Github Link:

This is for implementing RNN using keras for Text Generation.

```
# importing essential libraries
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import LSTM
from keras.utils import np_utils
```

My input file will be a section of a play from the playwright genius Shakespeare. I will be using a monologue from Othello.

```
#Firstly formatting the input_file so that it can be well understood by keras

#Read the data, turn it into lower case
data = open("input_file.txt").read().lower()
```

```
#This get the set of characters used in the data and sorts them
chars = sorted(list(set(data)))
chars
```

```
     ['\n',
      ' ',
      "'",
      ',',
      '-',
      '.',
      ';',
      'a',
      'b',
      'c',
      'd',
      'e',
      'f',
      'g',
      'h',
      'i',
      'k',
      'l',
      'm',
      'n',
      'o',
      'p',
      'q',
      'r',
      's',
      't',
      'u',
      'v',
      'w',
      'y']
```

```
#Total number of characters used in the data
totalChars = len(data)
totalChars
```

```
     1860
```

```
#Number of unique chars
numberOfUniqueChars = len(chars)
numberOfUniqueChars
```

```
     30
```

create a dictionary of each character so it can be easily represented

```
#This allows for characters to be represented by numbers
CharsForids = {char:Id for Id, char in enumerate(chars)}
CharsForids
```

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

```
#This is the opposite to the above
idsForChars = {Id:char for Id, char in enumerate(chars)}
idsForChars
```

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

```
#How many timesteps e.g how many characters we want to process in one go
numberOfCharsToLearn = 100
```

```
CharsForids["o"]
```

```
20
```

```
#Input data
charX = []
#Output data
y = []


#Since our timestep sequence represetns a process for every 100 chars we omit
#the first 100 chars so the loop runs a 100 less or there will be index out of
#range
counter = totalChars - numberOfCharsToLearn
#This loops through all the characters in the data skipping the first 100
for i in range(0, counter, 1):
  #This one goes from 0-100 so it gets 100 values starting from 0 and stops
  #just before the 100th value
  theInputChars = data[i:i+numberOfCharsToLearn]
  #With no ':' you start with 0, and so you get the actual 100th value
  #Essentially, the output Chars is the next char in line for those 100 chars in charX
  theOutputChars = data[i + numberOfCharsToLearn]
  #Appends every 100 chars ids as a list into charX
  charX.append([CharsForids[char] for char in theInputChars])
  #For every 100 values there is one y value which is the output
  y.append(CharsForids[theOutputChars])
```

To convert data into right format which can be fed to RNN

```
#Len(charX) represents how many of those time steps we have
#The numberOfCharsToLearn is how many character we process
#Our features are set to 1 because in the output we are only predicting 1 char
X = np.reshape(charX, (len(charX), numberOfCharsToLearn, 1))
```

```
#This is done for normalization
X = X/float(numberOfUniqueChars)
#This sets it up for us so we can have a categorical(#feature) output format
y = np_utils.to_categorical(y)
print(y)
```

```
[[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. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

Building RNN Model

```
 model = Sequential()



#Since we know the shape of our Data we can input the timestep and feature data
#The number of timestep sequence are dealt with in the fit function
model.add(LSTM(256, input_shape=(X.shape[1], X.shape[2])))
model.add(Dropout(0.2))



#number of features on the output
model.add(Dense(y.shape[1], activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam')
model.fit(X, y, epochs=50, batch_size=128)
model.save_weights("Othello.hdf5")
#model.load_weights("Othello.hdf5")
```

```
    14/14 [==============================] - 0s 20ms/step - loss: 2.9859
    Epoch 22/50
    14/14 [==============================] - 0s 20ms/step - loss: 2.9728
    Epoch 23/50
    14/14 [==============================] - 0s 21ms/step - loss: 2.9724
    Epoch 24/50
    14/14 [==============================] - 0s 20ms/step - loss: 2.9849
    Epoch 25/50
    14/14 [==============================] - 0s 20ms/step - loss: 2.9853
    Epoch 26/50
    14/14 [==============================] - 0s 20ms/step - loss: 2.9957
    Epoch 27/50
    14/14 [==============================] - 0s 20ms/step - loss: 2.9782
    Epoch 28/50
    14/14 [==============================] - 0s 20ms/step - loss: 2.9466
    Epoch 29/50
    14/14 [==============================] - 0s 20ms/step - loss: 2.9873
    Epoch 30/50
    14/14 [==============================] - 0s 20ms/step - loss: 2.9842
    Epoch 31/50
    14/14 [==============================] - 0s 20ms/step - loss: 2.9633
    Epoch 32/50
    14/14 [==============================] - 0s 21ms/step - loss: 2.9332
    Epoch 33/50
    14/14 [==============================] - 0s 20ms/step - loss: 2.9575
    Epoch 34/50
    14/14 [==============================] - 0s 21ms/step - loss: 2.9160
    Epoch 35/50
    14/14 [==============================] - 0s 20ms/step - loss: 2.9807
    Epoch 36/50
    14/14 [==============================] - 0s 20ms/step - loss: 2.9312
    Epoch 37/50
    14/14 [==============================] - 0s 20ms/step - loss: 2.9230
    Epoch 38/50
    14/14 [==============================] - 0s 21ms/step - loss: 2.9103
    Epoch 39/50
    14/14 [==============================] - 0s 20ms/step - loss: 2.9131
    Epoch 40/50
```

```
14/14 [==============================] - 0s 20ms/step - loss: 2.8855
Epoch 41/50
14/14 [==============================] - 0s 20ms/step - loss: 2.8953
Epoch 42/50
14/14 [==============================] - 0s 20ms/step - loss: 2.8759
Epoch 43/50
14/14 [==============================] - 0s 20ms/step - loss: 2.8767
Epoch 44/50
14/14 [==============================] - 0s 20ms/step - loss: 2.8906
Epoch 45/50
14/14 [==============================] - 0s 20ms/step - loss: 2.8888
Epoch 46/50
14/14 [==============================] - 0s 20ms/step - loss: 2.8833
Epoch 47/50
14/14 [==============================] - 0s 20ms/step - loss: 2.8987

Epoch 48/50
14/14 [==============================] - 0s 20ms/step - loss: 2.8876
Epoch 49/50
14/14 [==============================] - 0s 21ms/step - loss: 2.8751
Epoch 50/50
14/14 [==============================] - 0s 20ms/step - loss: 2.8264
```

Code to generate new text

```
for i in range(500):
  randomVal = np.random.randint(0, len(charX)-1)
  randomStart = charX[randomVal]

  x = np.reshape(randomStart, (1, len(randomStart), 1))
  x = x/float(numberOfUniqueChars)
  pred = model.predict(x)
  index = np.argmax(pred)
  randomStart.append(index)
  randomStart = randomStart[1: len(randomStart)]
```

So our newly generated text is:

```
print("".join([idsForChars[value] for value in randomStart]))
```

```
hence;
which ever she could with haste dispatch,
she'd come again, and with a greedy ear
devour up th
```

✓ 0s completed at 10:34 PM ● ✕