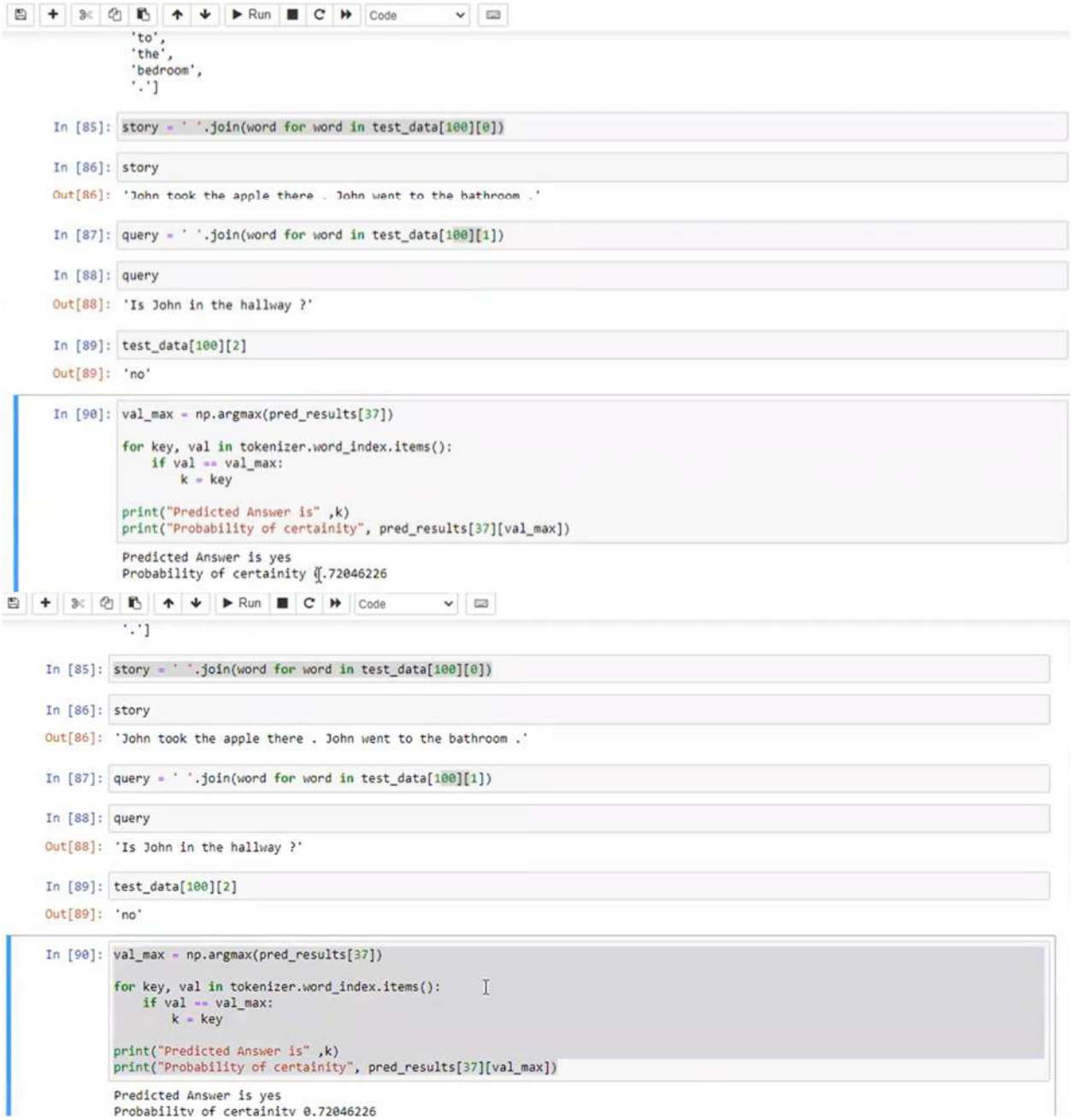
Out[84]:
```
['Mary',
 'got',
 'the',
 'milk',
 'there',
 '.',
 'John',
 'moved',
 'to',
 'the',
 'bedroom',
 '.']
```

In [85]:
```
story = ' '.join(word for word in test_data[100][0])
```

In [86]:
```
story
```

Out[86]:
```
'John took the apple there . John went to the bathroom .'
```

In [87]:
```
query = ' '.join(word for word in test_data[100][1])
```



```
'to',
'the',
'bedroom',
'.']

In [85]: story = ' '.join(word for word in test_data[100][0])

In [86]: story

Out[86]: 'John took the apple there . John went to the bathroom .'

In [87]: query = ' '.join(word for word in test_data[100][1])

In [88]: query

Out[88]: 'Is John in the hallway ?'

In [89]: test_data[100][2]

Out[89]: 'no'

In [90]: 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 certainity", pred_results[37][val_max])

         Predicted Answer is yes
         Probability of certainity 0.72046226
```


```
'.']

In [85]: story = ' '.join(word for word in test_data[100][0])

In [86]: story

Out[86]: 'John took the apple there . John went to the bathroom .'

In [87]: query = ' '.join(word for word in test_data[100][1])

In [88]: query

Out[88]: 'Is John in the hallway ?'

In [89]: test_data[100][2]

Out[89]: 'no'

In [90]: 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 certainity", pred_results[37][val_max])

         Predicted Answer is yes
         Probability of certainity 0.72046226
```


CONCLUSION

Smart solutions are important for the success of any business. From providing 24/7 customer service, improving current marketing activities, saving time spent on engaging with users to improving internal processes, chatbots can yield the much-needed competitive advantage. If you are looking to develop a

chatbot, the best thing to do is to approach a company that will understand your business needs to develop a chatbot that helps you achieve your business goals.

eInfochips helps organizations build highly-customized solutions on artificial intelligence, machine learning, deep learning, natural language processing as well as emerging technologies such as augmented reality and virtual reality. Know more about our AI and ML solutions.