Text Generation

Using LSTM Model

{ Long short term memory network : a type of Recurrent Neural Network }

Created by :

Aditi Kumar – 706/IT/14

Arushi Bhatt- 718/IT/14

Dhruv Kameshwar-730/IT/14

ABSTRACT

Neural networks which possess the capability to learn by themselves without any interference was first introduced in 1950's since then have advanced a lot with coming of single layer feed forward to multilayer feed forward then recurrent neural networks and so on . Here we have made use of the LSTM (Long short term memory network), which is a type of RNN and also possess the capability of remembering the information for a longer duration of time .

Recurrent neural networks can also be used as generative models.This means that in addition to being used for predictive models (making predictions) they can learn the sequences of a problem and then generate entirely new plausible sequences for the problem domain. Generative models like this are useful not only to study how well a model has learned a problem, but to learn more about the problem domain itself.

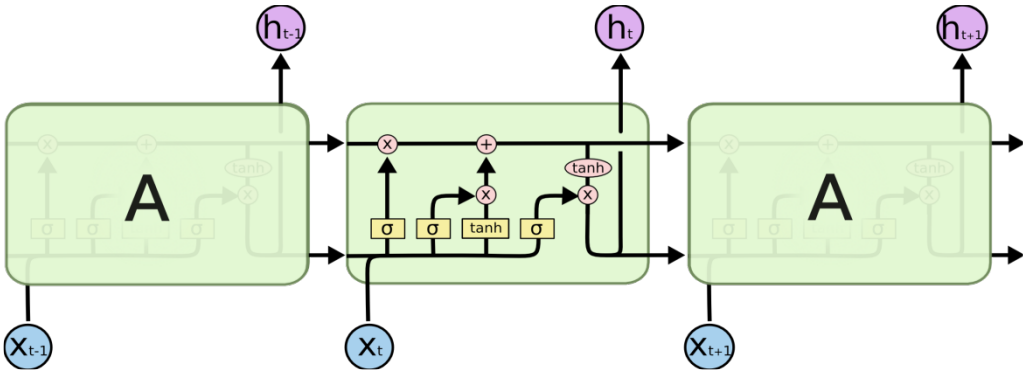
In this project we discovered how to create a generative model for text, character-by-character using LSTM recurrent neural networks in Python with Keras.

THEORY :

A recurrent neural network (RNN) is a class of artificial neural network where connections between units form a directed cycle. This allows it to exhibit dynamic temporal behavior. Unlike feedforward neural networks, RNNs can use their internal memory to process arbitrary sequences of inputs. This makes them applicable to tasks such as unsegmented, connectedhandwriting recognition or speech recognition.

Long short-term memory (LSTM) is a recurrent neural network (RNN) architecture that remembers values over arbitrary intervals. Stored values are not modified as learning proceeds[further explanation needed][citation needed]. RNNs[clarification needed]allow forward and backward connections between neurons.

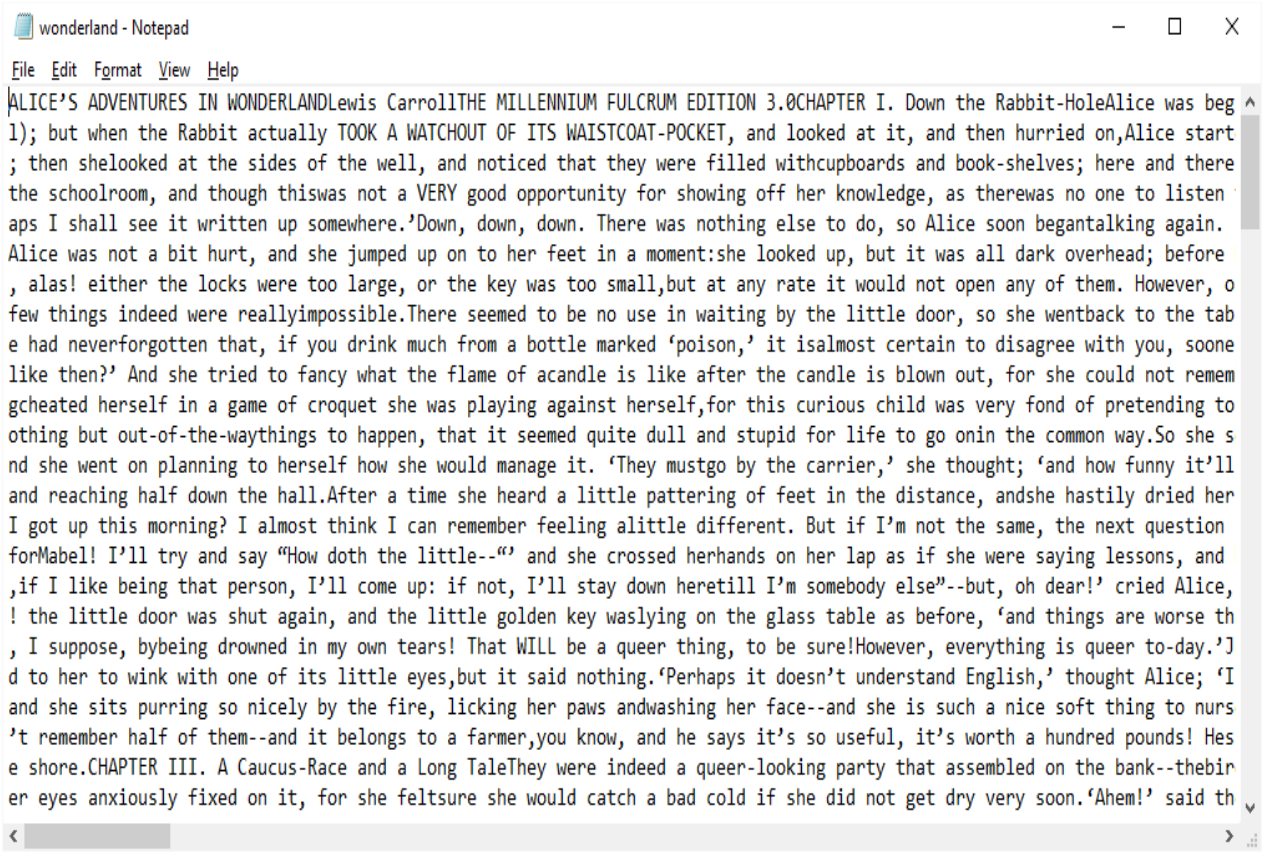
An LSTM is well-suited to classify, process and predict time series given time lags of unknown size and duration between important events. Relative insensitivity to gap length gives an advantage to LSTM over alternative RNNs[examples needed], hidden Markov models and other sequence learning methods in numerous applications.



The repeating module in an LSTM contains four interacting layers.

INPUT :

- Dataset is the whole novel of 'Alice's Adventures in Wonderland by Lewis Carroll'.
- Dataset was downloaded as complete text in ASCII format (Plain Text UTF-8) placed it in your working directory with the filename **wonderland.txt**.



OVERVIEW OF PROCESS :

We have learnt the dependencies between characters and the conditional probabilities of characters in sequences so that we can in turn generate wholly new and original sequences of characters.

We have used , Keras which is an open source neural network library written in Python. It is capable of running on top of MXNet, Deeplearning4j, Tensorflow, CNTK or Theano

CODE:

```
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
# load ascii text and covert to lowercase
filename = "wonderlandcopy.txt"
raw_text = open(filename).read()
raw_text = raw_text.lower()
# create mapping of unique chars to integers
chars = sorted(list(set(raw_text)))
char_to_int = dict((c, i) for i, c in enumerate(chars))
n_chars = len(raw_text)
n_vocab = len(chars)
print ("Total Characters:", n_chars)
print ("Total Vocab: ",n_vocab)
# 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)
# reshape X to be [samples, time steps, features]
X = numpy.reshape(dataX, (n_patterns, seq_length, 1))
# normalize
X = X / float(n_vocab)
# one hot encode the output variable
y = np_utils.to_categorical(dataY)
# define the LSTM model
model = Sequential()
model.add(LSTM(150, 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')
# 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]
model.fit(X, y, epochs=20, batch_size=128, callbacks=callbacks_list)
# load the network weights
filename = "weights-improvement-14-2.1243.hdf5"
```

```
model.load_weights(filename)
model.compile(loss='categorical_crossentropy', optimizer='adam')
# 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.ndarray.argmax(prediction)
  result = int_to_char[index]
  print ("new result",result)
  seq_in = [int_to_char[value] for value in pattern]
  sys.stdout.write(result)
  pattern.append(index)
  pattern = pattern[1:len(pattern)]
print ("\nDone.")
```

OUTPUTS:

Error Obtained from the 15th Epoch : 2.1243

```
Using TensorFlow backend.

Total Characters: 11440
Total Vocab:  45
Total Patterns:  11340
Epoch 1/20
11264/11340 [============================>.] - ETA: 0s - loss: 3.1293Epoch 00000: loss improved from inf to 3.1285
3, saving model to weights-improvement-00-3.1285.hdf5
11340/11340 [==============================] - 65s - loss: 3.1285
Epoch 2/20
11264/11340 [============================>.] - ETA: 0s - loss: 3.0327Epoch 00001: loss improved from 3.12853 to 3.0
3235, saving model to weights-improvement-01-3.0324.hdf5
11340/11340 [==============================] - 63s - loss: 3.0324
Epoch 3/20
11264/11340 [============================>.] - ETA: 0s - loss: 3.0160Epoch 00002: loss improved from 3.03235 to 3.0
1704, saving model to weights-improvement-02-3.0170.hdf5
11340/11340 [==============================] - 65s - loss: 3.0170
Epoch 4/20
11264/11340 [============================>.] - ETA: 0s - loss: 3.0170Epoch 00003: loss improved from 3.01704 to 3.0
1612, saving model to weights-improvement-03-3.0161.hdf5
11340/11340 [==============================] - 68s - loss: 3.0161
Epoch 5/20
11264/11340 [============================>.] - ETA: 0s - loss: 3.0115Epoch 00004: loss improved from 3.01612 to 3.0
1150, saving model to weights-improvement-04-3.0115.hdf5
11340/11340 [==============================] - 66s - loss: 3.0115
Epoch 6/20
11264/11340 [============================>.] - ETA: 0s - loss: 3.0056Epoch 00005: loss improved from 3.01150 to 3.0
0563, saving model to weights-improvement-05-3.0056.hdf5
11340/11340 [==============================] - 68s - loss: 3.0056
Epoch 7/20
11264/11340 [============================>.] - ETA: 0s - loss: 3.0003Epoch 00006: loss improved from 3.00563 to 3.0
0050, saving model to weights-improvement-06-3.0005.hdf5
```

```
Terminal  File  Edit  View  Search  Terminal  Help                (0:42, 82%)  Mon Oct 30 2017 11:40

In [4]: # 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):
   ...: ^Ix = numpy.reshape(pattern, (1, len(pattern), 1))
   ...: ^Ix = x / float(n_vocab)
   ...: ^Iprediction = model.predict(x, verbose=0)
   ...: ^Iindex = numpy.argmax(prediction)
   ...: ^Iresult = int_to_char[index]
   ...: ^Iseq_in = [int_to_char[value] for value in pattern]
   ...: ^Isys.stdout.write(result)
   ...: ^Ipattern.append(index)
   ...: ^Ipattern = pattern[1:len(pattern)]
   ...: print ("\nDone.")
   ...:
Seed:
" are your shoes done with?' said the gryphon. 'i mean, what
makes them so shiny?'

alice looked down  "

an the pabbit sase the waite oo the garte oa the botr and the cadl. and the was aolng the woile she whit sae in the coulo,
'hh yhu a latg toine i sas a little 'iite ' said the daterpillar.

'ie cou dot yhu,' said the kacte haree
tai ionttenning to the tabbit  and whit she was to torl to the tooe  and whst hlr lasde whtt the rooer oh the tooee
'hh you dan to toink i sas a datter an i saa -tte toine i sas t yhul toe 'orld 'hu, and the mort oaad to the toiee. and the wait to the woile  ald whs
t hlr looten the woole
'hh you dan to tan i toile 'hu, a dan and go whe aad the wai int in the garter, and the was aolng the woile she whit she was to the whrt she was oo th
e tooee, and she taid the was oo the tooe, and she taid the was oo the tooe, and she taid the was oo the gorr, and she tai iot io whet the was to the
woole
'ho wou dan to tay 'hu, and then io the darter
she harter was toe tiet tored an io was in the carter, and she was aolng the woile she whit she was to the whrt she white wa
Done.
```

REFERENCES :

1. SEPTEMBER 17, 2015 BY DENNY BRITZ
Recurrent Neural Networks Tutorial, Part 1 – Introduction to RNNs
http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

2. This blog is important : Understanding LSTM Networks :
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

3. The Unreasonable Effectiveness of Recurrent Neural Networks
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

4. Videos : Recurrent Neural Networks (RNN / LSTM)with Keras - Python
https://www.youtube.com/watch?v=4rG8IsKdC3U

5. KERAS Documentation

https://keras.io/