2.1 Single Layer

We start by describing our model in the single layer case, which implements a single memory hop operation. We then show it can be stacked to give multiple hops in memory.

Input memory representation: Suppose we are given an input set $x_1, .., x_i$ to be stored in memory. The entire set of $\{x_i\}$ are converted into memory vectors $\{m_i\}$ of dimension d computed by embedding each x_i in a continuous space, in the simplest case, using an embedding matrix A (of size $d \times V$). The query q is also embedded (again, in the simplest case via another embedding matrix B with the same dimensions as A) to obtain an internal state u. In the embedding space, we compute the match between u and each memory m_i by taking the inner product followed by a softmax:

$$p_i = \text{Softmax}(u^T m_i). \tag{1}$$

where $\text{Softmax}(z_i) = e^{z_i} / \sum_j e^{z_j}$. Defined in this way p is a probability vector over the inputs.

Output memory representation: Each x_i has a corresponding output vector c_i (given in the simplest case by another embedding matrix C). The response vector from the memory o is then a sum over the transformed inputs c_i, weighted by the probability vector from the input:

$$o = \sum_i p_i c_i. \tag{2}$$

Because the function from input to output is smooth, we can easily compute gradients and back-propagate through it. Other recently proposed forms of memory or attention take this approach, notably Bahdanau *et al.* [2] and Graves *et al.* [8], see also [9].

Generating the final prediction: In the single layer case, the sum of the output vector o and the input embedding u is then passed through a final weight matrix W (of size $V \times d$) and a softmax to produce the predicted label:

$$\hat{a} = \text{Softmax}(W(o + u)) \tag{3}$$

The overall model is shown in Fig. 1(a). During training, all three embedding matrices A, B and C, as well as W are jointly learned by minimizing a standard cross-entropy loss between $\hat{a}$ and the true label a. Training is performed using stochastic gradient descent (see Section 4.2 for more details).

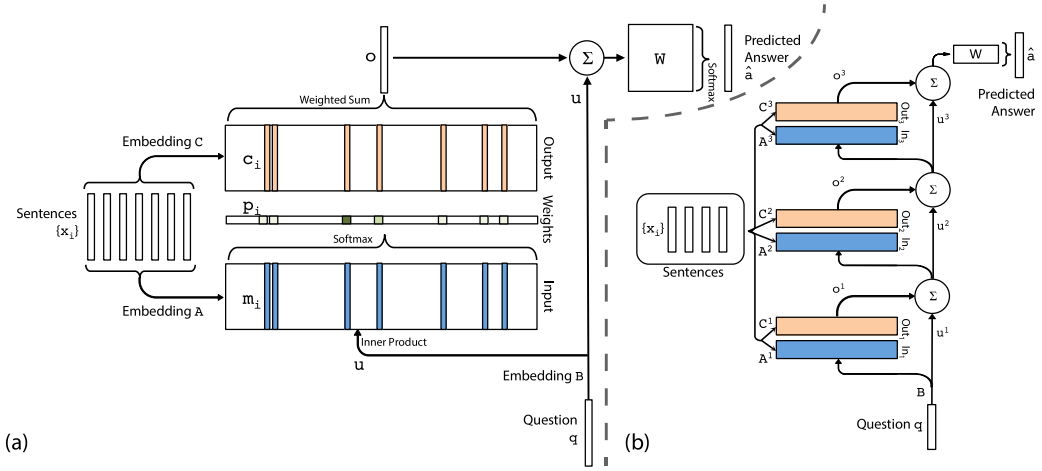


Figure 1: (a): A single layer version of our model. (b): A three layer version of our model. In practice, we can constrain several of the embedding matrices to be the same (see Section 2.2).

2.2 Multiple Layers

We now extend our model to handle K hop operations. The memory layers are stacked in the following way:

- The input to layers above the first is the sum of the output o^k and the input u^k from layer k (different ways to combine o^k and u^k are proposed later):

$$u^{k+1} = u^k + o^k. \tag{4}$$

By Nitish Adhikari

Email id :nitishbuzzpro@gmail.com (mailto:nitishbuzzpro@gmail.com), +91-9650740295

Linkedin : https://www.linkedin.com/in/nitish-adhikari-6b2350248 (https://www.linkedin.com/in/nitish-adhikari-6b2350248)

In [1]:

```
# import libraries
import pickle
import numpy as np
```

In [2]:

```
#Load the training data
with open('train_qa.txt','rb') as f:
    train_data = pickle.load(f)
```

In [3]:

```
#Load the test data
with open('test_qa.txt','rb') as f:
    test_data = pickle.load(f)
```

Exploring the Format of the Data

In [4]:

```
#check the data type
type(train_data)
```

Out[4]:

```
list
```

In [5]:

```
#check the data type
type(test_data)
```

Out[5]:

```
list
```

In [6]:

```
len(train_data)
```

Out[6]:

```
10000
```

In [7]:

```
len(test_data)
```

Out[7]:

```
1000
```

In [8]:

```
#check the train_data
train_data
```

Out[8]:

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

In [9]:

```
#Check the story
train_data[0][0]
```

Out[9]:

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

In [10]:

```
#join the story
' '.join(train_data[0][0])
```

Out[10]:

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

In [11]:

```
#join the question
' '.join(train_data[0][1])
```

Out[11]:

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

In [12]:

```
#Answer
train_data[0][2]
```

Out[12]:

```
'no'
```

Setting up Vocabulary of All Words

In [13]:

```
#create set of unique
vocab = set()

all_data = test_data + train_data

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

In [14]:

```
#Add 'yes' and 'no' to vocab
vocab.add('yes')
vocab.add('no')
```

In [15]:

```
#print vocab
print(vocab)
```

```
{'back', 'journeyed', 'dropped', '.', 'Mary', 'garden', '?', 'hallway', 'd
own', 'apple', 'Is', 'moved', 'the', 'to', 'went', 'put', 'left', 'up', 'D
aniel', 'grabbed', 'football', 'no', 'office', 'milk', 'travelled', 'yes',
'took', 'there', 'Sandra', 'got', 'bathroom', 'in', 'picked', 'kitchen',
'discarded', 'John', 'bedroom'}
```

In [16]:

```
vocab_len = len(vocab)+1
vocab_len
```

Out[16]:

```
38
```

In [17]:

```
#check for the long story
all_story_lens = [len(data[0]) for data in all_data]
all_story_lens
```

Out[17]:

```
[12,
 23,
 35,
 47,
 59,
 13,
 26,
 37,
 50,
 62,
 12,
 24,
 37,
 49,
 60,
 12,
 25,
 38,
```

In [18]:

```
#Maximum story length
max_story_len = max(all_story_lens)
max_story_len
```

Out[18]:

```
156
```

In [19]:

```
#Maximum qestion length
max_question_len = max(len(data[1]) for data in all_data)
max_question_len
```

Out[19]:

```
6
```

In [20]:

```
import tensorflow
from tensorflow .keras.preprocessing.sequence import pad_sequences
from tensorflow .keras.preprocessing.text import Tokenizer
```

```
C:\Users\DELL PC\AppData\Local\Programs\Python\Python310\lib\site-packages
\requests\__init__.py:109: RequestsDependencyWarning: urllib3 (1.26.9) or
chardet (5.1.0)/charset_normalizer (2.0.12) doesn't match a supported vers
ion!
  warnings.warn(
```

In [21]:

```
tokenizer = Tokenizer(filters=[])
tokenizer.fit_on_texts(vocab)
```

In [22]:

```
tokenizer.word_index
```

Out[22]:

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

In [23]:

```
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_answers.append(answer)
```

In [24]:

```
#convert to sequence
train_story_sequence = tokenizer.texts_to_sequences(train_story_text)
train_story_sequence
```

```
21,
28,
4,
29,
12,
14,
13,
31,
4,
36,
3,
13,
21,
4,
5,
15,
14,
13,
31,
4,
```

In [25]:

```
len(train_story_sequence)
```

Out[25]:

```
10000
```

In [26]:

```
def vectorize_stories(data,word_index=tokenizer.word_index,max_story_len=max_story_l

    import tensorflow
    from tensorflow .keras.preprocessing.sequence import pad_sequences
    from tensorflow .keras.preprocessing.text import Tokenizer

    X = [] #STORIES
    Xq = [] #SORIES question
    Y = [] #Correct answer(yes/no)

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

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

    return (pad_sequences(X,maxlen=max_story_len),pad_sequences(Xq,maxlen=max_questi
```

In [27]:

```
#Applying on train data
inputs_train, queries_train, answers_train = vectorize_stories(train_data)
```

In [28]:

```
#Applying on test data
inputs_test, queries_test, answers_test = vectorize_stories(test_data)
```

In [29]:

```
inputs_test
```

Out[29]:

```
array([[ 0,  0,  0, ..., 13, 37,  4],
       [ 0,  0,  0, ..., 13,  6,  4],
       [ 0,  0,  0, ..., 13,  6,  4],
       ...,
       [ 0,  0,  0, ..., 13, 10,  4],
       [ 0,  0,  0, ..., 13,  6,  4],
       [ 0,  0,  0, ..., 10, 28,  4]])
```

In [30]:

```
inputs_test.shape
```

Out[30]:

```
(1000, 156)
```

In [31]:

```
queries_test
```

Out[31]:

```
array([[11, 36, 32, 13, 34,  7],
       [11, 36, 32, 13, 34,  7],
       [11, 36, 32, 13,  6,  7],
       ...,
       [11,  5, 32, 13, 37,  7],
       [11, 29, 32, 13,  6,  7],
       [11,  5, 32, 13,  6,  7]])
```

In [32]:

```
queries_test.shape
```

Out[32]:

```
(1000, 6)
```

In [33]:

```
answers_test
```

Out[33]:

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

In [34]:

```
answers_test.shape
```

Out[34]:

```
(1000, 38)
```

In [35]:

```
sum(answers_test)
```

Out[35]:

```
array([  0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
         0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
       503.,   0.,   0.,   0., 497.,   0.,   0.,   0.,   0.,   0.,   0.,
         0.,   0.,   0.,   0.,   0.])
```

In [36]:

```
#Import neural network libraries
import tensorflow as tf
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Embedding, Input, Activation, Dense, Permute, Dr
```

In [37]:

```
#PLACEHOLDER shape =(max_story_len,batch_size)
input_sequence = Input((max_story_len,))
question = Input((max_question_len,))
```

In [38]:

```
len(vocab)
```

Out[38]:

```
37
```

In [39]:

```
#vocab_len
vocab_size = len(vocab) + 1
```

In [40]:

```
# Input Encoder M
input_encoder_m = Sequential()
input_encoder_m.add(Embedding(input_dim=vocab_size,output_dim=64))
input_encoder_m.add(Dropout(0.3))
```

In [41]:

```
input_encoder_m
```

Out[41]:

```
<keras.engine.sequential.Sequential at 0x157ac064f70>
```

In [42]:

```
# Input Encoder C
input_encoder_c = Sequential()
input_encoder_c.add(Embedding(input_dim=vocab_size,output_dim=max_question_len))
input_encoder_c.add(Dropout(0.3))
```

In [43]:

```
input_encoder_c
```

Out[43]:

```
<keras.engine.sequential.Sequential at 0x157c39f3e20>
```

In [44]:

```
# Question Encoder
question_encoder = Sequential()
question_encoder.add(Embedding(input_dim=vocab_size,output_dim=64,input_length=max_q
question_encoder.add(Dropout(0.3))
```

In [45]:

```
question_encoder
```

Out[45]:

```
<keras.engine.sequential.Sequential at 0x157c42b0f40>
```

In [46]:

```
#Encoded
input_encoded_m = input_encoder_m(input_sequence)
input_encoded_c = input_encoder_c(input_sequence)
question_encoded = question_encoder(question)
```

In [47]:

```
input_encoded_m
```

Out[47]:

```
<KerasTensor: shape=(None, 156, 64) dtype=float32 (created by layer 'seque
ntial')>
```

In [48]:

```
input_encoded_c
```

Out[48]:

```
<KerasTensor: shape=(None, 156, 6) dtype=float32 (created by layer 'sequen
tial_1')>
```

In [49]:

```
question_encoded
```

Out[49]:

```
<KerasTensor: shape=(None, 6, 64) dtype=float32 (created by layer 'sequent
ial_2')>
```

In [50]:

```
#Dot product to compute match between first vector sequence and query
match = dot([input_encoded_m,question_encoded],axes=(2,2))
match = Activation('softmax')(match)
match
```

Out[50]:

```
<KerasTensor: shape=(None, 156, 6) dtype=float32 (created by layer 'activa
tion')>
```

In [51]:

```
#Add the tensors
response = add([match, input_encoded_c])
response = Permute((2,1))(response)
response
```

Out[51]:

```
<KerasTensor: shape=(None, 6, 156) dtype=float32 (created by layer 'permut
e')>
```

In [52]:

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

Out[52]:

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

In [53]:

```
# reduce the answer tensor by LSTM Layer
answer = LSTM(32)(answer)
answer
```

Out[53]:

```
<KerasTensor: shape=(None, 32) dtype=float32 (created by layer 'lstm')>
```

In [54]:

```
# Add dropout layer for regularization
answer = Dropout(0.5)(answer)
answer
```

Out[54]:

```
<KerasTensor: shape=(None, 32) dtype=float32 (created by layer 'dropout_
3')>
```

In [55]:

```
#Add dense layer
answer = Dense(vocab_size)(answer)
answer
```

Out[55]:

```
<KerasTensor: shape=(None, 38) dtype=float32 (created by layer 'dense')>
```

In [56]:

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

Out[56]:

```
<KerasTensor: shape=(None, 38) dtype=float32 (created by layer 'activation
_1')>
```

In [57]:

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

In [58]:

```
#Summary of the Model
model.summary()
```

```
Model: "model"
__________________________________________________________________________________
_____________________________
 Layer (type)                   Output Shape         Param #     Connect
ed to
==================================================================================
============================
 input_1 (InputLayer)           [(None, 156)]        0           []

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

 sequential (Sequential)        (None, None, 64)     2432        ['input
_1[0][0]']

 sequential_2 (Sequential)      (None, 6, 64)        2432        ['input
_2[0][0]']

 dot (Dot)                      (None, 156, 6)       0           ['seque
ntial[0][0]',
                                                                  .
```

In [59]:

```
#Train the model
history = model.fit([inputs_train,queries_train],answers_train, batch_size=32,epochs
```

```
313/313 [==============================] - 6s 19ms/step - loss: 0.3400 -
accuracy: 0.8534 - val_loss: 0.3702 - val_accuracy: 0.8330
Epoch 95/100
313/313 [==============================] - 6s 19ms/step - loss: 0.3328 -
accuracy: 0.8585 - val_loss: 0.3655 - val_accuracy: 0.8300
Epoch 96/100
313/313 [==============================] - 6s 20ms/step - loss: 0.3250 -
accuracy: 0.8610 - val_loss: 0.3676 - val_accuracy: 0.8230
Epoch 97/100
313/313 [==============================] - 5s 17ms/step - loss: 0.3220 -
accuracy: 0.8612 - val_loss: 0.3721 - val_accuracy: 0.8330
Epoch 98/100
313/313 [==============================] - 6s 20ms/step - loss: 0.3244 -
accuracy: 0.8607 - val_loss: 0.3781 - val_accuracy: 0.8240
Epoch 99/100
313/313 [==============================] - 6s 19ms/step - loss: 0.3197 -
accuracy: 0.8609 - val_loss: 0.4082 - val_accuracy: 0.8130
Epoch 100/100
313/313 [==============================] - 6s 18ms/step - loss: 0.3149 -
accuracy: 0.8629 - val_loss: 0.3496 - val_accuracy: 0.8350
```

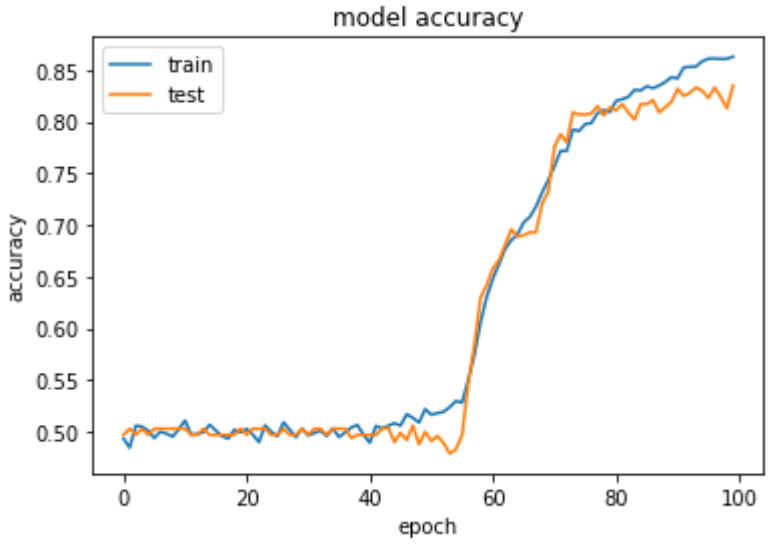
Evaluating the Model

Plotting Out Training History

In [60]:

```
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('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

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



In [61]:

```
#Save the model
model.save('chatbot_10.h5')
```

In [62]:

```
model.load_weights('chatbot_10.h5')
```

In [63]:

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

```
32/32 [==============================] - 1s 5ms/step
```

Out[63]:

```
array([[3.09907087e-08, 2.95258289e-08, 3.23730873e-08, ...,
        3.47191360e-08, 3.80365677e-08, 2.98733589e-08],
       [3.89142372e-08, 3.82839751e-08, 3.89094872e-08, ...,
        4.58020928e-08, 3.96928037e-08, 3.58063552e-08],
       [3.45161340e-08, 2.55341330e-08, 2.84235142e-08, ...,
        3.00708436e-08, 2.76699215e-08, 2.86630044e-08],
       ...,
       [8.15673502e-08, 9.23318879e-08, 9.81558230e-08, ...,
        1.02313834e-07, 8.36872545e-08, 7.83934979e-08],
       [5.87462168e-09, 4.54170346e-09, 4.94692731e-09, ...,
        4.97841945e-09, 4.64706673e-09, 4.76636020e-09],
       [4.42603714e-08, 3.49791307e-08, 3.71936366e-08, ...,
        4.09944683e-08, 4.10593408e-08, 4.43241532e-08]], dtype=float32)
```

Test the model on a brand new stroy and query

In [64]:

```
my_story = "John left the kitchen . Sandra dropped the football in the garden"
my_story.split()
```

Out[64]:

```
['John',
 'left',
 'the',
 'kitchen',
 '.',
 'Sandra',
 'dropped',
 'the',
 'football',
 'in',
 'the',
 'garden']
```

In [65]:

```
my_question = "Is the football in the garden ?"
my_question.split()
```

Out[65]:

```
['Is', 'the', 'football', 'in', 'the', 'garden', '?']
```

In [66]:

```
#Create dataset in the same format as training data
mydata = [(my_story.split(),my_question.split(), 'yes')]
mydata
```

Out[66]:

```
[(['John',
   'left',
   'the',
   'kitchen',
   '.',
   'Sandra',
   'dropped',
   'the',
   'football',
   'in',
   'the',
   'garden'],
  ['Is', 'the', 'football', 'in', 'the', 'garden', '?'],
  'yes')]
```

In [67]:

```
#Vectorize mydata
my_story, my_ques, my_ans = vectorize_stories(mydata)
```

In [68]:

```
my_story
```

Out[68]:

```
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,  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, 17, 13, 34,  4, 29,  3, 13, 21, 32, 13,  6]])
```

In [69]:

```
my_ques
```

Out[69]:

```
array([[13, 21, 32, 13,  6,  7]])
```

In [70]:

```
my_ans
```

Out[70]:

```
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., 1., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0.]])
```

In [71]:

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

```
1/1 [==============================] - 0s 158ms/step
```

Out[71]:

```
array([[4.6990927e-08, 3.3940616e-08, 3.5717463e-08, 3.3223571e-08,
        3.9938381e-08, 3.9252654e-08, 3.4986495e-08, 3.8550159e-08,
        3.6416644e-08, 3.9187270e-08, 4.1949612e-08, 3.8973642e-08,
        3.8412320e-08, 3.5924341e-08, 3.8187398e-08, 3.8528622e-08,
        3.7964369e-08, 3.9069057e-08, 3.2941262e-08, 3.8059781e-08,
        3.0868996e-08, 3.8555601e-08, 1.3599356e-01, 4.4126548e-08,
        3.1642106e-08, 3.6187458e-08, 8.6400509e-01, 3.1659976e-08,
        3.2056889e-08, 3.7009883e-08, 3.7989576e-08, 4.2547118e-08,
        3.9220470e-08, 3.8572594e-08, 4.4431768e-08, 3.9599545e-08,
        3.3852686e-08, 4.2501863e-08]], dtype=float32)
```

In [72]:

```
val_max = np.argmax(pred_results[0])
for key,val in tokenizer.word_index.items():
    if val == val_max:
        print(key)
```

```
yes
```

DONE!!