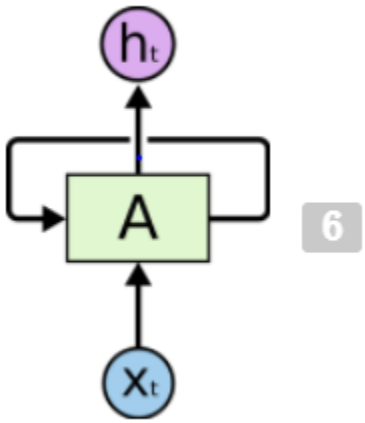
Text Generation with LSTM Recurrent Neural Networks

1. Abstract

In this paper, a sample text file is used as an input for which a machine will be trained. Based on the training, machine will automatically execute its own generated words. We explore the ability of Long short term memory (LSTM) recurrent neural network architecture as compared to normal Recurrent Neural Network. We find out that recurrent neural networks can be used as generative models. This makes a provision that they can be used for predictive models & to learn the sequences of a problem and generate new solutions as well. We discover how to create a generative model for text, character by character using Long Short Term recurrent neural networks with Keras and Tensorflow as backend.

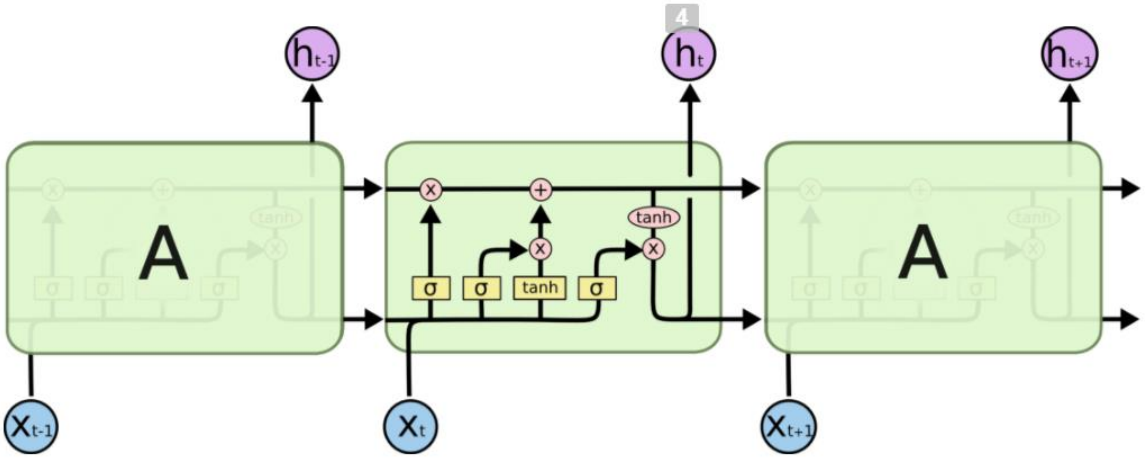
2. Introduction

Recurrent Neural Networks; the working of recurrent neural networks is like the working of a human brain. As a human brain processes and understands based on the understanding of words developed over time, neural networks try to do the same. However, the traditional neural networks are not capable of doing this. For example, imagine you want to guess what is going to happen next in a book, what is going through a writer's mind. We cannot use traditional neural networks for reasoning about previous events that have happened in the book so far. This problem is solved by Recurrent Neural Networks. They form a loop within the network which allows to store the information.



In the above diagram, a chunk of neural network, A, looks at some input xt and outputs a value ht. A loop allows information to be passed from one step of the network to the next. However, the biggest drawback of Recurrent Neural Networks is long term dependencies. In theory, RNN are capable of handling long term dependencies but they fail to understand when to use all the past data and when to use just the previous data. This problem is solved by LSTM.

LSTMs; LSTM stand for Long Short Term Memory. They are special kind of Recurrent Neural Networks which are specifically designed to handle long term dependencies. They remember information for longer periods of time by default. It is not something they have to explicitly learn or train for. LSTM, like all RNNs have a chain like structure of repeating modules of neural network. LSTM has a different chain like structure as compared to traditional RNN. They have four neural network layers instead of one.



3. Code

We import the book which is in .txt format. We need to specify the encoding as it is required to make the file compatible in Python 3.X .Then we convert the .txt to all lower case. Converting to lower case will reduce the number of words the model will have to learn. We further plan to use the .txt file as it is to train the model with better accuracy.

```
filename = "wonderland.txt"
book = open(filename,encoding="utf8").read() # Need to specify
the encoding in python 3+
book = book.lower()
```

We then divide the book in characters of length 100. In this way, the model will learn the next character after the 100 characters at a time and find out different patterns in the book. The more number of patterns that are discovered, better is the training set and will yield better results. For example, consider sequence length of 5 and word 'amazon'. It will first read 'amazo' and read 'n' after that. Consider a word greater than length 6. For example, 'wonderland' The reading for this word will be as follows:

1. 'wonde' & 'r'
2. 'onder' & 'l'
3. 'nderl' & 'a'

... and so on.

```
seq_length = 100
dataChar = []
dataInt = []
for i in range(0, no_of_chars - seq_length, 1):
    seq_in = book[i:i + seq_length]
    seq_out = book[i + seq_length]
    dataChar.append([char_to_int[char] for char in seq_in])
    dataInt.append(char_to_int[seq_out])
    no_of_patterns = len(dataChar)
print("Total Patterns: ", no_of_patterns)
```

Now that we have prepared our training data we need to transform it so that it is suitable for use with Keras. First we must transform the list of input sequences into the form expected by an LSTM network. Next we need to rescale the integers to the range 0-to-1 to make the patterns easier to learn by the LSTM network that uses the sigmoid activation function by default. LSTM is a modification of Recurrent Neural Network. Its performance is better than RNN because RNN is not capable of memorizing large amount of data. LSTM includes memory cell which can help maintain information in memory for longer amount of time.

```
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')
```

We fit the model and run numerous epochs with as small batch size as possible as it will train the model even better. After extensively training the model we pick a random seed and generate characters in a way mentioned above. We append a single index value to the sequence and remove the extra index if it does not suit well.

4. Results

After parsing the input text file, the result obtained is a machine generated text which is of length provided in the range method. The model.predict() function does the testing of the pattern which the machine learned. For providing the correct results, the result is converted again to characters. sys.stdout.write() is written for better alignment of texts as it generates a paragraph like structure which is easy to read.

```
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]
    sys.stdout.write(result)
    pattern.append(index) # To append a single remaining index
    pattern = pattern[1:len(pattern)] # To remove an extra index
print("\nDone.")
```

For better results and greater accuracy, we provide decreased amount of batch size and increase in the number of epochs. It will take a tremendous amount of time, in hours but it also lowers the value of the loss function and generates human readable sentences.

5. Discussion

Even after necessary alterations of the parameters, we will see that generally there are fewer spelling mistakes and the text looks more realistic. The machine has accurately showcased the sentence structure which is human readable and most of the words are making sense such as ‘little’, ‘herself’, ‘alice’, etc. Some of the words seems a bit nonsensical. For example, the same phrases get repeated. Quotes are opened but not closed. Irregularities in the word structure. These are better results but there is still a lot of room for improvement.

```
and the whrt on thet wise she wes oo the thet had bed an a leeuter wote. and the was nor in anliher wored in a little worde. an
d she taid to herself ‘iht sae a citters tas a aonser tf that io the sabe. and the west on tuiee to an and then at the codless,
and she was norking the thiet oaed th the was to gend her haad.
‘he io wonh th the wout!’ she kact rhidpedd in a tory of tiice of the gormh.
‘ht is sie suree ihtee bn a les lis wo teye you mo youh tees the sea--’

‘what woul toeete,’ said the match hare.

‘ih toete thee io the wes,’ said alice, ‘io would be a very oo tuee an all an it eane  and i saad io so teet to teee ’o

al cel toe white sited the dooreuse the dad not thee to tee otoersins oo the that was a lott of linel to bn the wint wire an in
hade an herse fer and taed to the thett whine whs dad boen the shing then soeetien, and the woide the was not in a lintte or tw
o she was notking an it was tore tie was thet hard sas she winle rai her hand and a lroked oo the tioed the was
Done.
```

We can improve our machine by doing either or all the following steps:

Training the machine for the chunk of sentences rather than random sequence of characters.

Increasing the number of epochs and decreasing the number of batch input size for training the text file.

Create a dropout for the visible input layer to provide error redundancy over the neural network by tuning the dropout percentage.

Increasing the size of the network layers to provide more memory utilization.

6. Reference

Bastien Moysset, Christopher Kermorvant, Christian Wolf, “Paragraph text segmentation into lines with Recurrent Neural Networks”, Document Analysis and Recognition (ICDAR), 2015 13th International Conference, 2015.

http://colah.github.io/posts/2015-08-Understanding-LSTMs

https://www.datacamp.com/community/tutorials/deep-learning-python

https://github.com/tensorflow/models

https://chunml.github.io/ChunML.github.io/project/Creating-Text-Generator-Using-Recurrent-Neural-Network/ (Author: Trung Tran. Licensed under MIT)

https://en.wikipedia.org/wiki/Long_short-term_memory

https://deeplearning4j.org/lstm.html