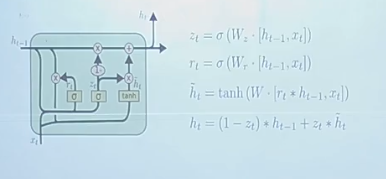


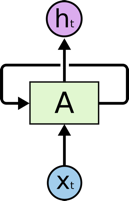
GRU



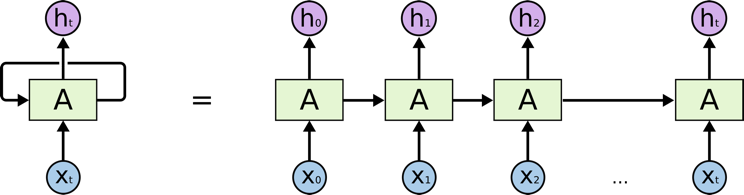
**Recurrent Neural Networks (RNN)**

Being human, when we watch a movie, we don’t think from scratch every time while understanding any event. We rely on the recent experiences happening in the movie and learn from them. But, a conventional neural network is unable to learn from the previous events because the information does not pass from one step to the next. On contrary, RNN learns information from immediate previous step.

For example, there is a scene in a movie where a person is in a basketball court. We will improvise the basketball activities in the future frames: an image of someone running and jumping probably be labeled as *playing basketball*, and an image of someone sitting and watching is probably *a spectator watching the game.*



A typical RNN (Source: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>)



A typical RNN looks like above-where X(t) is input, h(t) is output and A is the neural network which gains information from the previous step in a loop. The output of one unit goes into the next one and the information is passed.

But, sometimes we don’t need our network to learn only from immediate past information. Suppose we want to predict the blank word in the text ‘ David, a 36-year old man lives in San Francisco. He has a female friend Maria. Maria works as a cook in a famous restaurant in New York whom he met recently in a school alumni meet. Maria told him that she always had a passion for \_\_\_\_\_\_\_\_\_ . Here, we want our network to learn from dependency ‘cook’ to predict ‘cooking. There is a gap between the information what we want to predict and from where we want it to get predicted . This is called long-term dependency. We can say that anything larger than trigram as a long term dependency. Unfortunately, RNN does not work practically in this situation.

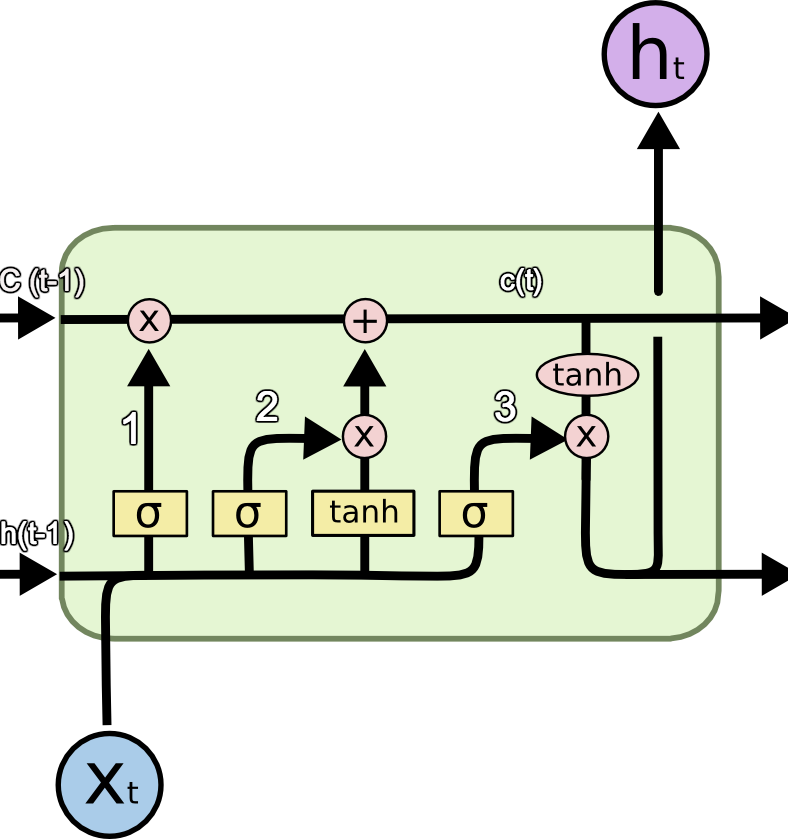
**Why RNN does not work practically**

During the training of RNN, as the information goes in loop again and again which results in very large updates to neural network model weights. This is due to the accumulation of error gradients during an update and hence, results in an unstable network. At an extreme, the values of weights can become so large as to overflow and result in NaN values.The explosion occurs through exponential growth by repeatedly multiplying gradients through the network layers that have values larger than 1 or vanishing occurs if the values are less than 1.

**Long Short Term Memory**

The above drawback of RNN pushed the scientists to develop and invent a new variant of the RNN model, called Long Short Term Memory. LSTM can solve this problem, because it uses gates to control the memorizing process.

Let’s understand the architecture of LSTM and compare it with that of RNN:



A LSTM unit (Source : [http://colah.github.io/posts/2015-08-Understanding-LSTMs](http://colah.github.io/posts/2015-08-Understanding-LSTMs/))

The symbols used here have following meaning:

a) X : Scaling of information

b)+ : Adding information

c) σ : Sigmoid layer

d) tanh: tanh layer

e) h(t-1) : Output of last LSTM unit

f) c(t-1) : Memory from last LSTM unit

g) X(t) : Current input

h) c(t) : New updated memory

i) h(t) : Current output

**Why tanh?**

To overcome the vanishing gradient problem, we need a function whose second derivative can sustain for a long range before going to zero. *tanh* is a suitable function with the above property.

**Why Sigmoid?**

As Sigmoid can output 0 or 1, it can be used to forget or remember the information.

Information passes through many such LSTM units.There are three main components of an LSTM unit which are labeled in the diagram:

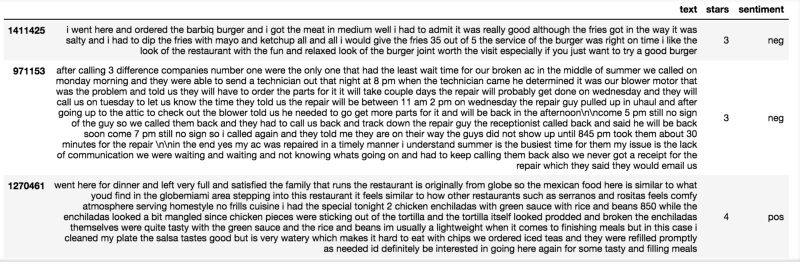
1. LSTM has a special architecture which enables it to forget the unnecessary information .The sigmoid layer takes the input X(t) and h(t-1) and decides which parts from old output should be removed (by outputting a 0). In our example, when the input is ‘He has a female friend Maria’, the gender of ‘David’ can be forgotten because the subject has changed to ‘Maria’. This gate is called forget gate f(t). The output of this gate is f(t)\*c(t-1).
2. The next step is to decide and store information from the new input X(t) in the cell state. A Sigmoid layer decides which of the new information should be updated or ignored. A *tanh* layer creates a vector of all the possible values from the new input. These two are multiplied to update the new cell sate. This new memory is then added to old memory c(t-1) to give c(t). In our example, for the new input ‘ He has a female friend Maria’, the gender of Maria will be updated. When the input is ‘Maria works as a cook in a famous restaurant in New York whom he met recently in a school alumni meet’, the words like ‘famous’, ‘school alumni meet’ can be ignored and words like ‘cook, ‘restaurant’ and ‘New York’ will be updated.
3. Finally, we need to decide what we’re going to output. A sigmoid layer decides which parts of the cell state we are going to output. Then, we put the cell state through a *tanh* generating all the possible values and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to. In our example, we want to predict the blank word, our model knows that it is a noun related to ‘cook’ from its memory, it can easily answer it as ‘cooking’. Our model does not learn this answer from the immediate dependency, rather it learnt it from long term dependency.

We just saw that there is a big difference in the architecture of a typical RNN and a LSTM. In LSTM, our model learns what information to store in long term memory and what to get rid of.

**Quick implementation of LSTM for Sentimental Analysis**

Here, I used LSTM on the reviews data from [Yelp open dataset](https://www.yelp.com/dataset) for sentiment analysis using keras.

This is what my data looks like.



Dataset

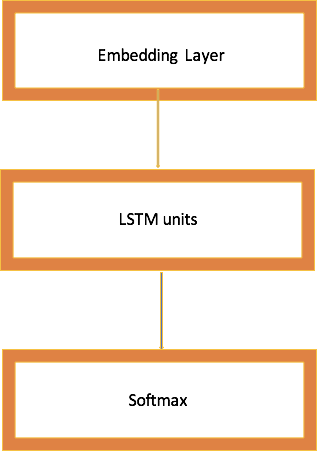
I used Tokenizer to vectorize the text and convert it into sequence of integers after restricting the tokenizer to use only top most common 2500 words. I used pad\_sequences to convert the sequences into 2-D numpy array.

Then, I built my LSTM network.There are a few hyper parameters:

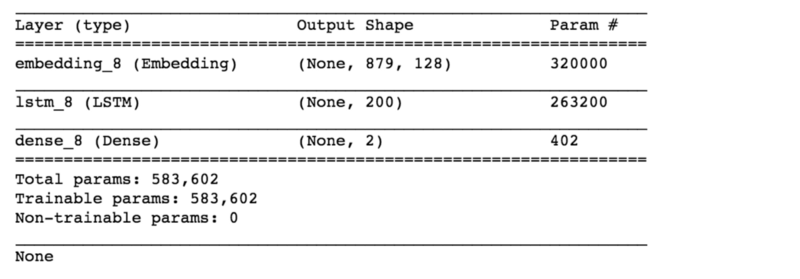
1. embed\_dim : The embedding layer encodes the input sequence  
   into a sequence of dense vectors of dimension embed\_dim.
2. lstm\_out : The LSTM transforms the vector sequence into a single vector of size lstm\_out, containing information about the entire sequence.

The other hyper parameters like dropout, batch\_size are similar to that of CNN.

I used softmax as activation function.

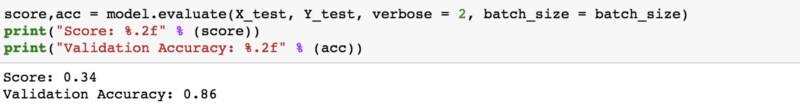


LSTM network



Now, I fit my model on training set and check the accuracy on validation set.

https://cdn-images-1.medium.com/max/800/1*XfPXSNqVb3vc5_jTRl-Q3w.png



I got a validation accuracy of 86% in just one epoch while running on a small dataset which includes all the businesses.

**Future Work:**

1. We can filter the specific businesses like restaurants and then use LSTM for sentiment analysis.
2. We can use much larger dataset with more epochs to increase the accuracy.
3. More hidden dense layers can be used to improve the accuracy. We can tune other hyper parameters as well.

**Conclusion**

LSTM outperforms the other models when we want our model to learn from long term dependencies. LSTM’s ability to forget, remember and update the information pushes it one step ahead of RNNs.