-

Advance level :Task 2

Next Word Prediction

- Objective:Using Tensorflow and Keras library train a RNN, to predict the next word.
- dataset:https://drive.google.com/file/d/1GeUzNVqiixXHnTlBoNiQ2W3CynX_Isu2/view
- Author :Mayuri arun Pathak,Data Science Intern

- Set working directories

In [3]:
```
import os
os.chdir("H:\Data Science\Internship\Spark")
```

- Importing Required library set

In [4]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
import heapq
from nltk.tokenize import RegexpTokenizer
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Dense, Activation
from tensorflow.keras.layers import LSTM, Dropout
from tensorflow.keras.layers import TimeDistributed
from tensorflow.keras.layers import Dense, Activation, Dropout, RepeatVector
from tensorflow.keras.optimizers import  RMSprop
import pickle
```

- Loading the dataset

In [5]:
```
text= open('1661-0.txt',encoding='UTF-8').read().lower()
print('corpus length:',len(text))
```

```
corpus length: 581888
```

- Splitting the entire dataset into each word in order without the presence of special characters

In [6]:
```
tokenizer = RegexpTokenizer(r'\w+')
words = tokenizer.tokenize(text)
```

- Dictionary(<key: value>) with each word form the unique_words list as key and its corresponding position as value

In [14]:
```
unique_words = np.unique(words)
unique_word_index = dict((c, i) for i, c in enumerate(unique_words))
```

- Feature engineering

In [15]:
```
WORD_LENGTH = 5
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
```

- One-Hot encoding

In [16]:
```
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
```

In [17]:
```
print(X[0][0])
```

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

- Building the model

In [18]:
```
model = Sequential()
model.add(LSTM(128, input_shape=(WORD_LENGTH, len(unique_words))))
model.add(Dense(len(unique_words)))
model.add(Activation('softmax'))
```

- Training

In [19]:
```
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=10, shuffle=True).history
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.RMSprop.
Epoch 1/10
811/811 [==============================] - 307s 374ms/step - loss: 6.7015 - accuracy: 0.0528 - val_loss: 7.2469 - val_accuracy: 0.0538
Epoch 2/10
811/811 [==============================] - 333s 411ms/step - loss: 6.4829 - accuracy: 0.0531 - val_loss: 7.2532 - val_accuracy: 0.0538
Epoch 3/10
811/811 [==============================] - 311s 384ms/step - loss: 6.4414 - accuracy: 0.0536 - val_loss: 7.2098 - val_accuracy: 0.0536
Epoch 4/10
811/811 [==============================] - 292s 359ms/step - loss: 6.3786 - accuracy: 0.0627 - val_loss: 7.1669 - val_accuracy: 0.0624
Epoch 5/10
811/811 [==============================] - 334s 411ms/step - loss: 6.2937 - accuracy: 0.0717 - val_loss: 7.0640 - val_accuracy: 0.0738
Epoch 6/10
811/811 [==============================] - 285s 351ms/step - loss: 6.2000 - accuracy: 0.0812 - val_loss: 7.0297 - val_accuracy: 0.0771
Epoch 7/10
811/811 [==============================] - 295s 364ms/step - loss: 6.1265 - accuracy: 0.0885 - val_loss: 6.9881 - val_accuracy: 0.0767
Epoch 8/10
811/811 [==============================] - 271s 334ms/step - loss: 6.0669 - accuracy: 0.0943 - val_loss: 6.9473 - val_accuracy: 0.0831
Epoch 9/10
811/811 [==============================] - 260s 321ms/step - loss: 6.0171 - accuracy: 0.0985 - val_loss: 6.9448 - val_accuracy: 0.0807
Epoch 10/10
811/811 [==============================] - 252s 310ms/step - loss: 5.9714 - accuracy: 0.1026 - val_loss: 6.9539 - val_accuracy: 0.0802
```

- Saving the model and loading it back

In [20]:
```
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"))
```

In [21]:
```
history
```

Out[21]:
```
{'loss': [6.701492786407471,
  6.482902050018311,
  6.441351413726807,
  6.378600120544434,
  6.293696618461609,
  6.200011730194092,
  6.126471996307373,
  6.066859960174475,
  6.017075061798096,
  5.971368312835693],
 'accuracy': [0.05279542133212089,
  0.05314237070846558,
  0.05360499024391174,
  0.06274147987295227,
  0.07167571038007736,
  0.08121705055236816,
  0.08845497667789459,
  0.09434362506890869,
  0.09884856450366974,
  0.10258387029170990],
 'val_loss': [7.248867654037764,
  7.253160953521729,
  7.209777355194092,
  7.166945457458496,
  7.063969135284424,
  7.029075002670288,
  6.988082408905029,
  6.947271347045898,
  6.944813251495361,
  6.953916549682617],
 'val_accuracy': [0.05382643640041351,
  0.05382643640041351,
  0.05364335328459358,
  0.06243134289979935,
  0.07737824961543083,
  0.07707799226045609,
  0.07671182692043967,
  0.08311973512172699,
  0.08072965466082027,
  0.08019040524959564]}
```

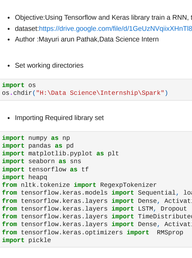
- Evaluation

In [22]:
```
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[22]:
```
<matplotlib.legend.Legend at 0x1f6a2cd6c70>
```


In [23]:
```
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[23]:
```
<matplotlib.legend.Legend at 0x1f6a2c44160>
```



- Prediction

In [24]:
```
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[24]:
```
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.]]])
```

In [25]:
```
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 [26]:
```
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]
```

In [27]:
```
q =  "There is nothing more deceptive than an obvious fact"
print("correct sentence: ",q)
seq = " ".join(tokenizer.tokenize(q.lower())[0:5])
print("Sequence: ",seq)
print("next possible words: ", predict_completions(seq, 5))
```

```
correct sentence:  There is nothing more deceptive than an obvious fact
Sequence:  there is nothing more deceptive
there
is
nothing
more
deceptive
next possible words:  ['to', 'the', 'i', 'that', 'it']
```

Suggestions are always welcome!

Thank You!