## Develop a Small LSTM Recurrent Neural Network

Develop a simple LSTM network to learn sequences of characters from "Alice in Wonderland". We will use this model to generate new sequences of characters.

importing the classes and functions we intend to use to train our model.

In [24]:

```
import sys
import numpy
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import LSTM
from keras.callbacks import ModelCheckpoint
from keras.utils import np_utils
```

Next, we need to load the ASCII text for the book into memory and convert all of the characters to lowercase to reduce the vocabulary that the network must learn.

In [25]:

```
#load ascii text and convert to lowercase
filename="wonderland.txt"
raw_text=open(filename).read()
raw_text=raw_text.lower()
```

Above code loads the book. Now create Neural Network for data modeling. We cannot model the characters directly, so we map each character to unique_id(integers).

Create a set of all distinct characters and then map these characters to integer.

In [26]:

```
# create map of unique chars to integers
chars=sorted(list(set(raw_text)))
char_to_int=dict((c,i)for i,c in enumerate(chars))
#print(chars)
print(char_to_int)
```

```
{'\n': 0, ' ': 1, '!': 2, '(': 3, ')': 4, '*': 5, ',': 6, '-': 7, '.': 8, '0': 9, '3': 10, ':':
11, ';': 12, '?': 13, '[': 14, ']': 15, '_': 16, 'a': 17, 'b': 18, 'c': 19, 'd': 20, 'e': 21, 'f':
22, 'g': 23, 'h': 24, 'i': 25, 'j': 26, 'k': 27, 'l': 28, 'm': 29, 'n': 30, 'o': 31, 'p': 32, 'q':
33, 'r': 34, 's': 35, 't': 36, 'u': 37, 'v': 38, 'w': 39, 'x': 40, 'y': 41, 'z': 42, '‘': 43, '’':
44, '“': 45, '”': 46}
```

In [27]:

```
n_chars=len(raw_text)
n_vocab=len(chars)
print("Total Characters : ",n_chars)
print("Total Vocab : ",n_vocab)
```

```
Total Characters :  144412
Total Vocab :  47
```

In [28]:

```
# prepare the dataset of input to output pairs encoded as integers
seq_length=100
dataX=[]
dataY=[]
for i in range(0,n_chars-seq_length,1):
    seq_in=raw_text[i:i+seq_length]
```

```
    seq_out=raw_text[i+seq_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 144312
```

The code shows that we have training pattern to predict each of the remaining characters

In [29]:

```
#reshape X to be [samples, time steps , features]
X=numpy.reshape(dataX,(n_patterns,seq_length,1))
# normalize
X=X/float(n_vocab)
#encode the output variable
y=np_utils.to_categorical(dataY)
```

## Designing LSTM model

We are going to define LSTM model with single hidden layer with 256 memory units. The output layer is a Dense laer using the softmax activation function to output a probability prediction for each of the 47 characters between 0 and 1.

In [30]:

```
# define the LSTM model
model=Sequential()
model.add(LSTM(256,input_shape=(X.shape[1],X.shape[2])))
model.add(Dropout(0.2))
model.add(Dense(y.shape[1],activation='softmax'))
model.compile(loss='categorical_crossentropy',optimizer='adam')
```

There is no test dataset. We are modeling the entire training dataset to learn the probability of each character in a sequence.

In [17]:

```
# define the checkpoint
filepath="weights-improvement-{epoch:02d}-{loss:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='loss', verbose=1, save_best_only=True, mode='min')
callbacks_list = [checkpoint]
```

In [18]:

```
model.fit(X, y, epochs=20, batch_size=128, callbacks=callbacks_list)
```

```
Epoch 1/20
144312/144312 [==============================] - 685s 5ms/step - loss: 2.9985

Epoch 00001: loss improved from inf to 2.99847, saving model to weights-improvement-01-2.9985.hdf5
Epoch 2/20
144312/144312 [==============================] - 664s 5ms/step - loss: 2.7888

Epoch 00002: loss improved from 2.99847 to 2.78880, saving model to weights-improvement-02-2.7888.
hdf5
Epoch 3/20
144312/144312 [==============================] - 631s 4ms/step - loss: 2.6752

Epoch 00003: loss improved from 2.78880 to 2.67522, saving model to weights-improvement-03-2.6752.
hdf5
Epoch 4/20
144312/144312 [==============================] - 599s 4ms/step - loss: 2.6027

Epoch 00004: loss improved from 2.67522 to 2.60275, saving model to weights-improvement-04-2.6027.
hdf5
Epoch 5/20
144312/144312 [==============================] - 628s 4ms/step - loss: 2.5423
```

```
Epoch 00005: loss improved from 2.60275 to 2.54227, saving model to weights-improvement-05-2.5423.
hdf5
Epoch 6/20
144312/144312 [==============================] - 633s 4ms/step - loss: 2.4834

Epoch 00006: loss improved from 2.54227 to 2.48337, saving model to weights-improvement-06-2.4834.
hdf5
Epoch 7/20
144312/144312 [==============================] - 639s 4ms/step - loss: 2.4302

Epoch 00007: loss improved from 2.48337 to 2.43025, saving model to weights-improvement-07-2.4302.
hdf5
Epoch 8/20
144312/144312 [==============================] - 632s 4ms/step - loss: 2.3781

Epoch 00008: loss improved from 2.43025 to 2.37810, saving model to weights-improvement-08-2.3781.
hdf5
Epoch 9/20
144312/144312 [==============================] - 615s 4ms/step - loss: 2.3314

Epoch 00009: loss improved from 2.37810 to 2.33140, saving model to weights-improvement-09-2.3314.
hdf5
Epoch 10/20
144312/144312 [==============================] - 536s 4ms/step - loss: 2.2879

Epoch 00010: loss improved from 2.33140 to 2.28786, saving model to weights-improvement-10-2.2879.
hdf5
Epoch 11/20
144312/144312 [==============================] - 534s 4ms/step - loss: 2.2437

Epoch 00011: loss improved from 2.28786 to 2.24373, saving model to weights-improvement-11-2.2437.
hdf5
Epoch 12/20
144312/144312 [==============================] - 588s 4ms/step - loss: 2.2026

Epoch 00012: loss improved from 2.24373 to 2.20257, saving model to weights-improvement-12-2.2026.
hdf5
Epoch 13/20
144312/144312 [==============================] - 564s 4ms/step - loss: 2.1658

Epoch 00013: loss improved from 2.20257 to 2.16579, saving model to weights-improvement-13-2.1658.
hdf5
Epoch 14/20
144312/144312 [==============================] - 537s 4ms/step - loss: 2.1251

Epoch 00014: loss improved from 2.16579 to 2.12508, saving model to weights-improvement-14-2.1251.
hdf5
Epoch 15/20
144312/144312 [==============================] - 535s 4ms/step - loss: 2.0894

Epoch 00015: loss improved from 2.12508 to 2.08941, saving model to weights-improvement-15-2.0894.
hdf5
Epoch 16/20
144312/144312 [==============================] - 536s 4ms/step - loss: 2.0563

Epoch 00016: loss improved from 2.08941 to 2.05635, saving model to weights-improvement-16-2.0563.
hdf5
Epoch 17/20
144312/144312 [==============================] - 533s 4ms/step - loss: 2.0260

Epoch 00017: loss improved from 2.05635 to 2.02597, saving model to weights-improvement-17-2.0260.
hdf5
Epoch 18/20
144312/144312 [==============================] - 533s 4ms/step - loss: 1.9951

Epoch 00018: loss improved from 2.02597 to 1.99514, saving model to weights-improvement-18-1.9951.
hdf5
Epoch 19/20
144312/144312 [==============================] - 533s 4ms/step - loss: 1.9692

Epoch 00019: loss improved from 1.99514 to 1.96918, saving model to weights-improvement-19-1.9692.
hdf5
Epoch 20/20
144312/144312 [==============================] - 534s 4ms/step - loss: 1.9378

Epoch 00020: loss improved from 1.96918 to 1.93776, saving model to weights-improvement-20-1.9378.
hdf5
```

Out[18]:

```
<keras.callbacks.History at 0x27a88f0e0b8>
```

## Generating Text with an LSTM Network

Generating text using the trained LSTM network is relatively straightforward.

Firstly, we load the data and define the network in exactly the same way, except the network weights are loaded from a checkpoint file and the network does not need to be trained.

In [36]:

```
# load the network weights
filename="weights-improvement-20-1.9378.hdf5"
model.load_weights(filename)
model.compile(loss='categorical_crossentropy',optimizer='adam')
```

Creating reverse mapping for converting integers to character

In [37]:

```
int_to_char=dict((i,c)for i,c in enumerate(chars))
print("Reverse mapping :\n",int_to_char)
```

```
Reverse mapping :
 {0: '\n', 1: ' ', 2: '!', 3: '(', 4: ')', 5: '*', 6: ',', 7: '-', 8: '.', 9: '0', 10: '3', 11: ':
', 12: ';', 13: '?', 14: '[', 15: ']', 16: '_', 17: 'a', 18: 'b', 19: 'c', 20: 'd', 21: 'e', 22: '
f', 23: 'g', 24: 'h', 25: 'i', 26: 'j', 27: 'k', 28: 'l', 29: 'm', 30: 'n', 31: 'o', 32: 'p', 33:
'q', 34: 'r', 35: 's', 36: 't', 37: 'u', 38: 'v', 39: 'w', 40: 'x', 41: 'y', 42: 'z', 43: '‘', 44:
'’', 45: '“', 46: '”'}
```

Now we are going to use seed sequence as input, generate the next character then update the seed sequence to add the generated character on the end and trim of the first character. This process is repeated for as long as we want to predict new characters(e.g. a sequence of 1000 chars in len)

In [39]:

```
#pick a random seed
start=numpy.random.randint(0,len(dataX)-1)
pattern=dataX[start]
print("Seed")
print("\"",''.join([int_to_char[value] for value in pattern]),"\"")
#generate characters

for i in range(1000):
    x=numpy.reshape(pattern,(1,len(pattern),1))
    x=x/float(n_vocab)
    prediction=model.predict(x,verbose=0)
    index=numpy.argmax(prediction)
    result=int_to_char[index]
    seq_in=[int_to_char[value] for value in pattern]
   # print("\n\n\n\ Generated text: \n")
    sys.stdout.write(result)
    pattern.append(index)
    pattern=pattern[1:len(pattern)]
print("\nDone")
```

```
Seed
" the others looked round also, and
all of them bowed low.

‘would you tell me,’ said alice, a little  "
semiiee, ‘in you dnn the mort brtiors ’huh the gortt ’ou tnonk!’ she said to herself, ‘io was a li
ttle so the tooe tf the was shinking the had soe the was oo the thieg aearge, and she tein soe toi
ne an the cade  ‘ht was a little oo the tian whth al c lo ent teaee teat i saou to the thet t thou
g the gurhhos weie the sas of the goose  she had not the when tae if the had sote the was soinki
she was soine on the bank, and she tai oo the whrte the was aoo ohe thee, and she toine whet she w
```

```
as ani goren thet sae if the gorrt wo the korke taate
the was sorilg and the toeds wfse tuiet in a loetter to ceoen ti the white rabbit. and the wordd h
ad no toem a art oe thet would be armlers  she houst whin she was an the cate tai in a goeat hurry
.
'the mort mite the more 'f think,' said the monke,

'ie horst shin t ali the duchess and bli the tai if then iad sae the was shing the had soee the wa
s oo the thilg  she was soinkigng to the toeee of the table, and she thing the was soinking at the
could s
Done
```

# ---------------------end-------------------

# Some Notes

## Larger LSTM Recurrent Neural Network

We can improve the results or the quality of the generated text by creating a much larger network.

---code--- -Begins--

model = Sequential() model.add(LSTM(256, input_shape=(X.shape[1], X.shape[2]), return_sequences=True)) model.add(Dropout(0.2)) model.add(LSTM(256)) model.add(Dropout(0.2)) model.add(Dense(y.shape[1], activation='softmax')) model.compile(loss='categorical_crossentropy', optimizer='adam')

---end---

We will also need to retrain the model and change the file path name.

Training of larger Lstm network will require more resources. Finally, we will increase the number of training epochs from 20 to 50 and decrease the batch size from 128 to 64 to give the network more of an opportunity to be updated and learn.