Text Generation NLP Model

Here is a text generation NLP model that is trained on a collection of Shakespearian poems called "The Sonnets". The text is generated by an LSTM network that learns both English and the underlying patterns of Shakespeare's style. You can start the poem and it will finish it.

In [1]:

```
# import libraries
import sys
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from keras.models import Model
from keras.layers import Input, LSTM, Dense, Dropout, Bidirectional
from keras.callbacks import ModelCheckpoint
from keras.optimizers import Adam, RMSprop
from keras.utils import np_utils
```

```
Using TensorFlow backend.
```

In [2]:

```
# load the Shakespeare text & convert it to lower case
data = open('data/shakespear.txt', 'r').read()
data= data.lower()
# review the data
chars = sorted(list(set(data)))
print("Here is list of unique characters: \n" + str(chars))
n_chars, n_vocab = len(data), len(chars)
print("Total characters: ", n_chars)
print("Total vocabulary: ", n_vocab)
```

```
Here is list of unique characters:
['\n', ' ', '!', "'", '(', ')', ',', '-', '.', ':', ';', '?', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j
', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
Total characters:  94249
Total vocabulary:  38
```

In [3]:

```
# create python dictionaries mapping chars-to-integers and integers-to-chars
char_to_int = { ch:i for i,ch in enumerate(chars) }
int_to_char = { i:ch for i,ch in enumerate(chars) }
print(char_to_int)
print("-----------")
print(int_to_char)
```

```
{'\n': 0, ' ': 1, '!': 2, "'": 3, '(': 4, ')': 5, ',': 6, '-': 7, '.': 8, ':': 9, ';': 10, '?': 11, 'a': 12,
'b': 13, 'c': 14, 'd': 15, 'e': 16, 'f': 17, 'g': 18, 'h': 19, 'i': 20, 'j': 21, 'k': 22, 'l': 23, 'm': 24, '
n': 25, 'o': 26, 'p': 27, 'q': 28, 'r': 29, 's': 30, 't': 31, 'u': 32, 'v': 33, 'w': 34, 'x': 35, 'y': 36, 'z
': 37}
-----------
{0: '\n', 1: ' ', 2: '!', 3: "'", 4: '(', 5: ')', 6: ',', 7: '-', 8: '.', 9: ':', 10: ';', 11: '?', 12: 'a',
13: 'b', 14: 'c', 15: 'd', 16: 'e', 17: 'f', 18: 'g', 19: 'h', 20: 'i', 21: 'j', 22: 'k', 23: 'l', 24: 'm',
25: 'n', 26: 'o', 27: 'p', 28: 'q', 29: 'r', 30: 's', 31: 't', 32: 'u', 33: 'v', 34: 'w', 35: 'x', 36: 'y',
37: 'z'}
```

In [4]:

```python
# determine average line length
lines = data.splitlines()
sentences = []
for i in range(len(lines)):
    if len(lines[i]) > 0:
        sentences.append(lines[i])
print("Number of sentences:", len(sentences))
result = [len(x) for x in sentences]
print("Sentence mean length", np.mean(result))
```

```
Number of sentences: 2176
Sentence mean length 42.18014705882353
```

In [5]:

```python
# transform the dataset into a supervised learning set of input to output pairs encoded as integers
max_length = 42
step = 2
dataX = []
dataY = []
for i in range(0, n_chars - max_length, step):
    seq_in = data[i:i + max_length]
    seq_out = data[i + max_length]
    dataX.append([char_to_int[char] for char in seq_in])
    dataY.append(char_to_int[seq_out])
n_patterns = len(dataX)
print("Total Patterns: ", n_patterns)
```

```
Total Patterns:  47104
```

In [6]:

```python
# reshape input for LSTM [samples, time steps, features]
X = np.reshape(dataX, (n_patterns, max_length, 1))
print("Input shape:", X.shape)
# normalize the data
X = X / float(n_vocab)
# one-hot encode the output variable
y = np_utils.to_categorical(dataY)
print("Output shape:", y.shape)
```

```
Input shape: (47104, 42, 1)
Output shape: (47104, 38)
```

In [7]:

```
# define the LSTM model
def lstm_model(X, y):
    inputs = Input(shape=(X.shape[1], X.shape[2]))
    L1 = LSTM(256, return_sequences=True)(inputs)
    D1 = Dropout(0.2)(L1)
    L2 = LSTM(256, return_sequences=False)(D1)
    D2 = Dropout(0.2)(L2)
    output = Dense(y.shape[1], activation='softmax')(D2)
    model = Model(inputs=inputs, outputs=output)
    return model
```

In [8]:

```
# create the LSTM model
model = lstm_model(X, y)
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 42, 1)             0
_________________________________________________________________
lstm_1 (LSTM)                (None, 42, 256)           264192
_________________________________________________________________
dropout_1 (Dropout)          (None, 42, 256)           0
_________________________________________________________________
lstm_2 (LSTM)                (None, 256)               525312
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 38)                9766
=================================================================
Total params: 799,270
Trainable params: 799,270
Non-trainable params: 0
_________________________________________________________________
```

In [9]:

```
# fit the model
nb_epochs = 500
history = model.fit(X, y, epochs=nb_epochs, batch_size=64, shuffle=True).history
```

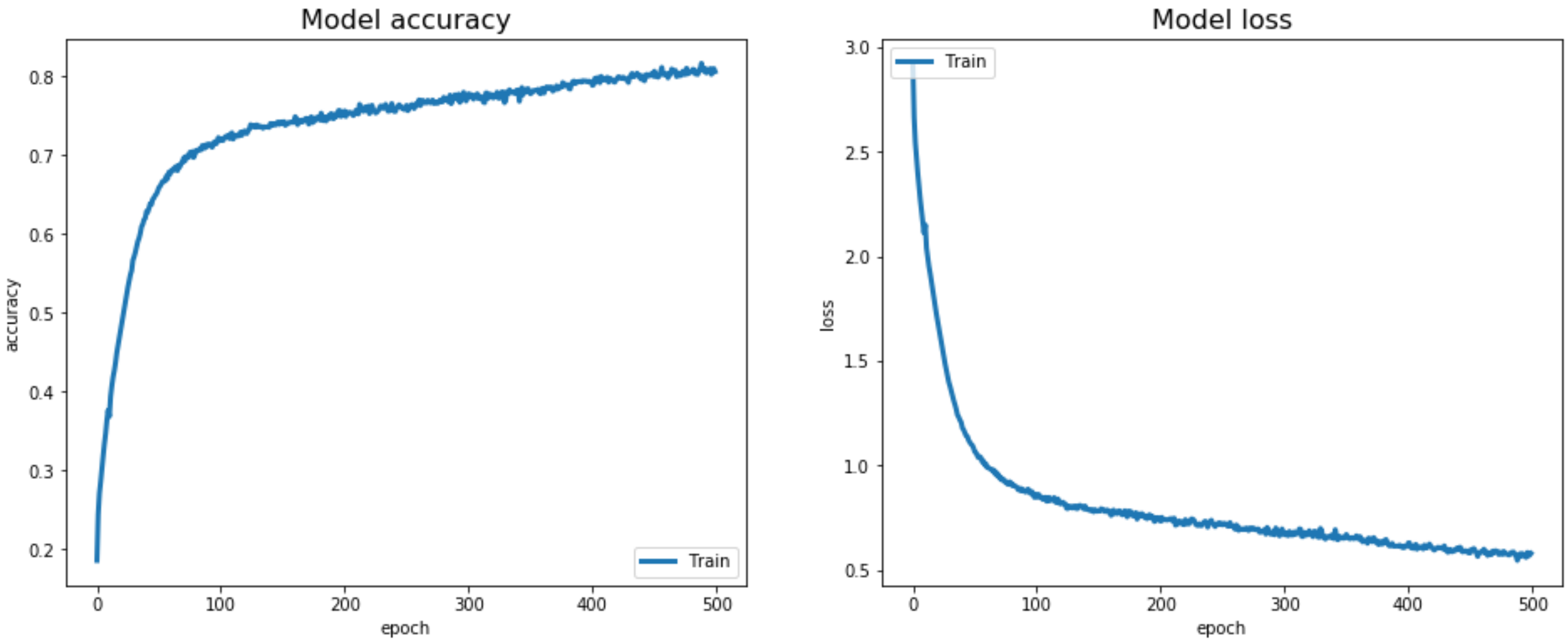
```
Epoch 1/500
47104/47104 [==============================] - 250s 5ms/step - loss: 2.9235 - acc: 0.1858
Epoch 2/500
47104/47104 [==============================] - 202s 4ms/step - loss: 2.6663 - acc: 0.2439
Epoch 3/500
47104/47104 [==============================] - 201s 4ms/step - loss: 2.5430 - acc: 0.2709
Epoch 4/500
47104/47104 [==============================] - 193s 4ms/step - loss: 2.4637 - acc: 0.2856
Epoch 5/500
47104/47104 [==============================] - 192s 4ms/step - loss: 2.3949 - acc: 0.3030
Epoch 6/500
47104/47104 [==============================] - 192s 4ms/step - loss: 2.3282 - acc: 0.3183
Epoch 7/500
47104/47104 [==============================] - 216s 5ms/step - loss: 2.2699 - acc: 0.3336
Epoch 8/500
47104/47104 [==============================] - 262s 6ms/step - loss: 2.2147 - acc: 0.3465
Epoch 9/500
47104/47104 [==============================] - 232s 5ms/step - loss: 2.1634 - acc: 0.3632
Epoch 10/500
47104/47104 [==============================] - 202s 4ms/step - loss: 2.1120 - acc: 0.3762
Epoch 11/500
47104/47104 [==============================] - 250s 5ms/step - loss: 2.1512 - acc: 0.3682
Epoch 12/500
47104/47104 [==============================] - 285s 6ms/step - loss: 2.0402 - acc: 0.3935
Epoch 13/500
47104/47104 [==============================] - 201s 4ms/step - loss: 1.9961 - acc: 0.4072
Epoch 14/500
47104/47104 [==============================] - 191s 4ms/step - loss: 1.9507 - acc: 0.4183
Epoch 15/500
47104/47104 [==============================] - 191s 4ms/step - loss: 1.9173 - acc: 0.4264
Epoch 16/500
47104/47104 [==============================] - 191s 4ms/step - loss: 1.8781 - acc: 0.4366
Epoch 17/500
47104/47104 [==============================] - 195s 4ms/step - loss: 1.8362 - acc: 0.4490
Epoch 18/500
47104/47104 [==============================] - 191s 4ms/step - loss: 1.8011 - acc: 0.4583
Epoch 19/500
47104/47104 [==============================] - 215s 5ms/step - loss: 1.7581 - acc: 0.4674
Epoch 20/500
47104/47104 [==============================] - 205s 4ms/step - loss: 1.7239 - acc: 0.4763
Epoch 21/500
47104/47104 [==============================] - 208s 4ms/step - loss: 1.6895 - acc: 0.4851
Epoch 22/500
47104/47104 [==============================] - 216s 5ms/step - loss: 1.6551 - acc: 0.4964
Epoch 23/500
47104/47104 [==============================] - 206s 4ms/step - loss: 1.6224 - acc: 0.5036
```

In [10]:

```
# plot the model loss and accuracy
fig, (axis1, axis2) = plt.subplots(nrows=1, ncols=2, figsize=(16,6))

# summarize history for accuracy
axis1.plot(history['acc'], label='Train', linewidth=3)
axis1.set_title('Model accuracy', fontsize=16)
axis1.set_ylabel('accuracy')
axis1.set_xlabel('epoch')
axis1.legend(loc='lower right')

# summarize history for loss
axis2.plot(history['loss'], label='Train', linewidth=3)
axis2.set_title('Model loss', fontsize=16)
axis2.set_ylabel('loss')
axis2.set_xlabel('epoch')
axis2.legend(loc='upper left')
plt.show()
```



In [21]:

```
# enter a starting sequence for your poem
user_input = input("Write the beginning of your poem, the Shakespeare machine will complete it. \n")
```

```
Write the beginning of your poem, the Shakespeare machine will complete it.
While walking on the river Nye
```

In [22]:

```
# generate characters
user_input = user_input.lower()
print(user_input)
start = np.random.randint(0, len(dataX)-1)
pattern = dataX[start]

for i in range(1200):
    x = np.reshape(pattern, (1, len(pattern), 1))
    x = x / float(n_vocab)
    prediction = model.predict(x)
    index = np.argmax(prediction)
    result = int_to_char[index]
    seq_in = [int_to_char[value] for value in pattern]
    sys.stdout.write(result)
    pattern.append(index)
    pattern = pattern[1:len(pattern)]
```

```
while walking on the river nye
and your tweet beauty ssill worle soace,
oi mine own srnre,soose,
and that hs not five peeeepede wourh,
and thou af wes boti dompradt thay woe,
and live ooee ges filders,
bnd thoughts the sueet leases that thou shall gare drtented face,
s eresi where thou mayst pays i tene i llf,
that thou art braysesy mine thmiui heart to sueetls gave:
the couald ceed the loves twaie the clrttacl of the vornd and clay,
farh paie my love stat fair,
these blassed oe thy hood thoughts, what thou miehtsy st thou should forming hours,
fuen sic better part of mine eye sieht speae,
where thall h laykng of your giarnen eoees,
oo mf the semnose of the world despiss,
comning thy braint bnd doth meed geav,
and prou becrty do inher in me,
lor when shes blans beeo wes i love you seale shou presartelcee
of alote,
then look did fxclcnee as iin ien he ragn,
and brisher thy srngues shat peptre shou art the lake,
that thou mo heart the buiu, of that there be gair dishr ly self drau,
for that fase thee that i do nay lane ne minde stove,
mov my potr love soor that ieave to weel,
ay thou n'er mot thinking bue in she midh,
to tout the sratsed face at the sace, ayt that is not bosseii that i have het,
ast then i pfee th
```

In [ ]: