×

-

(http://play.google.com/store/apps/details?id=com.analyticsvidhya.android)

(https://www.facebook.com/AnalyticsVidhya) (http

(https://plus.google.com/+Analyticsvidhya/posts) (https://in.linkedin.c

MURALI94 (HTTPS://ID.AN

HOME (HTTPS://WWW.ANALYTICSVIDHYA.COM) BLOG ARCHIVE (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG-ARCHIVE/)

DISCUSS (HTTPS://DISCUSS.ANALYTICSVIDHYA.COM) CORPORATE (HTTPS://WWW.ANALYTICSVIDHYA.COM/CORPORATE/)

(https://www.analyticsvidhya.com/myfeed/?utm-source=blog&utm-medium=top-icon/)

(https://dsat.analyticsvidhya.com/?utm_source=blog&utm_

Home (https://www.analyticsvidhya.com/) › Advanced (https://www.analyticsvidhya.com/blog/category/advanced/) › Essentials of Deep Learning : Introduction to Long Short Term Memory (https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/)

ADVANCED (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/ADVANCED/)
ALGORITHM (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/ALGORITHM/)
DEEP LEARNING (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/DEEP-LEARNING/)
NLP (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/NLP/)
PYTHON (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/PYTHON-2/)
SEQUENCE MODELING (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/SEQUENCE-MODELING/)
SUPERVISED (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/SUPERVISED/)

Essentials of Deep Learning : Introduction to Long Short Term Memory

PRANJAL SRIVASTAVA (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/AUTHOR/PRANJ52/), DECEMBER 10, 2017

Introduction

Sequence prediction problems have been around for a long time. They are considered as one of the hardest problems to solve in the data science industry. These include a wide range of problems; from predicting sales to finding patterns in stock markets' data, from understanding movie plots to recognizing your way of speech, from language translations to predicting your next word on your iPhone's keyboard.

With the recent breakthroughs that have been happening in data science, it is found that for almost all of these sequence prediction problems, Long short Term Memory networks, a.k.a LSTMs have been observed as the most effective solution.

LSTMs have an edge over conventional feed-forward neural networks and RNN in many ways. This is because of their property of selectively remembering patterns for long durations of time. The purpose of this article is to explain LSTM and enable you to use it in real life problems. Let's have a look!

Note: To go through the article, you must have basic knowledge of neural networks and how Keras (a deep learning library) works. You can refer the mentioned articles to understand these concepts:

- Understanding Neural Network From Scratch (https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/)
- Fundamentals of Deep Learning – Introduction to Recurrent Neural Networks (https://www.analyticsvidhya.com/blog/2017/12/introduction-to-recurrent-neural-networks/)
- Tutorial: Optimizing Neural Networks using Keras (with Image recognition case study) (https://www.analyticsvidhya.com/blog/2016/10/tutorial-optimizing-neural-networks-using-keras-with-image-recognition-case-study/)

Table of Contents

1. Flashback: A look into Recurrent Neural Networks (RNN)

2. Limitations of RNNs
3. Improvement over RNN : Long Short Term Memory (LSTM)
4. Architecture of LSTM
    1. Forget Gate
    2. Input Gate
    3. Output Gate

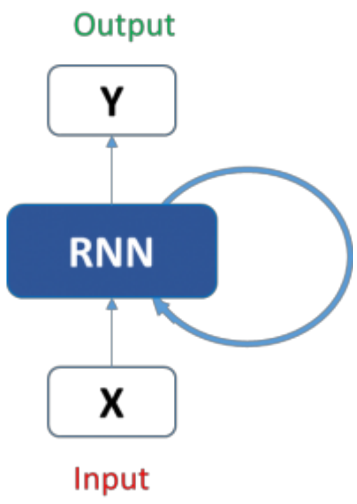
5. Text generation using LSTMs.

1. Flashback: A look into Recurrent Neural Networks (RNN)

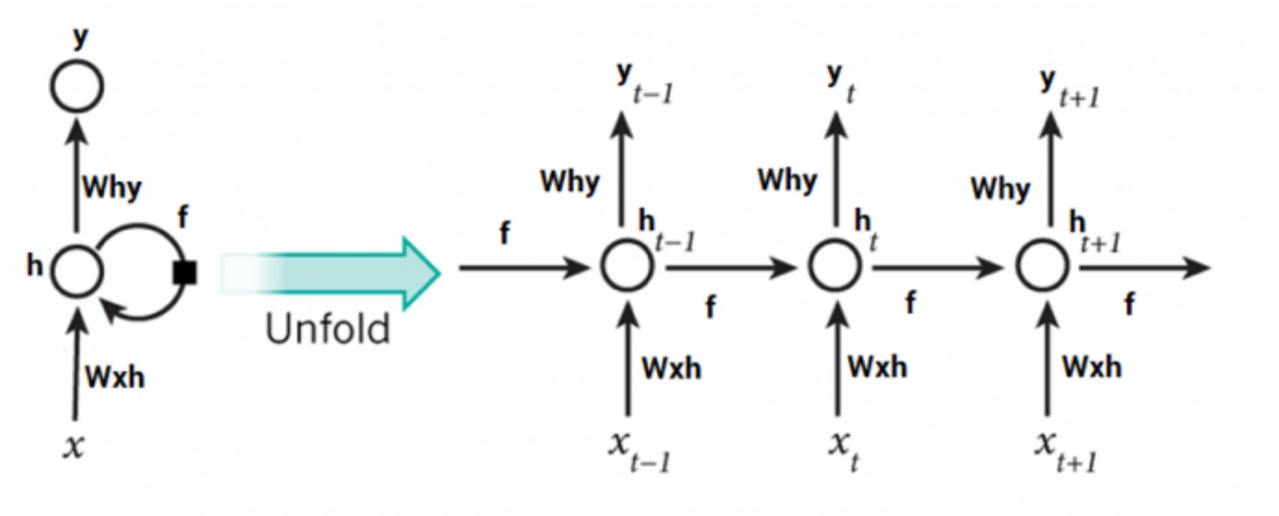
Take an example of sequential data, which can be the stock market's data for a particular stock. A simple machine learning model or an Artificial Neural Network may learn to predict the stock prices based on a number of features: the volume of the stock, the opening value etc. While the price of the stock depends on these features, it is also largely dependent on the stock values in the previous days. In fact for a trader, these values in the previous days (or the trend) is one major deciding factor for predictions.

In the conventional feed-forward neural networks, all test cases are considered to be independent. That is when fitting the model for a particular day, there is no consideration for the stock prices on the previous days.

This dependency on time is achieved via Recurrent Neural Networks. A typical RNN looks like:



This may be intimidating at first sight, but once unfolded, it looks a lot simpler:



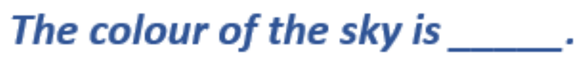
Now it is easier for us to visualize how these networks are considering the trend of stock prices, before predicting the stock prices for today. Here every prediction at time t (h_t) is dependent on all previous predictions and the information learned from them.

RNNs can solve our purpose of sequence handling to a great extent but not entirely. We want our computers to be good enough to write Shakespearean sonnets (https://www.psfk.com/2014/01/shakespeare-machine-learning-poetry-app.html). Now RNNs are great when it comes to short contexts, but in order to be able to build a story and remember it, we need our models to be able to understand and remember the context behind the sequences, just like a human brain. This is not possible with a simple RNN.

Why? Let's have a look.

2. Limitations of RNNs

Recurrent Neural Networks work just fine when we are dealing with short-term dependencies. That is when applied to problems like:



The colour of the sky is ______.

RNNs turn out to be quite effective. This is because this problem has nothing to do with the context of the statement. The RNN need not remember what was said before this, or what was its meaning, all they need to know is that in most cases the sky is blue. Thus the prediction would be:

The colour of the sky is blue.

However, vanilla RNNs fail to understand the context behind an input. Something that was said long before, cannot be recalled when making predictions in the present. Let's understand this as an example:

I spent 20 long years working for the under-privileged kids in Spain. I then moved to Africa.

……….

I can speak fluent ________.

Here, we can understand that since the author has worked in Spain for 20 years, it is very likely that he may possess a good command over Spanish. But, to make a proper prediction, the RNN needs to remember this context. The relevant information may be separated from the point where it is needed, by a huge load of irrelevant data. This is where a Recurrent Neural Network fails!

The reason behind this is the problem of **Vanishing Gradient.** In order to understand this, you'll need to have some knowledge about how a feed-forward neural network learns. We know that for a conventional feed-forward neural network, the weight updating that is applied on a particular layer is a multiple of the learning rate, the error term from the previous layer and the input to that layer. Thus, the error term for a particular layer is somewhere a product of all previous layers' errors. When dealing with activation functions like the sigmoid function, the small values of its derivatives (occurring in the error function) gets multiplied multiple times as we move towards the starting layers. As a result of this, the gradient almost vanishes as we move towards the starting layers, and it becomes difficult to train these layers.

A similar case is observed in Recurrent Neural Networks. RNN remembers things for just small durations of time, i.e. if we need the information after a small time it may be reproducible, but once a lot of words are fed in, this information gets lost somewhere. This issue can be resolved by applying a slightly tweaked version of RNNs – the Long Short-Term Memory Networks.

3. Improvement over RNN: LSTM (Long Short-Term Memory) Networks

When we arrange our calendar for the day, we prioritize our appointments right? If in case we need to make some space for anything important we know which meeting could be canceled to accommodate a possible meeting.

Turns out that an RNN doesn't do so. In order to add a new information, it transforms the existing information completely by applying a function. Because of this, the entire information is modified, on the whole, i. e. there is no consideration for *'important'* information and *'not so important'* information.

LSTMs on the other hand, make small modifications to the information by multiplications and additions. With LSTMs, the information flows through a mechanism known as cell states. This way, LSTMs can selectively remember or forget things. The information at a particular cell state has three different dependencies.

We'll visualize this with an example. Let's take the example of predicting stock prices for a particular stock. The stock price of today will depend upon:

1. The trend that the stock has been following in the previous days, maybe a downtrend or an uptrend.
2. The price of the stock on the previous day, because many traders compare the stock's previous day price before buying it.
3. The factors that can affect the price of the stock for today. This can be a new company policy that is being criticized widely, or a drop in the company's profit, or maybe an unexpected change in the senior leadership of the company.

These dependencies can be generalized to any problem as:

1. The previous cell state *(i.e. the information that was present in the memory after the previous time step)*
2. The previous hidden state *(i.e. this is the same as the output of the previous cell)*
3. The input at the current time step *(i.e. the new information that is being fed in at that moment)*

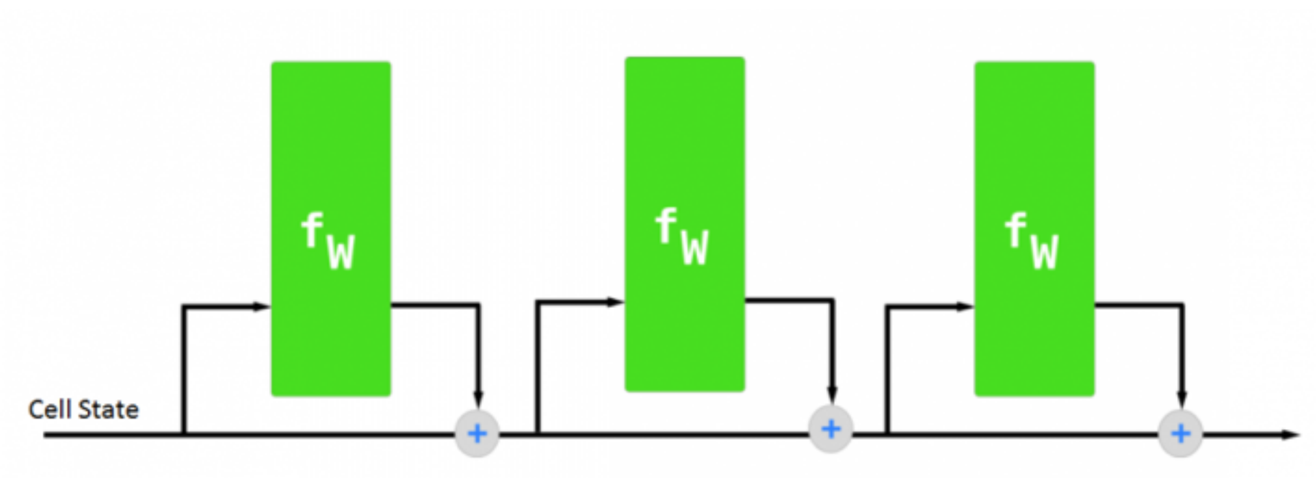
Another important feature of LSTM is its analogy with conveyor belts!

That's right!

Industries use them to move products around for different processes. LSTMs use this mechanism to move information around.

We may have some addition, modification or removal of information as it flows through the different layers, just like a product may be molded, painted or packed while it is on a conveyor belt.

The following diagram explains the close relationship of LSTMs and conveyor belts.



Source (https://ayearofai.com/rohan-lenny-3-recurrent-neural-networks-10300100899b)

Although this diagram is not even close to the actual architecture of an LSTM, it solves our purpose for now.

Just because of this property of LSTMs, where they do not manipulate the entire information but rather modify them slightly, they are able to *forget* and *remember* things selectively. How do they do so, is what we are going to learn in the next section?

4. Architecture of LSTMs

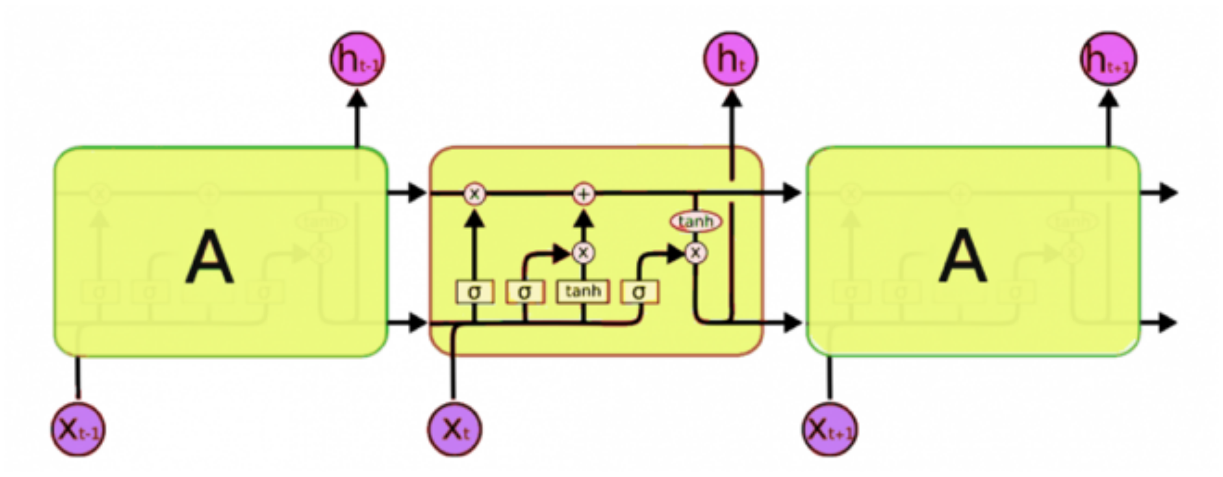
The functioning of LSTM can be visualized by understanding the functioning of a news channel's team covering a murder story. Now, a news story is built around facts, evidence and statements of many people. Whenever a new event occurs you take either of the three steps.

Let's say, we were assuming that the murder was done by 'poisoning' the victim, but the autopsy report that just came in said that the cause of death was 'an impact on the head'. Being a part of this news team what do you do? You immediately **forget** the previous cause of death and all stories that were woven around this fact.

What, if an entirely new suspect is introduced into the picture. A person who had grudges with the victim and could be the murderer? You **input** this information into your news feed, right?

Now all these broken pieces of information cannot be served on mainstream media. So, after a certain time interval, you need to summarize this information and **output** the relevant things to your audience. Maybe in the form of "*XYZ turns out to be the prime suspect.*".

Now let's get into the details of the architecture of LSTM network:



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

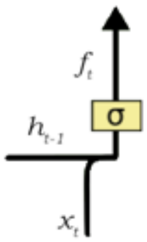
Now, this is nowhere close to the simplified version which we saw before, but let me walk you through it. A typical LSTM network is comprised of different memory blocks called **cells** (the rectangles that we see in the image). There are two states that are being transferred to the next cell; the **cell state** and the **hidden state**. The memory blocks are responsible for remembering things and manipulations to this memory is done through three major mechanisms, called **gates.** Each of them is being discussed below.

4.1 Forget Gate

Taking the example of a text prediction problem. Let's assume an LSTM is fed in, the following sentence:

Bob is a nice person. Dan on the other hand is evil.

As soon as the first full stop after "*person"* is encountered, the forget gate realizes that there may be a change of context in the next sentence. As a result of this, the *subject* of the sentence is *forgotten* and the place for the subject is vacated. And when we start speaking about "*Dan"* this position of the subject is allocated to "*Dan*". This process of forgetting the subject is brought about by the forget gate.



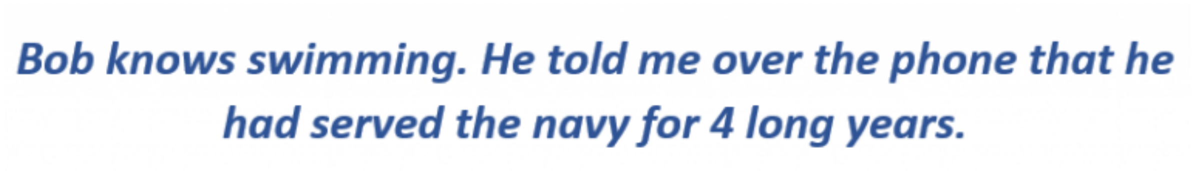
A forget gate is responsible for removing information from the cell state. The information that is no longer required for the LSTM to understand things or the information that is of less importance is removed via multiplication of a filter. This is required for optimizing the performance of the LSTM network.

This gate takes in two inputs; h_t-1 and x_t.

h_t-1 is the hidden state from the previous cell or the output of the previous cell and x_t is the input at that particular time step. The given inputs are multiplied by the weight matrices and a bias is added. Following this, the sigmoid function is applied to this value. The sigmoid function outputs a vector, with values ranging from 0 to 1, corresponding to each number in the cell state. Basically, the sigmoid function is responsible for deciding which values to keep and which to discard. If a '0' is output for a particular value in the cell state, it means that the forget gate wants the cell state to forget that piece of information completely. Similarly, a '1' means that the forget gate wants to remember that entire piece of information. This vector output from the sigmoid function is multiplied to the cell state.

4.2 Input Gate

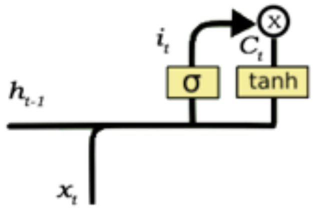
Okay, let's take another example where the LSTM is analyzing a sentence:



Bob knows swimming. He told me over the phone that he had served the navy for 4 long years.

Now the important information here is that “Bob” knows swimming and that he has served the Navy for four years. This can be added to the cell state, however, the fact that he told all this over the phone is a less important fact and can be ignored. This process of adding some new information can be done via the **input** gate.

Here is its structure:



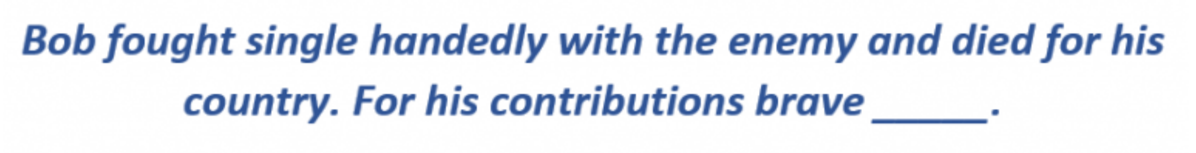
The input gate is responsible for the addition of information to the cell state. This addition of information is basically three-step process as seen from the diagram above.

1. Regulating what values need to be added to the cell state by involving a sigmoid function. This is basically very similar to the forget gate and acts as a filter for all the information from h_t-1 and x_t.
2. Creating a vector containing all possible values that can be added (as perceived from h_t-1 and x_t) to the cell state. This is done using the **tanh** function, which outputs values from -1 to +1.
3. Multiplying the value of the regulatory filter (the sigmoid gate) to the created vector (the tanh function) and then adding this useful information to the cell state via addition operation.

Once this three-step process is done with, we ensure that only that information is added to the cell state that is *important* and is not *redundant.*

4.3 Output Gate

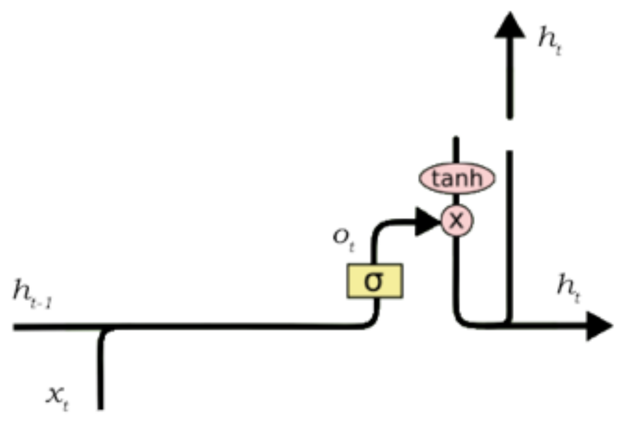
Not all information that runs along the cell state, is fit for being output at a certain time. We’ll visualize this with an example:



Bob fought single handedly with the enemy and died for his country. For his contributions brave ______.

In this phrase, there could be a number of options for the empty space. But we know that the current input of *'brave',* is an adjective that is used to describe a noun. Thus, whatever word follows, has a strong tendency of being a noun. And thus, Bob could be an apt output.

This job of selecting useful information from the current cell state and showing it out as an output is done via the output gate. Here is its structure:



The functioning of an output gate can again be broken down to three steps:

1. Creating a vector after applying **tanh** function to the cell state, thereby scaling the values to the range -1 to +1.
2. Making a filter using the values of h_t-1 and x_t, such that it can regulate the values that need to be output from the vector created above. This filter again employs a sigmoid function.
3. Multiplying the value of this regulatory filter to the vector created in step 1, and sending it out as a output and also to the hidden state of the next cell.

The filter in the above example will make sure that it diminishes all other values but *'Bob'.* Thus the filter needs to be built on the input and hidden state values and be applied on the cell state vector.

5. Text generation using LSTMs

We have had enough of theoretical concepts and functioning of LSTMs. Now we would be trying to build a model that can predict some *n* number of characters after the original text of Macbeth. Most of the classical texts are no longer protected under copyright and can be found here. (https://www.gutenberg.org/) An updated version of the .txt file can be found here (https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2017/12/10165151/macbeth.txt).

We will use the library **Keras,** which is a high-level API for neural networks and works on top of TensorFlow or Theano. So make sure that before diving into this code you have **Keras** installed and functional.

Okay, so let’s generate some text!

- **Importing dependencies**

```
# Importing dependencies numpy and keras
import numpy
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
```

We import all the required dependencies and this is pretty much self-explanatory.

- **Loading text file and creating character to integer mappings**

```
# load text
filename = "/macbeth.txt"

text = (open(filename).read()).lower()

# mapping characters with integers
unique_chars = sorted(list(set(text)))

char_to_int = {}
int_to_char = {}

for i, c in enumerate (unique_chars):
    char_to_int.update({c: i})
    int_to_char.update({i: c})
```

The text file is open, and all characters are converted to lowercase letters. In order to facilitate the following steps, we would be mapping each character to a respective number. This is done to make the computation part of the LSTM easier.

- **Preparing dataset**

```
# preparing input and output dataset
X = []
Y = []

for i in range(0, len(text) - 50, 1):
    sequence = text[i:i + 50]
    label =text[i + 50]
    X.append([char_to_int[char] for char in sequence])
    Y.append(char_to_int[label])
```

Data is prepared in a format such that if we want the LSTM to predict the *'O'* in *'HELLO'* we would feed in ['H', 'E' , 'L ' , 'L'] as the input and ['O'] as the expected output. Similarly, here we fix the length of the sequence that we want (set to 50 in the example) and then save the encodings of the first 49 characters in X and the expected output i.e. the 50th character in Y.

- **Reshaping of X**

```
# reshaping, normalizing and one hot encoding
X_modified = numpy.reshape(X, (len(X), 50, 1))
X_modified = X_modified / float(len(unique_chars))
Y_modified = np_utils.to_categorical(Y)
```

A LSTM network expects the input to be in the form [samples, time steps, features] where samples is the number of data points we have, time steps is the number of time-dependent steps that are there in a single data point, features refers to the number of variables we have for the corresponding true value in Y. We then scale the values in X_modified between 0 to 1 and one hot encode our true values in Y_modified.

- **Defining the LSTM model**

```
# defining the LSTM model
model = Sequential()
model.add(LSTM(300, input_shape=(X_modified.shape[1], X_modified.shape[2]),
return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(300))
model.add(Dropout(0.2))
model.add(Dense(Y_modified.shape[1], activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam')
```

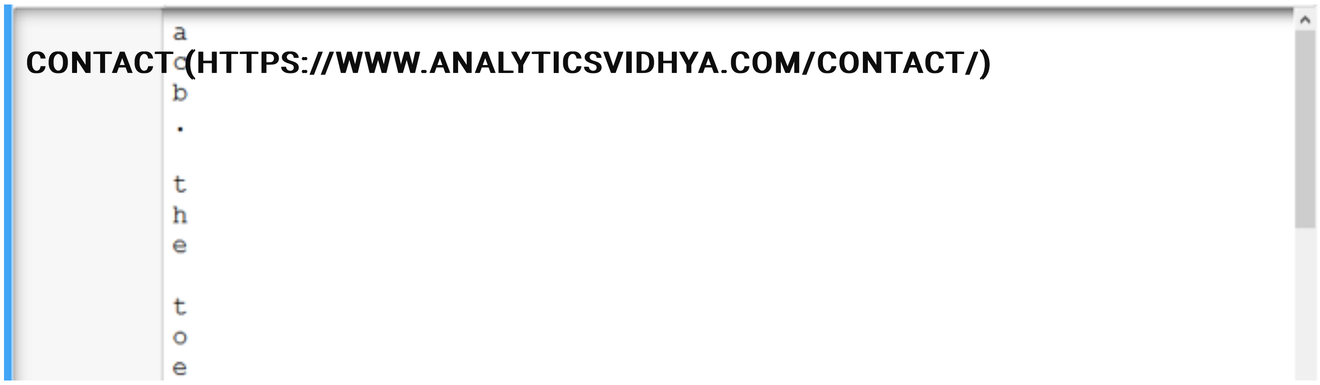
A sequential model which is a linear stack of layers is used. The first layer is an LSTM layer with 300 memory units and it returns sequences. This is done to ensure that the next LSTM layer receives sequences and not just randomly scattered data. A dropout layer is applied after each LSTM layer to avoid overfitting of the model. Finally, we have the last layer as a fully connected layer with a '*softmax*' activation and neurons equal to the number of unique characters, because we need to output one hot encoded result.

- **Fitting the model and generating characters**

```
# fitting the model
model.fit(X_modified, Y_modified, epochs=1, batch_size=30)

# picking a random seed
start_index = numpy.random.randint(0, len(X)-1)
new_string = X[start_index]

# generating characters
for i in range(50):
    x = numpy.reshape(new_string, (1, len(new_string), 1))
    x = x / float(len(unique_chars))

    #predicting
    pred_index = numpy.argmax(model.predict(x, verbose=0))
    char_out = int_to_char[pred_index]
    seq_in = [int_to_char[value] for value in new_string]
    print(char_out)

    new_string.append(pred_index)
    new_string = new_string[1:len(new_string)]
```

The model is fit over 100 epochs, with a batch size of 30. We then fix a random seed (for easy reproducibility) and start generating characters. The prediction from the model gives out the character encoding of the predicted character, it is then decoded back to the character value and appended to the pattern.

This is how the output of the network would look like



```
a
c
b
.

t
h
e

t
o
e
```

Eventually, after enough training epochs, it will give better and better results over the time. This is how you would use LSTM to solve a sequence prediction task.

End Notes

LSTMs are a very promising solution to sequence and time series related problems. However, the one disadvantage that I find about them, is the difficulty in training them. A lot of time and system resources go into training even a simple model. But that is just a hardware constraint! I hope I was successful in giving you a basic understanding of these networks. For any problems or issues related to the blog, please feel free to comment below.

Learn (https://www.analyticsvidhya.com/blog), engage (http://discuss.analyticsvidhya.com/) , hack (https://datahack.analyticsvidhya.com/) and get hired (https://www.analyticsvidhya.com/jobs/#/user/)!

You can also read this article on Analytics Vidhya's Android APP GET IT ON Google Play

(//play.google.com/store/apps/details?id=com.analyticsvidhya.android&utm_source=blog_article&utm_campaign=blog&pcampaignid=MKT-Other-global-all-co-prtnr-py-PartBadge-Mar2515-1)

Share this:

(https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/?share=linkedin&nb=1)

(https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/?share=facebook&nb=1)

(https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/?share=twitter&nb=1)

(https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/?share=pocket&nb=1)

(https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/?share=reddit&nb=1)

Like this:

Loading...

TAGS : FORGET GATE (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/FORGET-GATE/), INPUT GATE (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/INPUT-GATE/), LONG SHORT TERM MEMORY NETWORK (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/LONG-SHORT-TERM-MEMORY-NETWORK/), LSTM (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/LSTM/), OUTPUT GATE (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/OUTPUT-GATE/), RNN (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/RNN/), SEQUENCE PREDICTION (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/SEQUENCE-PREDICTION/), TEXT GENERATOR (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/TEXT-GENERATOR/), TIME SERIES (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/TIME-SERIES/)

PREVIOUS ARTICLE

Fundamentals of Deep Learning – Introduction to Recurrent Neural Networks
(https://www.analyticsvidhya.com/blog/2017/12/introduction-to-recurrent-neural-networks/)

NEXT ARTICLE

Introduction to Computational Linguistics and Dependency Trees in data science
(https://www.analyticsvidhya.com/blog/2017/12/intr
computational-linguistics-

dependency-trees/)

(https://www.analyticsvidhya.com/blog/author/pranj52/)

Pranjal Srivastava (Https://Www.analyticsvidhya.com/Blog/Author/Pranj52/)

I am a Senior Undergraduate at IIT (BHU), Varanasi and a Deep Learning enthusiast. Data is surely going to be the biggest thing of this century, instead of witnessing this as a mere spectator, I chose to be a part of this revolution.

(mailto:pranjal.srivastava.cer14@iitbhu.ac.in)

(https://www.facebook.com/pranjal.srivastava3)

(https://www.linkedin.com/in/pranjal52/)

20 COMMENTS

BIKRAM KACHARI Reply

December 11, 2017 at 10:56 am (https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/#comment-146965)

Very nicely written

PRANJAL SRIVASTAVA Reply

[December 12, 2017 at 11:23 am (https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/#comment-147160)](https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/#comment-147160)

Thanks.

SUJATHA SIVARAMAN **Reply**

December 11, 2017 at 11:05 am (https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/#comment-146966)

Very well articulated article on LSTM

PRANJAL SRIVASTAVA **Reply**

December 12, 2017 at 11:26 am (https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/#comment-147162)

Thank You.

JAMES CHIBOLE **Reply**

December 11, 2017 at 1:18 pm (https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/#comment-146977)

Simple explanations. Thank you.

PRANJAL SRIVASTAVA **Reply**

December 12, 2017 at 12:05 pm (https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/#comment-147167)

Thanks.

SAURABH SINGH **Reply**

December 12, 2017 at 9:57 am (https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/#comment-147144)

Very nice article … and very well explained . Thanks for the quality article .

PRANJAL SRIVASTAVA Reply

December 12, 2017 at 11:19 am (https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/#comment-147159)

Thanks. Glad you liked it.

SASIKANTH Reply

December 14, 2017 at 9:10 pm (https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/#comment-147661)

Is there a similar package in R language to invoke LSTM?

PRANJAL SRIVASTAVA Reply

December 23, 2017 at 4:51 pm (https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/#comment-149220)

Yes. Keras is available for R as well. Please checkout https://keras.rstudio.com/ (https://keras.rstudio.com/)

RICHARD LUCIUS Reply

December 18, 2017 at 11:31 pm (https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/#comment-148597)

Your prediction section should be indented. When you copy an pasted into this format you last the indenting. The prediction does not look like it is part of the for loop.

PRANJAL SRIVASTAVA Reply

December 23, 2017 at 4:57 pm (https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/#comment-149223)

Thanks Richard for pointing it out. Have made the necessary changes.

SRINIVAS Reply

January 11, 2018 at 12:33 pm (https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/#comment-150719)

This is the best article i ever found on LSTM.You nailed it with good Examples..Waiting for other articles from you

---

PRANJAL SRIVASTAVA Reply

March 23, 2018 at 5:38 pm (https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/#comment-152118)

Thanks Srinivas.

BILL Reply

March 8, 2018 at 1:13 pm (https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/#comment-151767)

Hi
I'm a starter and interested in this subject. I think I like the way you said be part not a spectator. If you could, let me know how I could engage into this subject and where do I start?

VENKATRAJU Reply

March 16, 2018 at 9:34 pm (https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/#comment-151949)

This is very nice article and explained well . Thank you

NEHRU Reply

March 27, 2018 at 2:03 pm (https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/#comment-152203)

Actually the question is what is the benfit getting by multiplying Tanh and sigmoid functions to take input and what is the necessity of Tanh function

---

FAIZAN SHAIKH Reply

March 27, 2018 at 4:20 pm (https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/#comment-152206)

Hey – There is a whole article written on this topic. I suggest you to refer this article "Activation Functions and their use (https://www.analyticsvidhya.com/blog/2017/10/fundamentals-deep-learning-activation-functions-when-to-use-them/)"

EI EI MON Reply

May 24, 2018 at 9:33 am (https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/#comment-153509)

Thank you.

SHREESH GUPTA Reply

August 4, 2018 at 12:24 pm (https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/#comment-154436)

well written content !!
thanks dude!!

ANALYTICS VIDHYA

About Us (http://www.analyticsvidhya.com/about-me/)

Our Team (https://www.analyticsvidhya.com/about-me/team/)

Career (https://www.analyticsvidhya.com/career-analytics-vidhya/)

Contact Us (https://www.analyticsvidhya.com/contact/)

Write for us (https://www.analyticsvidhya.com/about-me/write/)

DATA SCIENTISTS

Blog (https://www.analyticsvidhya.com/blog/)

Hackathon (https://datahack.analyticsvidhya.com/)

Discussions (https://discuss.analyticsvidhya.com/)

Apply Jobs (https://www.analyticsvidhya.com/jobs/)

Leaderboard (https://datahack.analyticsvidhya.com/

COMPANIES

Post Jobs (https://www.analyticsvidhya.com/corporate/)

Trainings (https://trainings.analyticsvidhya.com)

Hiring Hackathons (https://datahack.analyticsvidhya.com/)

Advertising (https://www.analyticsvidhya.com/contact/)

Reach Us (https://www.analyticsvidhya.com/contact/)

JOIN OUR COMMUNITY :

(https://www.facebook.com/Analytic

46336

(https://www.facebook.com/Analytic

Followers

(https://www.facebook.com/Analytic

(https://plus.google.com/+Analyticsvi

(https://plus.google.com/+Analyticsvi

Followers

(https://plus.google.com/+Analyticsvi

(https://twitter.com/analytic

23045

(https://twitter.com/analytic

Followers

(https://twitter.com/analytic

(https://in.linkedin.com/company/analytics-vidhya)

7513

(https://in.linkedin.com/company/analytics-vidhya)

Followers

(https://in.linkedin.com/company/analytics-vidhya)

Subscribe to emailer >

Privacy Policy (https://www.analyticsvidhya.com/privacy-policy/)

Terms of Use (https://www.analyticsvidhya.com/terms/)

Refund Policy (https://www.analyticsvidhya.com/refund-policy/)

Don't have an account? Si