```
%tensorflow_version 2.x
import tensorflow as tf
import string
import requests


response = requests.get('https://ocw.mit.edu/ans7870/6/6.006/s08/lecturenotes/files/t8.shakes
#response.text[:1500]


data = response.text.split('\n')
data = data[253:]
#len(data)
data = " ".join(data)


def clean_text(doc):
  tokens = doc.split()
  table = str.maketrans('', '', string.punctuation)
  tokens = [w.translate(table) for w in tokens]
  tokens = [word for word in tokens if word.isalpha()]
  tokens = [word.lower() for word in tokens]
  return tokens

tokens = clean_text(data)
print(tokens[:50])
```

```
['from', 'fairest', 'creatures', 'we', 'desire', 'increase', 'that', 'thereby', 'beautys
```

```
#len(tokens)
len(set(tokens))
```

```
27956
```

Let's use a set of 50 words to predict the 51st word

```
length = 50 + 1
lines = []

for i in range(length, len(tokens)):
  seq = tokens[i-length:i]
  line = ' '.join(seq)
  lines.append(line)
  if i > 200000:
    break

print(len(lines))
#lines[0]
```

```
 - -
```

```
199951
```

```
type(lines)
```

```
list
```

Building the LSTM model

```
import numpy as np
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Embedding
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(lines)
sequences = tokenizer.texts_to_sequences(lines)
```

```
sequences = np.array(sequences)
X, y = sequences[:, :-1], sequences[:,-1]
```

```
vocab_size = len(tokenizer.word_index) + 1
y = to_categorical(y, num_classes=vocab_size)
seq_length = X.shape[1]
seq_length
```

```
50
```

```
model = Sequential()
model.add(Embedding(vocab_size, 50, input_length=seq_length))
model.add(LSTM(100, return_sequences=True))
model.add(LSTM(100))
model.add(Dense(100, activation='relu'))
model.add(Dense(vocab_size, activation='softmax'))
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding (Embedding)        (None, 50, 50)            650450    
_________________________________________________________________
lstm (LSTM)                  (None, 50, 100)           60400     
_________________________________________________________________
```

```
lstm_1 (LSTM)                (None, 100)               80400
_________________________________________________________________
dense (Dense)                (None, 100)               10100
_________________________________________________________________
dense_1 (Dense)              (None, 13009)             1313909
=================================================================
Total params: 2,115,259
Trainable params: 2,115,259
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(loss = 'categorical_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
model.fit(X, y, batch_size = 256, epochs = 100)
```

```
Epoch 73/100
782/782 [==============================] - 46s 59ms/step - loss: 3.4001 - accuracy:
Epoch 74/100
782/782 [==============================] - 45s 58ms/step - loss: 3.3833 - accuracy:
Epoch 75/100
782/782 [==============================] - 45s 58ms/step - loss: 3.3677 - accuracy:
Epoch 76/100
782/782 [==============================] - 45s 58ms/step - loss: 3.3526 - accuracy:
Epoch 77/100
782/782 [==============================] - 45s 58ms/step - loss: 3.3375 - accuracy:
Epoch 78/100
782/782 [==============================] - 45s 58ms/step - loss: 3.3199 - accuracy:
Epoch 79/100
782/782 [==============================] - 45s 58ms/step - loss: 3.3075 - accuracy:
Epoch 80/100
782/782 [==============================] - 45s 58ms/step - loss: 3.2921 - accuracy:
Epoch 81/100
782/782 [==============================] - 46s 58ms/step - loss: 3.2784 - accuracy:
Epoch 82/100
782/782 [==============================] - 46s 58ms/step - loss: 3.2602 - accuracy:
Epoch 83/100
782/782 [==============================] - 45s 58ms/step - loss: 3.2477 - accuracy:
Epoch 84/100
782/782 [==============================] - 46s 59ms/step - loss: 3.2317 - accuracy:
Epoch 85/100
782/782 [==============================] - 46s 59ms/step - loss: 3.2185 - accuracy:
Epoch 86/100
782/782 [==============================] - 45s 58ms/step - loss: 3.2049 - accuracy:
Epoch 87/100
782/782 [==============================] - 45s 58ms/step - loss: 3.1907 - accuracy:
Epoch 88/100
782/782 [==============================] - 45s 58ms/step - loss: 3.1767 - accuracy:
Epoch 89/100
782/782 [==============================] - 45s 58ms/step - loss: 3.1631 - accuracy:
Epoch 90/100
782/782 [==============================] - 45s 58ms/step - loss: 3.1477 - accuracy:
Epoch 91/100
782/782 [==============================] - 45s 58ms/step - loss: 3.1367 - accuracy:
Epoch 92/100
782/782 [==============================] - 45s 58ms/step - loss: 3.1212 - accuracy:
Epoch 93/100
782/782 [==============================] - 45s 58ms/step - loss: 3.1083 - accuracy:
```

```
Epoch 94/100
782/782 [==============================] - 46s 58ms/step - loss: 3.0961 - accuracy:
Epoch 95/100
782/782 [==============================] - 45s 58ms/step - loss: 3.0860 - accuracy:
Epoch 96/100
782/782 [==============================] - 45s 58ms/step - loss: 3.0689 - accuracy:
Epoch 97/100
782/782 [==============================] - 45s 58ms/step - loss: 3.0549 - accuracy:
Epoch 98/100
782/782 [==============================] - 46s 59ms/step - loss: 3.0450 - accuracy:
Epoch 99/100
782/782 [==============================] - 46s 59ms/step - loss: 3.0316 - accuracy:
Epoch 100/100
782/782 [==============================] - 46s 59ms/step - loss: 3.0238 - accuracy:
<tensorflow.python.keras.callbacks.History at 0x7f1f0f24d210>
```

```
seed_text=lines[12343]
seed_text
```

```
'home of love if i have ranged like him that travels i return again just to the time not
y self bring water for my stain never believe though in my nature reigned all frailties
at it could so'
```

```
def generate_text_seq(model, tokenizer, text_seq_length, seed_text, n_words):
  text = []

  for _ in range(n_words):
    encoded = tokenizer.texts_to_sequences([seed_text])[0]
    encoded = pad_sequences([encoded], maxlen = text_seq_length, truncating='pre')

    y_predict = model.predict_classes(encoded)

    predicted_word = ''
    for word, index in tokenizer.word_index.items():
      if index == y_predict:
        predicted_word = word
        break
    seed_text = seed_text + ' ' + predicted_word
    text.append(predicted_word)
  return ' '.join(text)
```

```
generate_text_seq(model, tokenizer, seq_length, seed_text, 100)
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/sequential.py:455
  warnings.warn('`model.predict_classes()` is deprecated and '
'preposterously be stained to leave to fail and by the fatness of our senseless judgment
world the virginal infectious second servant of the world and let me hear of him and abo
angd of them brats in a perpetual dulness let their hast been within and we in solemn ho
ings that milks and does the prisond walls tables therein and lack extinct in take and f
ore strong horrible shaming a crooked sentence of barren senators and desprate creature
```

✓ 4s completed at 4:10 PM