Name- Sandipan Mukherjee

LETSGROWMORE INTERNSHIP :

03) ADVANCED LEVEL TASK -

2)Next Word Prediction:

Import library

In [15]:
```
import numpy as np
from nltk.tokenize import RegexpTokenizer
from keras.models import Sequential, load_model
from keras.layers import LSTM
from keras.layers.core import Dense, Activation
from tensorflow.keras.optimizers import RMSprop
import matplotlib.pyplot as plt
import pickle
import heapq
```

Load the Data

In [16]:
```
path = '1661-0.txt'
text = open(path, encoding="utf8").read().lower()
print('corpus length:', len(text))
```

```
corpus length: 581888
```

Split the Dataset

In [17]:
```
tokenizer = RegexpTokenizer(r'\w+')
words = tokenizer.tokenize(text)
```

In [18]:
```
unique_words = np.unique(words)
unique_word_index = dict((c, i) for i, c in enumerate(unique_words))
```

Taking whatever information we have about our problem and turning it into numbers that we can use to build our feature matrix.

In [19]:
```
WORD_LENGTH = 7
prev_words = []
next_words = []
for i in range(len(words) - WORD_LENGTH):
    prev_words.append(words[i:i + WORD_LENGTH])
    next_words.append(words[i + WORD_LENGTH])
print(prev_words[0])
print(next_words[0])
```

```
['project', 'gutenberg', 's', 'the', 'adventures', 'of', 'sherlock']
holmes
```

Iterate x and y if the word is available so that the corresponding position becomes 1.

In [20]:
```
X = np.zeros((len(prev_words), WORD_LENGTH, len(unique_words)), dtype=bool)
Y = np.zeros((len(next_words), len(unique_words)), dtype=bool)
for i, each_words in enumerate(prev_words):
    for j, each_word in enumerate(each_words):
        X[i, j, unique_word_index[each_word]] = 1
        Y[i, unique_word_index[next_words[i]]] = 1
print(X[0][0])
```

```
[False False False ... False False False]
```

Building the Recurrent Neural network

In [21]:
```
model = Sequential()
model.add(LSTM(128, input_shape=(WORD_LENGTH, len(unique_words))))
model.add(Dense(len(unique_words)))
model.add(Activation('softmax'))
```

Training the Next Word Prediction Model

In [22]:
```
optimizer = RMSprop(lr=0.01)
model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
history = model.fit(X, Y, validation_split=0.05, batch_size=128, epochs=8, shuffle=True).history
```

```
c:\Users\acer\appdata\local\programs\python\python38\lib\site-packages\keras\optimizer_v2\optimizer_v2.py:355: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  warnings.warn(
Epoch 1/8
811/811 [==============================] - 350s 423ms/step - loss: 6.0074 - accuracy: 0.1066 - val_loss: 7.0023 - val_accuracy: 0.1003
Epoch 2/8
811/811 [==============================] - 244s 301ms/step - loss: 5.7722 - accuracy: 0.1489 - val_loss: 7.9543 - val_accuracy: 0.1046
Epoch 3/8
811/811 [==============================] - 324s 400ms/step - loss: 5.7436 - accuracy: 0.1764 - val_loss: 7.9943 - val_accuracy: 0.1110
Epoch 4/8
811/811 [==============================] - 264s 326ms/step - loss: 5.4341 - accuracy: 0.2125 - val_loss: 8.2855 - val_accuracy: 0.1060
Epoch 5/8
811/811 [==============================] - 278s 343ms/step - loss: 5.0695 - accuracy: 0.2511 - val_loss: 8.4223 - val_accuracy: 0.0925
Epoch 6/8
811/811 [==============================] - 255s 315ms/step - loss: 4.7160 - accuracy: 0.2919 - val_loss: 8.8054 - val_accuracy: 0.0914
Epoch 7/8
811/811 [==============================] - 259s 320ms/step - loss: 4.3364 - accuracy: 0.3360 - val_loss: 8.8121 - val_accuracy: 0.0804
Epoch 8/8
811/811 [==============================] - 322s 398ms/step - loss: 3.9666 - accuracy: 0.3791 - val_loss: 9.3285 - val_accuracy: 0.0786
```

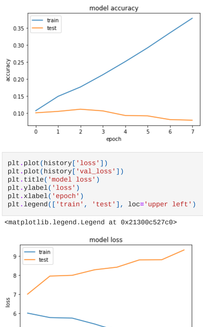
In [25]:
```
model.save('keras_next_word_model.h5')
pickle.dump(history, open("history.p", "wb"))
model = load_model('keras_next_word_model.h5')
history = pickle.load(open("history.p", "rb"))
history
```

Out[25]:
```
{'loss': [6.007428169250488,
  5.772189617156982,
  5.743634223937988,
  5.434145927429199,
  5.069524765014648,
  4.715907955188185,
  4.336385726928711,
  3.9666139542755127],
 'accuracy': [0.10656527429819107,
  0.14887526631355286,
  0.17643940448760986,
  0.21247518062591553,
  0.25114208459854126,
  0.2918714588290704,
  0.3359740097518921,
  0.3791033029556274],
 'val_loss': [7.002333886260986,
  7.954339504241943,
  7.994301795959473,
  8.285538673400879,
  8.422257423400879,
  8.805442810058594,
  8.812137603759766,
  9.328510284423828],
 'val_accuracy': [0.10034792125225067,
  0.10455960780382156,
  0.11096686124801636,
  0.10602454096078873,
  0.09247390925884247,
  0.09137520933151171,
  0.08038821071386337,
  0.07855704426765442]}
```

Evaluating the Next Word Prediction Model

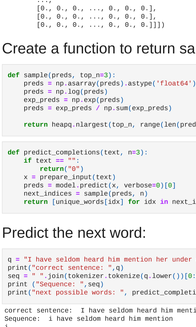
In [26]:
```
plt.plot(history['accuracy'])
plt.plot(history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
```

Out[26]:
```
<matplotlib.legend.Legend at 0x2130dfcd160>
```



In [27]:
```
plt.plot(history['loss'])
plt.plot(history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
```

Out[27]:
```
<matplotlib.legend.Legend at 0x21300c527c0>
```



Testing Next Word Prediction Model

In [28]:
```
def prepare_input(text):
    x = np.zeros((1, WORD_LENGTH, len(unique_words)))
    for t, word in enumerate(text.split()):
        print(word)
        x[0, t, unique_word_index[word]] = 1
    return x
prepare_input("It is not a lack".lower())
```

```
it
is
not
a
lack
```

Out[28]:
```
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.]]])
```

Create a function to return samples:

In [29]:
```
def sample(preds, top_n=3):
    preds = np.asarray(preds).astype('float64')
    preds = np.log(preds)
    exp_preds = np.exp(preds)
    preds = exp_preds / np.sum(exp_preds)

    return heapq.nlargest(top_n, range(len(preds)), preds.take)
```

In [30]:
```
def predict_completions(text, n=3):
    if text == "":
        return("0")
    x = prepare_input(text)
    preds = model.predict(x, verbose=0)[0]
    next_indices = sample(preds, n)
    return [unique_words[idx] for idx in next_indices]
```

Predict the next word:

In [36]:
```
q = "I have seldom heard him mention her under any other name"
print("correct sentence: ",q)
seq = " ".join(tokenizer.tokenize(q.lower())[0:6])
print ("Sequence: ",seq)
print("next possible words: ", predict_completions(seq, 6))
```

```
correct sentence:  I have seldom heard him mention her under any other name
Sequence:  i have seldom heard him mention
i
have
seldom
heard
him
mention
next possible words:  ['who', 'there', 'her', 'in', 'of', 'about']
```

In [ ]:
```
```