ABSTRACT

In this project, we are working on bAbI project to predict the answer from the given data and evaluate the correctness of the predicted answer. First we will have the data that include a list in which each element of the row is of the type [story, question, answer]. We will form a model that will reduce loss and increase accuracy after every epoch it performs.

OBJECTIVE

The intent here is to perform training and testing for a Keras RNN Model and thereby making it compatible to predict the answer to random combinations of questions from the train and test data. As it is already known that Keras RNN is especially used to predict the n+1 token in a sequence when n tokens are already given. As illustrated in the project: The aim is that each task tests a unique aspect of text and reasoning, and hence tests different capabilities of learning models.

INTRODUCTION

RNN is a part of networks that is appropriate to learn sequential data for example text in Natural Language Processing. The motive here is to utilize sequential data to predict the next sequence which is not given in the data.

METHODOLOGY

The first thing we need to do is to go through the data that we are given. So we realize that the data is in the form of a list and it is divided into two segments that are train_qa.txt and test_qa.txt. When we go from an over the eye analysis we find that there are 1000 questions for training, and 1000 for testing. We need to keep in mind to use as little data as possible with the best accuracy. Now we have to load data and perform EDA to get familiar with data.

Vectorization

- Setting up vocab: Vocab basically includes the set of all words, characters without repetition. Here we are creating and empty set to begin then we have used vocab.union to make the required vocabulary. Also setting vocab_len = len(vocab) + 1 so as to include padding atleast once.
- Importing: Now we will import pad_sequences from keras.utils to apply padding and Tokenizer from keras.preprocessing.text to allocate tokens to each element of vocab. All

the elements with allotted integer is stored in tokenizer. We have the command tokenizer.text_to_sequences to convert the given text into sequences of the allotted integer.

- Padding: We will apply padding to get all the sequences in the same length.

Now we are done with Vectorization.

RNN

- Input: Assigning the input for the RNN model. We will have to give the shape for the input. For instance, shape=(32,) indicates that the expected input will be batches of 32-dimensional vectors. Now Introducing Sequential model which is appropriate for a plain stack of layers where each layer has exactly one input and one output. In our sequential model we will add Embedding with defined input and output also we will add dropout to avoid overfitting in our model. Similarly we will encode the questions.
- After applying the encoder, its output is received by the decoder. The scaled dot-product attention receives these queries, keys, and values as inputs and first computes the dot-product of the queries with the keys. This will produce the attention scores which are then given for activation into softmax to get the weights. At last we will permute and concatenate.
- LSTM: we will use LSTM to retain the information stored in the RNN model for a long period of time.It will make our data more accurate and efficient.
- Model: We will give input to our model given sequence as input_sequence and question. Then we will compile our model where we have used optimizer: rmsprop (Optimizers defines the weights we are using in RNN to reduce losses), loss = categorical_crossentropy (loss will reduce fitting in the model as our model is a kind of classification model so we will use categorical_crossentropy), metrics = accuracy (metrics is used to evaluate the performance of the model). Now we will model.summary to check all the details of the model we have made. Finally we will begin training our data using model.fit function. With batch_size = 32, epochs = 20. So our model is ready and we can now analyze losses and accuracy.
- Prediction:The most important task of our model is to predict after it is trained and evaluated. So we will use model.predict to predict results from the test data and evaluate its correctness.

FORMING SEQUENCES

We can now form our own sequence from the vocab we already have. Then splitting and vectorizing each term and finally putting into model prediction to predict the required results.

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

import warnings
warnings.filterwarnings('ignore')
```

```
with open("train_qa.txt","rb") as fp:
  train_data = pickle.load(fp)
```

```
train_data
```

```
        '.',
        'John',
        'took',
        'the',
        'milk',
        'there',
        '.',
        'Sandra',
        'moved',
        'to',
        'the',
        'office',
        '.',
        'John',
        'dropped',
        'the',
        'milk',
        '.',
        'Mary',
        'went',
        'to',
        'the',
        'office',
        '.',
        'Daniel',
        'journeyed',
        'to',
        'the',
        'bathroom',
        '.',
        'Mary',
        'journeyed',
        'to',
        'the',
        'hallway',
        '.',
        'Sandra',
        'went',
        'to',
        'the',
        'bathroom',
        '.',
        'Sandra',
        'travelled',
        'to',
        'the',
        'kitchen',
        '.',
        'Daniel',
        'picked',
        'up',
        'the',
        'apple',
        'there',
        '.'],
       ['Is', 'Sandra', 'in', 'the', 'hallway', '?'],
       'no'),
      ...]
```

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

```
#“Pickling” is the process whereby a Python object hierarchy is converted into a byte stream
```

```
type(test_data)
```

```
list
```

```
len(test_data)
```

```
1000
```

```
len(train_data)
```

```
10000
```

```
train_data[0]
```

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

```
' '.join(train_data[0][0])
```

```
'Mary moved to the bathroom . Sandra journeyed to the bedroom .'
```

```
' '.join(train_data[0][1])
```

```
'Is Sandra in the hallway ?'
```

```
' '.join(train_data[0][2])
```

```
'n o'
```

```
#set up vocab
vocab = set()
all_data = test_data + train_data
```

```
type(all_data)
```

```
list
```

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

```
vocab.add('yes')
vocab.add('no')
```

```
vocab
```

```
{'.',
 '?',
 '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'}
```

```
len(vocab)
```

```
    37
```

```
vocab_len = len(vocab) + 1
#leaving one space for pad sequence
```

```
for data in all_data:
 print(len(data[0]))
```

```
12
24
35
47
59
12
25
37
48
61
12
25
37
50
62
```

```
#easier way to perform the previous command
max_story_len = max([len(data[0]) for data in all_data])
max_story_len
```

```
156
```

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

```
6
```

```
#now we have to vectorize the data
#means to convert it into integers
from keras.utils import pad_sequences
from keras.preprocessing.text import Tokenizer
```

```
tokenizer = Tokenizer(filters = [])
```

```
tokenizer.fit_on_texts(vocab)
```

```
tokenizer.word_index
```

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

```
train_story_seq[0]
```

```
    [6, 35, 19, 30, 21, 5, 27, 25, 19, 30, 24, 5]
```

```
len(train_story_text)
```

```
    10000
```

```
len(train_story_seq)
```

```
    10000
```

```
#padding
#functionalize the vectors
def vectorizer_stories(data, word_index = tokenizer.word_index, max_story_len = max_story_len, max_ques_len = max_ques_len):

  X = [] #stories
  Xq = [] #questions
  Y = [] #correct answer

  for story, query, answer in data:
    x = [word_index[word.lower()] for word in story]
    xq = [word_index[word.lower()] for word in query]
    y = np.zeros(len(word_index) +1)
    y[word_index[answer]] =1


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

```
inputs_train, queries_train, answers_train = vectorizer_stories(train_data)
```

```
inputs_test, queries_test, answers_test = vectorizer_stories(test_data)
```

```
inputs_train
#The data type expected by the input, as a string
```

```
    array([[ 0,  0,  0, ..., 30, 24,  5],
           [ 0,  0,  0, ..., 30, 14,  5],
           [ 0,  0,  0, ..., 30, 21,  5],
           ...,
           [ 0,  0,  0, ..., 30, 24,  5],
           [ 0,  0,  0, ..., 33, 17,  5],
           [ 0,  0,  0, ..., 11, 17,  5]], dtype=int32)
```

```
inputs_train[23]
```

```
    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,  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,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
           22, 13, 30, 11, 17,  5, 36, 29,  3, 30, 37, 17,  5, 22,  4, 30, 11,
            5, 22, 35, 19, 30, 34,  5, 27, 15, 26, 19, 30, 21,  5, 27,  8, 30,
           11, 17,  5, 36, 10, 30, 37,  5, 36, 13, 30, 37, 17,  5, 27, 23, 32,
           30, 11,  5, 27,  8, 30, 11, 17,  5, 22, 28, 19, 30, 14,  5, 27,  4,
           30, 11,  5], dtype=int32)
```

```
queries_test
```

```
     array([[ 9, 36,  1, 30, 34,  2],
            [ 9, 36,  1, 30, 34,  2],
            [ 9, 36,  1, 30, 12,  2],
            ...,
            [ 9,  6,  1, 30, 24,  2],
            [ 9, 27,  1, 30, 12,  2],
            [ 9,  6,  1, 30, 12,  2]], dtype=int32)
```

```
answers_test
```

```
     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.]])
```

```
inputs_test[13]
```

```
     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,  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,  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, 36, 35, 19, 30, 14,  5, 27, 15, 19, 30, 24,  5,
            27, 28, 19, 30, 12,  5, 36, 13, 30, 37, 17,  5, 22, 15, 26, 19, 30,
            24,  5,  6, 35, 19, 30, 21,  5,  6, 15, 19, 30, 34,  5, 27, 15, 19,
            30, 14,  5], dtype=int32)
```

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

```
input_sequence = Input((max_story_len,))
#comma at last stores input as tuple only
question = Input((max_ques_len,))
# A shape tuple (integers), not including the batch size.
#For instance, shape=(32,) indicates that the expected input will be batches of 32-dimensional vectors.
```

```
#input encoder m
input_encoder_m = Sequential()
'''A Sequential model is appropriate for a plain stack of
 layers where each layer has exactly one input tensor and one output tensor.'''
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
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)


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


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

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

```
answer
```

```
<KerasTensor: shape=(None, 6, 220) dtype=float32 (created by layer 'concatenate_5')>
```

```
#LSTM= long shoet term memory
answer = LSTM(32)(answer)
```

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

```
answer = Dense(vocab_len)(answer)
```

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

```
#now we will build the final model
model = Model([input_sequence, question], answer)
model.compile(optimizer = 'rmsprop', loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

```
model.summary()
```

```
Model: "model_9"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_9 (InputLayer)           [(None, 156)]        0           []

 input_10 (InputLayer)          [(None, 6)]          0           []

 sequential_12 (Sequential)     (None, None, 64)     2432        ['input_9[0][0]']

 sequential_14 (Sequential)     (None, 6, 64)        2432        ['input_10[0][0]']

 dot_5 (Dot)                    (None, 156, 6)       0           ['sequential_12[0][0]',
                                                                  'sequential_14[0][0]']

 activation_10 (Activation)     (None, 156, 6)       0           ['dot_5[0][0]']

 sequential_13 (Sequential)     (None, None, 6)      228         ['input_9[0][0]']

 add_5 (Add)                    (None, 156, 6)       0           ['activation_10[0][0]',
                                                                  'sequential_13[0][0]']

 permute_5 (Permute)            (None, 6, 156)       0           ['add_5[0][0]']

 concatenate_5 (Concatenate)    (None, 6, 220)       0           ['permute_5[0][0]',
                                                                  'sequential_14[0][0]']

 lstm_5 (LSTM)                  (None, 32)           32384       ['concatenate_5[0][0]']

 dropout_20 (Dropout)           (None, 32)           0           ['lstm_5[0][0]']

 dense_5 (Dense)                (None, 38)           1254        ['dropout_20[0][0]']

 activation_11 (Activation)     (None, 38)           0           ['dense_5[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 = 32, epochs = 20,
                    validation_data =([inputs_test, queries_test], answers_test))
```

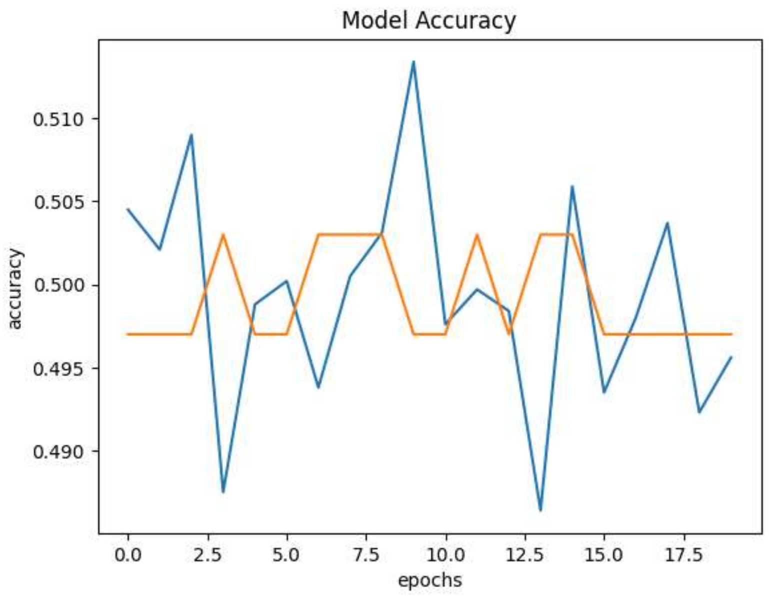
```
Epoch 1/20
313/313 [==============================] - 8s 26ms/step - loss: 0.6945 - accuracy: 0.5045 - val_loss: 0.6933 - val_accuracy: 0.4970
Epoch 2/20
313/313 [==============================] - 8s 25ms/step - loss: 0.6951 - accuracy: 0.5021 - val_loss: 0.6964 - val_accuracy: 0.4970
Epoch 3/20
313/313 [==============================] - 7s 21ms/step - loss: 0.6945 - accuracy: 0.5090 - val_loss: 0.6938 - val_accuracy: 0.4970
Epoch 4/20
313/313 [==============================] - 8s 25ms/step - loss: 0.6954 - accuracy: 0.4875 - val_loss: 0.6933 - val_accuracy: 0.5030
Epoch 5/20
```

```
313/313 [==============================] - 7s 21ms/step - loss: 0.6948 - accuracy: 0.4988 - val_loss: 0.6963 - val_accuracy: 0.4970
Epoch 6/20
313/313 [==============================] - 7s 22ms/step - loss: 0.6950 - accuracy: 0.5002 - val_loss: 0.6939 - val_accuracy: 0.4970
Epoch 7/20
313/313 [==============================] - 7s 22ms/step - loss: 0.6950 - accuracy: 0.4938 - val_loss: 0.6941 - val_accuracy: 0.5030
Epoch 8/20
313/313 [==============================] - 7s 22ms/step - loss: 0.6945 - accuracy: 0.5005 - val_loss: 0.6932 - val_accuracy: 0.5030
Epoch 9/20
313/313 [==============================] - 7s 23ms/step - loss: 0.6945 - accuracy: 0.5031 - val_loss: 0.6934 - val_accuracy: 0.5030
Epoch 10/20
313/313 [==============================] - 7s 21ms/step - loss: 0.6942 - accuracy: 0.5134 - val_loss: 0.6950 - val_accuracy: 0.4970
Epoch 11/20
313/313 [==============================] - 8s 24ms/step - loss: 0.6952 - accuracy: 0.4976 - val_loss: 0.6934 - val_accuracy: 0.4970
Epoch 12/20
313/313 [==============================] - 6s 20ms/step - loss: 0.6949 - accuracy: 0.4997 - val_loss: 0.6932 - val_accuracy: 0.5030
Epoch 13/20
313/313 [==============================] - 8s 24ms/step - loss: 0.6947 - accuracy: 0.4984 - val_loss: 0.6979 - val_accuracy: 0.4970
Epoch 14/20
313/313 [==============================] - 6s 20ms/step - loss: 0.6953 - accuracy: 0.4864 - val_loss: 0.6942 - val_accuracy: 0.5030
Epoch 15/20
313/313 [==============================] - 8s 25ms/step - loss: 0.6947 - accuracy: 0.5059 - val_loss: 0.6931 - val_accuracy: 0.5030
Epoch 16/20
313/313 [==============================] - 6s 20ms/step - loss: 0.6949 - accuracy: 0.4935 - val_loss: 0.6933 - val_accuracy: 0.4970
Epoch 17/20
313/313 [==============================] - 8s 25ms/step - loss: 0.6952 - accuracy: 0.4980 - val_loss: 0.6934 - val_accuracy: 0.4970
Epoch 18/20
313/313 [==============================] - 6s 21ms/step - loss: 0.6942 - accuracy: 0.5037 - val_loss: 0.6933 - val_accuracy: 0.4970
Epoch 19/20
313/313 [==============================] - 8s 24ms/step - loss: 0.6951 - accuracy: 0.4923 - val_loss: 0.6962 - val_accuracy: 0.4970
Epoch 20/20
313/313 [==============================] - 8s 24ms/step - loss: 0.6953 - accuracy: 0.4956 - val_loss: 0.6932 - val_accuracy: 0.4970
```

```
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'])
Text(0.5, 0, 'epochs')
```



```
#save model
model.save("chatbot model")
```

```
WARNING:absl:Found untraced functions such as _update_step_xla, lstm_cell_5_layer_call_fn, lstm_cell_5_layer_call_and_return_conditional
```

```
#evaluation on the test set
model.load_weights("chatbot model")
```

```
<tensorflow.python.checkpoint.checkpoint.CheckpointLoadStatus at 0x7f28c7f5a020>
```

```
test_data[0]
```

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

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

```
story
```

```
'Marygotthemilkthere.Johnmovedtothebedroom.'
```

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

```
query
```

```
'IsJohninthekitchen?'
```

```
test_data[0][2]
```

```
'no'
```

```
n = int(input("Select the story"))
story = ''.join(word for word in
                test_data[n][0])
query = ''.join(word for word in test_data[n][1])
print(test_data[n][2])
```

```
Select the story13
yes
```

```
n = int(input("Select the story"))
story = ''.join(word for word in
                test_data[n][0])
query = ''.join(word for word in test_data[n][1])
print(test_data[n][2])
```

```
Select the story23
no
```

Making Predictions

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

```
32/32 [==============================] - 1s 7ms/step
```

```
val_max=np.argmax(pred_results[13])
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[13][val_max])
```

```
predicted answer is yes
probability of certainity 0.50142884
```

```
val_max = np.argmax(pred_results[23])

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[23][val_max])
```

```
     Predicted Answer is yes
     Probability of Certainity 0.50152254
```

Forming Stories

```
vocab
```

```
     {'.',
      '?',
      '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'}
```

```
#now we will create the stor by ourself
story = "Mary dropped the football . Sandra discarded apple in the kitchen"
story.split()
```

```
     ['Mary',
      'dropped',
      'the',
      'football',
      '.',
      'Sandra',
      'discarded',
      'apple',
      'in',
      'the',
      'kitchen']
```

```
my_question = "Is apple in the kitchen ?"
my_question.split()
```

```
     ['Is', 'apple', 'in', 'the', 'kitchen', '?']
```

```
mydata = [(story.split(), my_question.split(), "yes")]
```

```
my_story, my_ques, my_ans = vectorizer_stories(mydata)
```

```
pred_results = model.predict(([my_story, my_ques]))

     1/1 [==============================] - 0s 27ms/step
```

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

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[0][val_max])

     Predicted Answer is yes
     Probability of Certainity 0.5014136
```

```
#now we will create the story by ourself
story = "Mary got the milk there . John moved to the bedroom . Mary discarded the milk . John is in the garden . Daniel moved to the bedroom
story.split()

     ['Mary',
      'got',
      'the',
      'milk',
      'there',
      '.',
      'John',
      'moved',
      'to',
      'the',
      'bedroom',
      '.',
      'Mary',
      'discarded',
      'the',
      'milk',
      '.',
      'John',
      'is',
      'in',
      'the',
      'garden',
      '.',
      'Daniel',
      'moved',
      'to',
      'the',
      'bedroom',
      '.',
      'Daniel',
      'went',
      'to',
      'the',
      'garden',
      '.']
```

```
my_question = "Is Daniel in the bedroom ?"
my_question.split()

     ['Is', 'Daniel', 'in', 'the', 'bedroom', '?']
```

```
mydata = [(story.split(), my_question.split(), 'no')]
```

```
my_story, my_ques, my_ans = vectorizer_stories(mydata)
```

```
pred_results = model.predict(([my_story, my_ques]))

     1/1 [==============================] - 0s 47ms/step
```

```
print(test_data[2])
```

```
(['Mary', 'got', 'the', 'milk', 'there', '.', 'John', 'moved', 'to', 'the', 'bedroom', '.', 'Mary', 'discarded', 'the', 'milk', '.', 'Jo
```

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

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[0][val_max])
```

```
Predicted Answer is yes
Probability of Certainity 0.50161445
```

✓ 0s completed at 11:35 PM

CONCLUSION

After 20 epochs we have got loss: 0.6953 and accuracy: 0.4956. We do not have good accuracy and loss is also not acceptable but working on this project gave a good idea about the working of Keras RNN and also about the deep learning concepts. It was great to work on this project.

Thank You