PROJECT REPORT

A Chatbot based on

bAbI dataset

Submitted by:

Manan Jain

Data Science - IBM – B4

Table of Contents

Abstract

Chatbots, or conversational interfaces as they are also known, present a new way for individuals to interact with computer systems. Chatbots are intelligent agents with which users can hold conversations, usually via text or voice. 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 well-known chatbots currently are voice chatbots: Alexa and Siri. However, chatbots are currently being adopted at a high rate on computer chat platforms. The use of chatbots evolved rapidly in numerous fields in recent years, including Marketing, Supporting Systems, Education, Health Care, Cultural Heritage, and Entertainment. 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. 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 DeepText, and IBM Watson. Platforms where chatbots are deployed include Facebook Messenger, Skype, and Slack, among many others.

Objective

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 longterm 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.

There are two primary ways chatbots are offered to visitors:

- Web-based applications
- Standalone applications

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.

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

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.

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.

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.

Supervision Signal: Further while the MemNN results in the paper use full supervision (including of the supporting facts) results with weak supervision would also be ultimately preferable as this kind of data is easier to collect. Hence results of that form are very welcome. E.g. this paper does include weakly supervised results.

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 “<SILENCE>” 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”).

Some general steps required for creating the model:

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

Step 2: Data Exploration

Step 3: Setting up vocabulary of all words.

Step 4: Vectorizing the data

Step 5: Creating the model

Step 6: Evaluating the Model

Step 7: Test Results

Code

import pickle

import numpy as np

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

    train_data = pickle.load(fp)

train_data

```
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
```

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

len(train_data)

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

    test_data = pickle.load(fp)

```
In [4]:
len(train_data)
Out[4]: 10000
```

```
In [5]:
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]:
vocab = set()
```

```
In [12]:
all_data = test_data + train_data
```

```
In [13]:
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

```
In [14]:
for story, question, answer in all_data:
    vocab = vocab.union(set(story))
    vocab = vocab.union(set(question))
```

```
In [15]:
vocab.add('yes')
vocab.add('no')
```

```
In [16]:
len(vocab)

Out[16]: 37
```

```
In [17]:
max_story_len = max(len(data[0]) for data in all_data)
max_story_len

Out[17]: 156
```

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

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

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

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

tokenizer.word_index

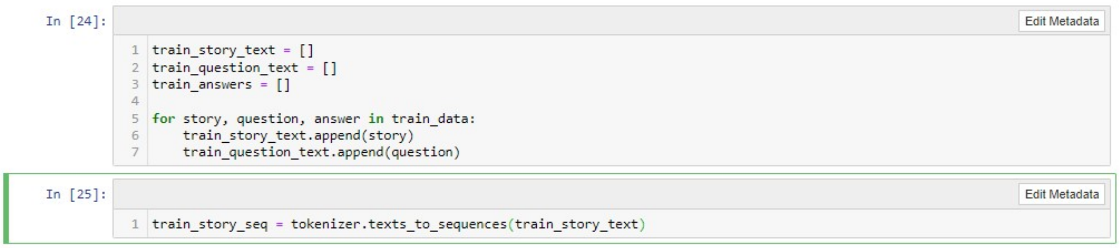
```
In [23]:
tokenizer.word_index
Out[23]: {'there': 1,
 'discarded': 2,
 'picked': 3,
 'office': 4,
 'yes': 5,
 'journeyed': 6,
 'daniel': 7,
 'the': 8,
 'dropped': 9,
 'mary': 10,
 'travelled': 11,
 'left': 12,
 'got': 13,
 'sandra': 14,
 '.': 15,
 'moved': 16,
 'bathroom': 17,
 'down': 18,
 'put': 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]:

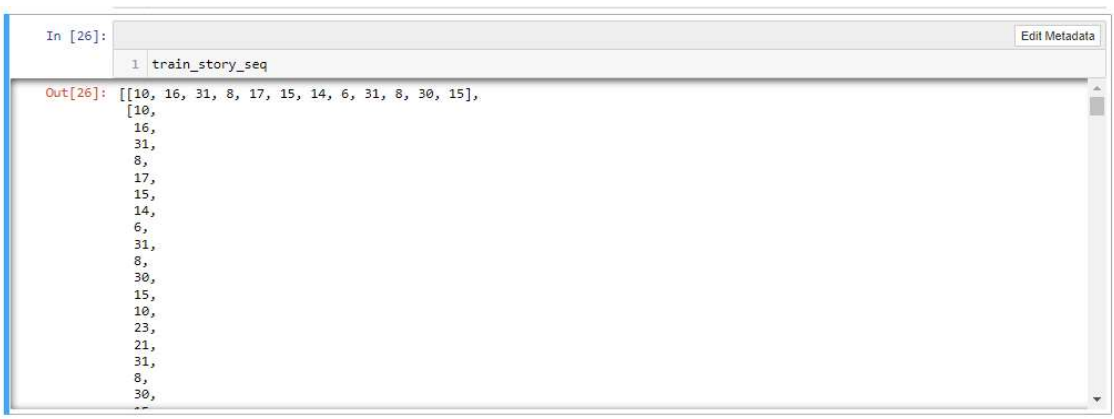
```
train_story_text = []
train_question_text = []
train_answers = []

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

In [25]:

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

```
train_story_seq
```



In [26]:

```
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

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

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
```

import sys

print(sys.path)

```
In [42]:                                                    Edit Metadata
import sys
print(sys.path)

['C:\\Users\\Aakanksha\\Desktop\\Manan\\Python IBM\\Chatbot', 'C:\\Users\\Aakanksha\\anaconda3\\python39.zip', 'C:\\Users\\Aaka
nksha\\anaconda3\\DLLs', 'C:\\Users\\Aakanksha\\anaconda3\\lib', 'C:\\Users\\Aakanksha\\anaconda3', '', 'C:\\Users\\Aakanksha
\\anaconda3\\lib\\site-packages', 'C:\\Users\\Aakanksha\\anaconda3\\lib\\site-packages\\win32', 'C:\\Users\\Aakanksha\\anaconda
3\\lib\\site-packages\\win32\\lib', 'C:\\Users\\Aakanksha\\anaconda3\\lib\\site-packages\\Pythonwin']
```

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]:

```
# 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]:

```
# 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]:

```
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]:

```
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]:

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

In [100]:

```
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]:

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

In [102]:

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

In [103]:

```
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]:

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

In [105]:

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

In [108]:

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

model.summary()

In [109]:

```
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_3 (Sequential)       (None, 6, 64)        2432        input_2[0][0]
__________________________________________________________________________________________________
dot_4 (Dot)                     (None, 156, 6)       0           sequential[2][0]
                                                                 sequential_3[1][0]
__________________________________________________________________________________________________
activation_3 (Activation)       (None, 156, 6)       0           dot_4[0][0]
__________________________________________________________________________________________________
sequential_1 (Sequential)       (None, None, 6)      228         input_1[0][0]
__________________________________________________________________________________________________
add_26 (Add)                    (None, 156, 6)       0           activation_3[0][0]
                                                                 sequential_1[2][0]
__________________________________________________________________________________________________
permute_1 (Permute)             (None, 6, 156)       0           add_26[0][0]
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 6, 220)       0           permute_1[0][0]
                                                                 sequential_3[1][0]
__________________________________________________________________________________________________
lstm_1 (LSTM)                   (None, 32)           32384       concatenate[0][0]
__________________________________________________________________________________________________
dropout_4 (Dropout)             (None, 32)           0           lstm_1[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 38)           1254        dropout_4[0][0]
__________________________________________________________________________________________________
activation_4 (Activation)       (None, 38)           0           dense[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))

In [110]:

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 [==============================] - 15s 18ms/step - loss: 0.8594 - accuracy: 0.5068 - val_loss: 0.6995 - val_accuracy:
0.5030
Epoch 2/22
334/334 [==============================] - 5s 15ms/step - loss: 0.6997 - accuracy: 0.5033 - val_loss: 0.6932 - val_accuracy: 0.
5030
Epoch 3/22
334/334 [==============================] - 5s 15ms/step - loss: 0.6953 - accuracy: 0.5037 - val_loss: 0.6944 - val_accuracy: 0.
4970
Epoch 4/22
334/334 [==============================] - 5s 15ms/step - loss: 0.6952 - accuracy: 0.4949 - val_loss: 0.6934 - val_accuracy: 0.
4970
Epoch 5/22
334/334 [==============================] - 5s 16ms/step - loss: 0.6936 - accuracy: 0.5041 - val_loss: 0.7056 - val_accuracy: 0.
5030
Epoch 6/22
334/334 [==============================] - 5s 15ms/step - loss: 0.6947 - accuracy: 0.4975 - val_loss: 0.6932 - val_accuracy: 0.
4830
Epoch 7/22
334/334 [==============================] - 5s 15ms/step - loss: 0.6947 - accuracy: 0.5022 - val_loss: 0.6933 - val_accuracy: 0.
5100
Epoch 8/22
334/334 [==============================] - 5s 15ms/step - loss: 0.6939 - accuracy: 0.5062 - val_loss: 0.6934 - val_accuracy: 0.
4780
Epoch 9/22
334/334 [==============================] - 5s 15ms/step - loss: 0.6930 - accuracy: 0.5118 - val_loss: 0.6953 - val_accuracy: 0.
5010
Epoch 10/22
334/334 [==============================] - 5s 15ms/step - loss: 0.6863 - accuracy: 0.5381 - val_loss: 0.6849 - val_accuracy: 0.
5580
Epoch 11/22
334/334 [==============================] - 5s 15ms/step - loss: 0.6690 - accuracy: 0.5839 - val_loss: 0.6542 - val_accuracy: 0.
6340
Epoch 12/22
334/334 [==============================] - 5s 15ms/step - loss: 0.6475 - accuracy: 0.6310 - val_loss: 0.6353 - val_accuracy: 0.
6420
Epoch 13/22
334/334 [==============================] - 5s 15ms/step - loss: 0.6248 - accuracy: 0.6595 - val_loss: 0.6131 - val_accuracy: 0.
6620
Epoch 14/22
334/334 [==============================] - 5s 16ms/step - loss: 0.6025 - accuracy: 0.6829 - val_loss: 0.5722 - val_accuracy: 0.
6910
Epoch 15/22
334/334 [==============================] - 5s 16ms/step - loss: 0.5711 - accuracy: 0.7111 - val_loss: 0.5263 - val_accuracy: 0.
7440
Epoch 16/22
334/334 [==============================] - 6s 17ms/step - loss: 0.5374 - accuracy: 0.7371 - val_loss: 0.4846 - val_accuracy: 0.
7810
Epoch 17/22
334/334 [==============================] - 6s 17ms/step - loss: 0.5045 - accuracy: 0.7688 - val_loss: 0.4522 - val_accuracy: 0.
8000
Epoch 18/22
334/334 [==============================] - 5s 16ms/step - loss: 0.4696 - accuracy: 0.7927 - val_loss: 0.4269 - val_accuracy: 0.
8120
Epoch 19/22
334/334 [==============================] - 5s 15ms/step - loss: 0.4488 - accuracy: 0.8041 - val_loss: 0.4278 - val_accuracy: 0.
8170
Epoch 20/22
334/334 [==============================] - 5s 15ms/step - loss: 0.4308 - accuracy: 0.8139 - val_loss: 0.4175 - val_accuracy: 0.
8180
Epoch 21/22
334/334 [==============================] - 5s 16ms/step - loss: 0.4232 - accuracy: 0.8235 - val_loss: 0.4062 - val_accuracy: 0.
8260
Epoch 22/22
334/334 [==============================] - 5s 16ms/step - loss: 0.4086 - accuracy: 0.8294 - val_loss: 0.4141 - val_accuracy: 0.
8320
```

```
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")
```

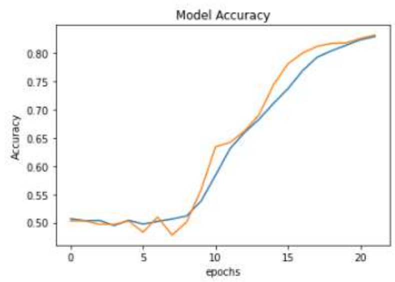
plt.xlabel("epochs")

```
In [111]:
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[111]: Text(0.5, 0, 'epochs')
```



#save

model.save("chatbot_model")

#Evaluation on the test set

model.load_weights("chatbot_model")

```
In [112]:
#save
model.save("chatbot_model")

INFO:tensorflow:Assets written to: chatbot_model\assets

C:\Users\Aakanksha\anaconda3\lib\site-packages\tensorflow\python\keras\utils\generic_utils.py:494: CustomMaskWarning: Custom ma
sk layers require a config and must override get_config. When loading, the custom mask layer must be passed to the custom_objec
ts argument.
  warnings.warn('Custom mask layers require a config and must override '

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

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

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

pred_results

In [114]:
```
pred_results = model.predict(([inputs_test, queries_test]))
```

In [115]:
```
pred_results
```

Out[115]:
```
array([[2.9916819e-07, 2.9792116e-07, 3.4124142e-07, ..., 3.0517151e-07,
        3.2096983e-07, 6.3408983e-01],
       [7.9504012e-09, 6.7845844e-09, 8.2744132e-09, ..., 6.6453305e-09,
        7.3805286e-09, 8.1857866e-01],
       [1.1507406e-12, 1.7158580e-12, 1.3201461e-12, ..., 1.7363556e-12,
        1.2034221e-12, 3.5890199e-02],
       ...,
       [1.4915043e-08, 1.2963903e-08, 1.5669368e-08, ..., 1.3458899e-08,
        1.4840922e-08, 6.2176710e-01],
       [1.8296439e-12, 2.6838503e-12, 2.1738670e-12, ..., 2.7688125e-12,
        1.9604444e-12, 4.1040763e-02],
       [4.4116742e-11, 4.8607434e-11, 4.5246140e-11, ..., 5.2155918e-11,
        4.1295654e-11, 1.5210931e-01]], 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]:
```
query = ' '.join(word for word in test_data[100][1])
```

In [120]:
```
query
```

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

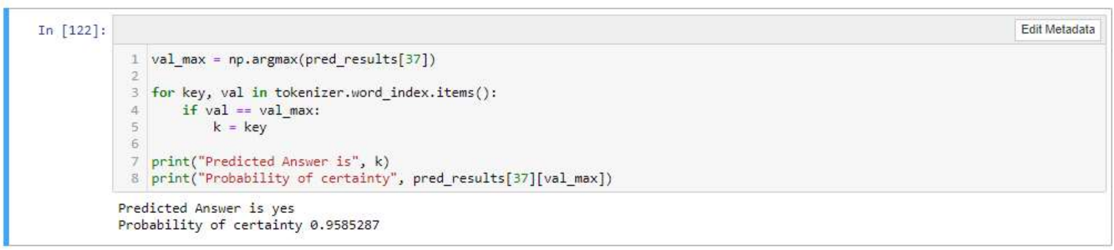
In [121]:
```
test_data[100][2]
```

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])
```



Conclusion

A chatbot allows even small and medium businesses to automate customer service live chat conversations. Chatbots in the late 20th century followed basic rules and had no reasoning capability. But with help of technologies such as big data, data mining and artificial intelligence, chat bots are getting smarter day by day. From this study, it becomes clear that natural language processing plays a predominant role in designing a chat bot. Many organizations have adopted to handle some of their customer's query using open domain model. It has become obvious that chat bots which provides natural open domain conversations ultimately have the potential to enhance the customer experience and thereby reducing organizations dependency on human resources.