Using RNN to Predict Stock Market Prices

Ford Smith

February 11, 2019

Abstract

//TODO

1 Introduction

When most people talk about Artificial Intelligence, they are describing machine learning. Machine learning has several difference subsections of tools that can be used to teach a machine from previous experiences by adjusting to inputs to "perform human-like tasks" (SAS). In this particular experiment, Neural Networks will be used. Neural Networks are very powerful tools because they grant flexibility to many different scenarios depending on how they are constructed.

History At its core, machine learning is all about taking in inputs and creating a generalized function that can then be extended to predict future values. In fact, humans have been doing basic variations of this for hundreds of years: linear regressions. While some function are too difficult to create, a line of best fit can be found on data with an input (x) and output (y) to create a function that can approximate or predict a different set of data. See Figure 1.

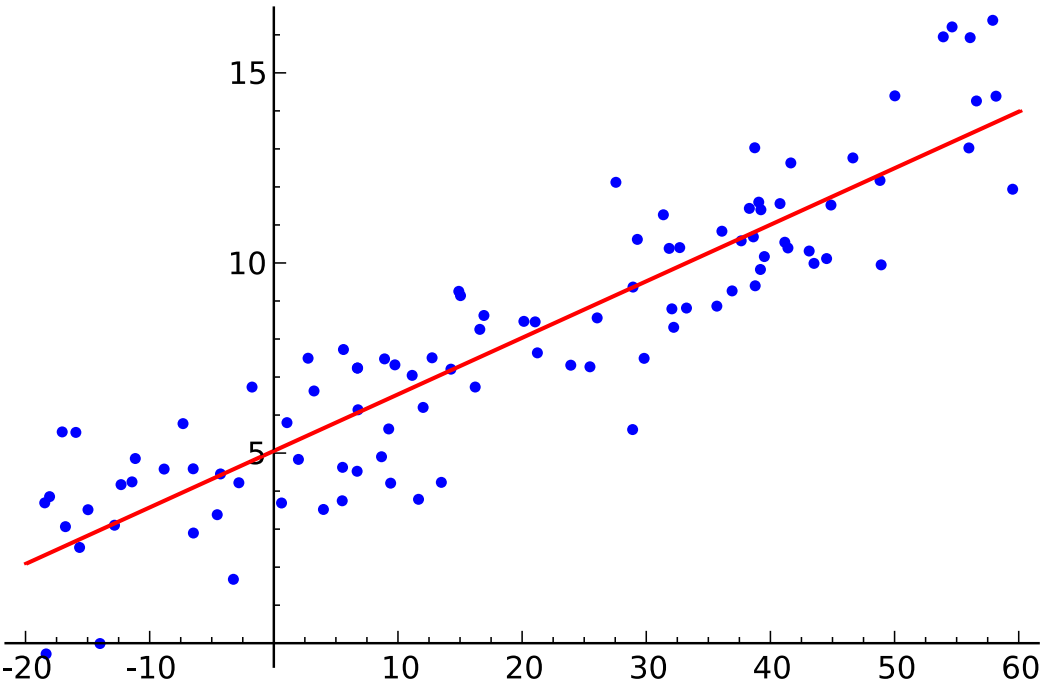


Figure 1: An example of linear regression used on input data to create a function to approximate more data

This type of learning is called supervised machine learning because through the process of approximating the function, input and output data can be used to make the approximation much more accurate. Unsupervised learning, although not important in this experiment, is learning where there are not output values so different learning methods are used to learn more about the inputs and how it is structured.

Skip forward about 150 years to the 1950s, there is the development of the Perceptron. The Perceptron was created by Frank Rosenblatt to be a mathematical model of the neurons in our brains. It takes a set of inputs from nearby Perceptrons, multiplies each of the inputs by a valued weight, and will output a 1 if the weighted inputs reach a threshold, otherwise a 0. This was massive back then because Perceptrons could create basic OR/AND/NOT functions. This was the gateway to formal logic for computers. The most exciting part, however, was that this model could learn by slightly adjusting the weights whenever the output is too low/high. It followed this general

order: ”

1. Start off with a Perceptron having random weights and a training set
2. For the inputs of an example in the training set, compute the Perceptron's output
3. If the output of the Perceptron does not match the output that is known to be correct for the example: If the output should have been 0 but was 1, decrease the weights that had an input of 1. If the output should have been 1 but was 0, increase the weights that had an input of 1.
4. Go to the next example in the training set and repeat steps 2-4 until the Perceptron makes no more mistakes”

This procedure is simple, not terribly computationally heavy, and works best when there is only a limited set of outputs due to thresholds. This is exactly what classification requires - out of a set, accurately determine which output the input data is. Rosenblatt implemented the Perceptrons and inputted 20x20 inputs to accurately classify shapes. Although simple, when many Perceptrons are put together in series called **layers**, the computation power becomes much higher and can work on much more complex data. The same happens with multiple layers, where data is put into the input layer, which feeds their output to ”hidden” layers who pass their information on to another layer until the information is passed to an output layer. The importance of these hidden layers stems from their ability to find features within the data. Features are essentially specific parts about the data passing through. For instance, if the network was trained to identify whether there was a cat in the image, the layer could identify whether the image has a cat mouth or not. This, in turn, allows it to become more accurate. However, there becomes several serious issues when done. First, Rosenblatt's method of training does not work with multiple layers. The Perceptrons also can't work on complex features because of the extreme linearity of the weight function inside the Perceptrons.

Modern Networks **Backpropagation** was developed across multiple researchers in the 60s and is still commonly used. To break down what backpropagation is, is there is a cost function C with respect to the weights w in

the network. Take the partial derivative of C in respect to w to then see how quickly the cost changes when the weights change. The reason backpropagation is so important was because it not only became a much faster learning algorithm, but also describes detailed information about the change in the system in respect to the weights.

With a newer, faster, and all around better learning algorithm, there is still bottle-necking by the actual Perceptron itself. When there is a tiny change in its weights, it could cause a drastic change in the output, which actually makes training more difficult. If there is more fine control of the weights to have more fine control over the output, it is possible to more accurately train layers to get the results desired. This is where the sigmoid function comes in handy. If the threshold is replaced with the sigmoid function, as pictured below, there will be more fine change in the output with the change in the weights, giving a higher degree of accuracy.

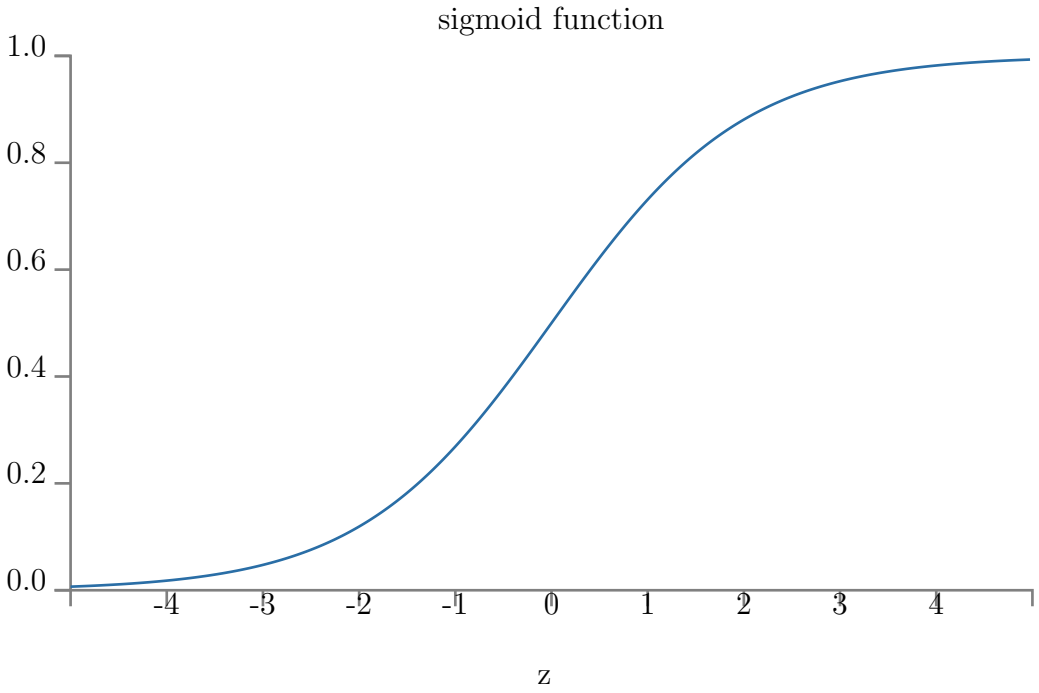


Figure 2: An example of a sigmoid function

You may be wondering, why use the sigmoid function instead of just

having the threshold? Well, the threshold is very linear and doesn't give much opportunity to learn around the data. When the **activation function** is non-linear (like the sigmoid function) it allows the network as a whole to better adapt to the data. Note, the activation function is what is used to determine the output of the nodes. Here is an image of what a complete Neural Network will use as Figure 3.

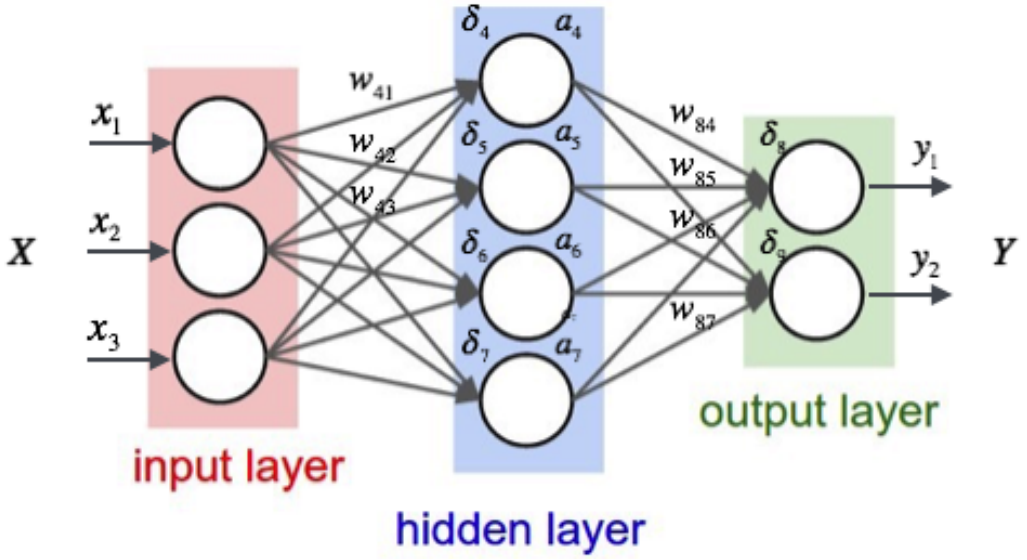


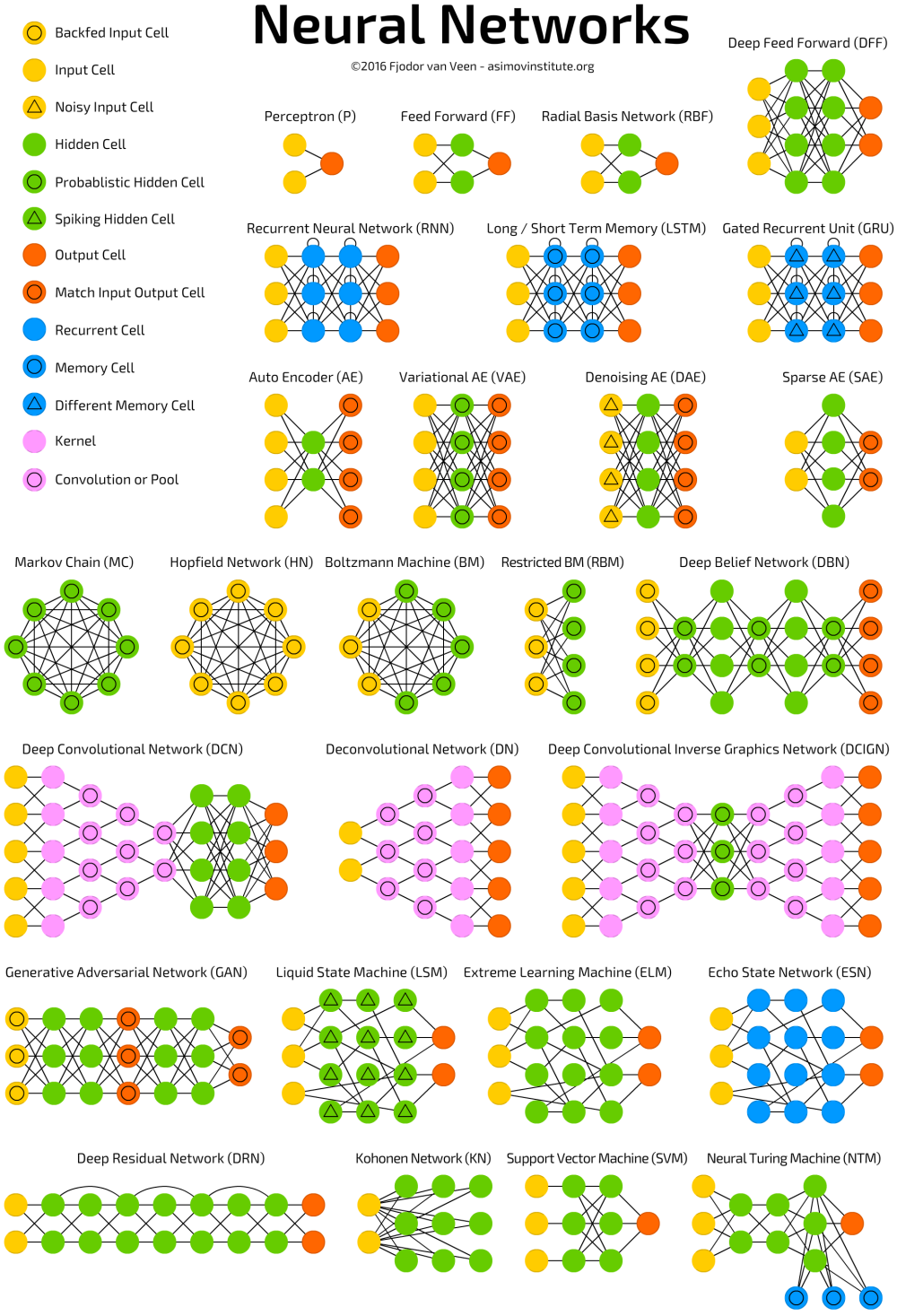
Figure 3: A neural network

2 Different Types

With the basics of what a neural network is and how it developed, it is important to cover the different types of Neural Networks - which will now just be referred to as NN - to best decide which will be used for this experiment. You will understand which one will be chosen and why it is the only one that can work in a moment. The main types are demonstrated in a graphic by Fjodor van Veen from Asimov Institute.

A mostly complete chart of

Neural Networks



Choosing There are many different options to choose from, however, the majority of them can't be used. Why? In this particular experiment, NN are being used to predict stock prices. The stock market doesn't arbitrarily choose the price every second, instead, the price is a derived from the previous prices to create a new price (it is also important to note stocks are very dependent on the news around the company). This *requires* the NN to remember previous information to make informed decisions. As a result, only Recurrent Neural Networks (RNN) can be used. Something to note, GRU and LSTM are a subsection of RNN not demonstrated in the figure above.

RNN What is so special about RNN? To put it short, RNN are a special type of NN where the previous information directly affects the result of the next step in a sequential order. While in theory they are able to have arbitrarily long pieces of sequential information, a generic RNN is unable to do so. This is very bad in the context of running it on stock data as stock data can be over decades. This is why LSTM - long short term memory - are used.

LSTM While generic RNN struggle to remember long term information, LSTM are designed to do so making them the perfect candidate for stock market data! Below, is going to be the break down of exactly how the LSTM is able to do so.

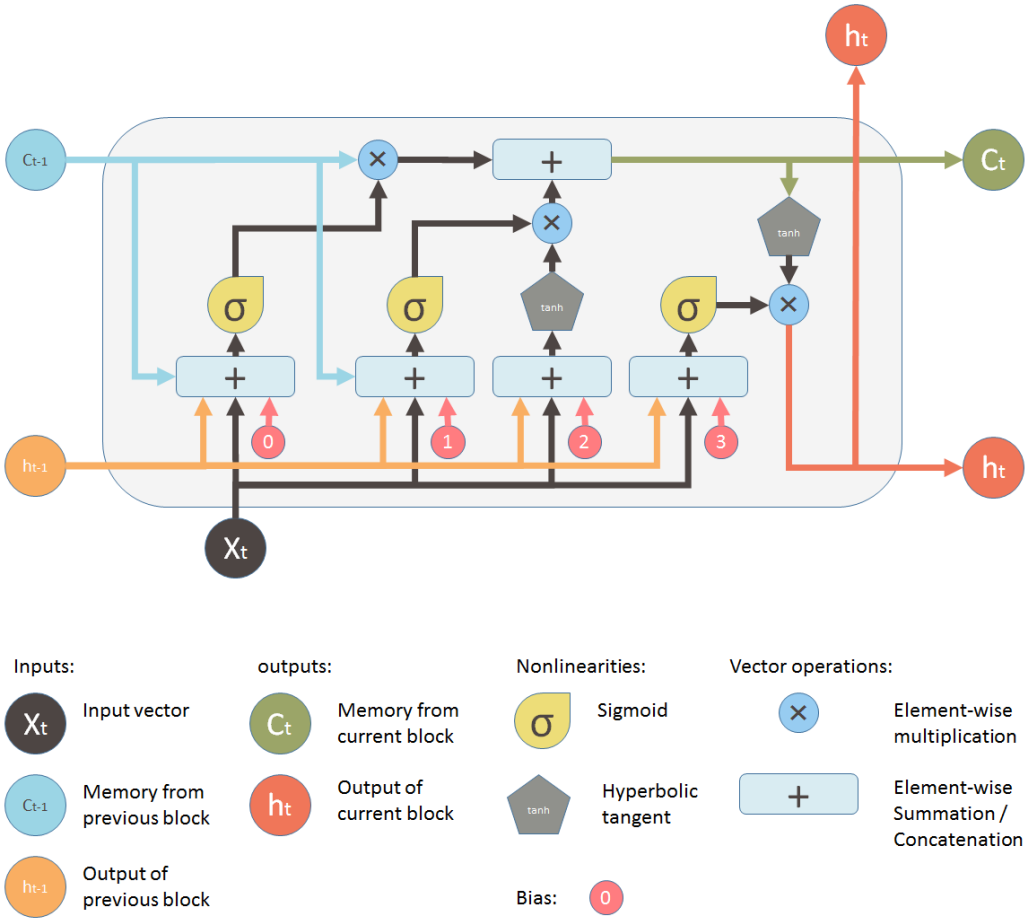


Figure 4: Internals of LSTM cell

Although it looks very intimidating, the operations taking place are fairly straight forward. First, notice what is going in and out of this basic LSTM cell. X_t is the input at the current step, h_{t-1} is the output from the previous LSTM cell and C_{t-1} is the "memory" of the previous cell. Lastly, we have h_t for output of the current network and C_t is the current memory of this cell.

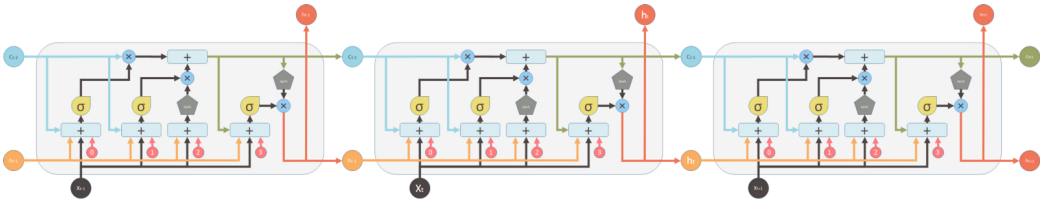


Figure 5: Multiple LSTM strung together in a chain

Here is several cells put together to visualize a segment of what is going on. The top bar (where C_{t-1} is passed) is the transfer of the memory through the network. In Figure 6 (below) we are looking at what is the "forget" gate. The purpose is to use a single layer NN with a sigmoid activation function which is than applied to the old memory. As a result, it can affect the memory that is continued to be passed on (which is how the network doesn't get bogged down by remembering absolutely everything).

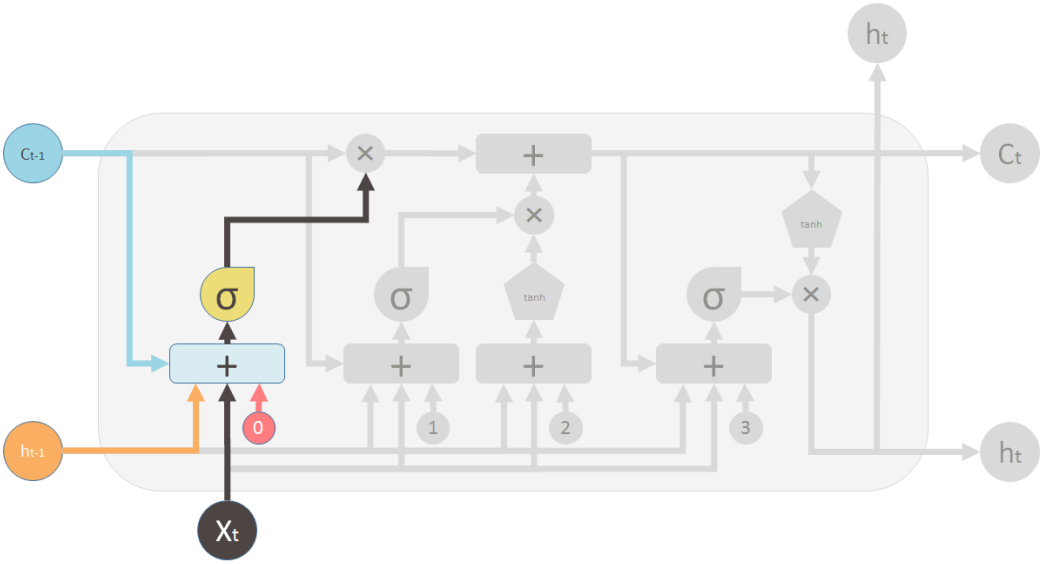


Figure 6: Multiple LSTM strung together in a chain

Afterwards, we go to the "new memory" single layer NN. This NN determines how much the new memory should influence the old memory. However, the new memory is generated by a different single layer NN, this time with tanh as the activation function. The output of both of these is combined with each other and applied to the transferring memory.

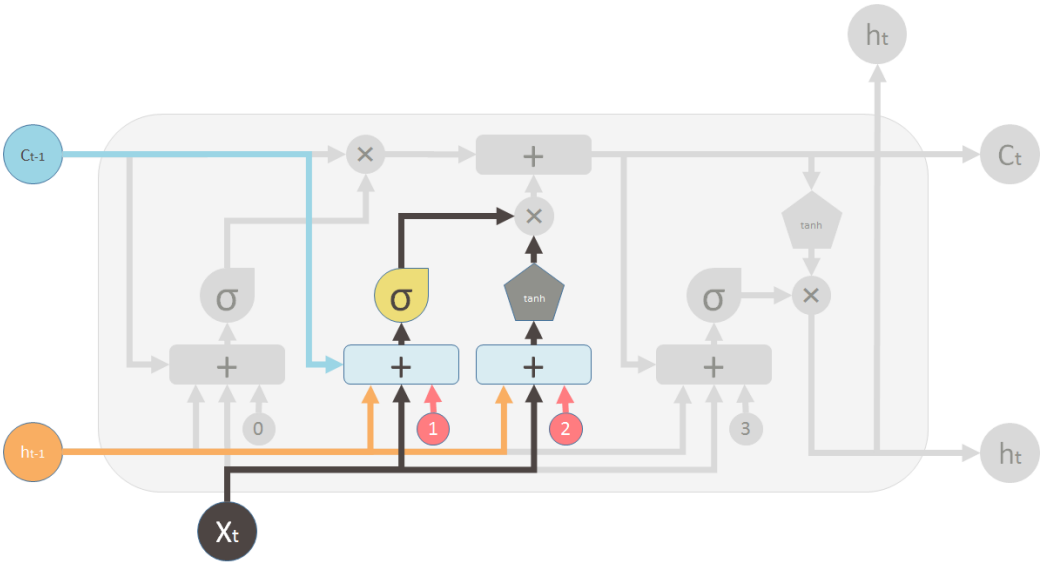


Figure 7: First section of LSTM

Lastly, we need to actually generate the output for this cell. This step once again includes a single layer NN that determines how much the new memory should affect the output of the cell.

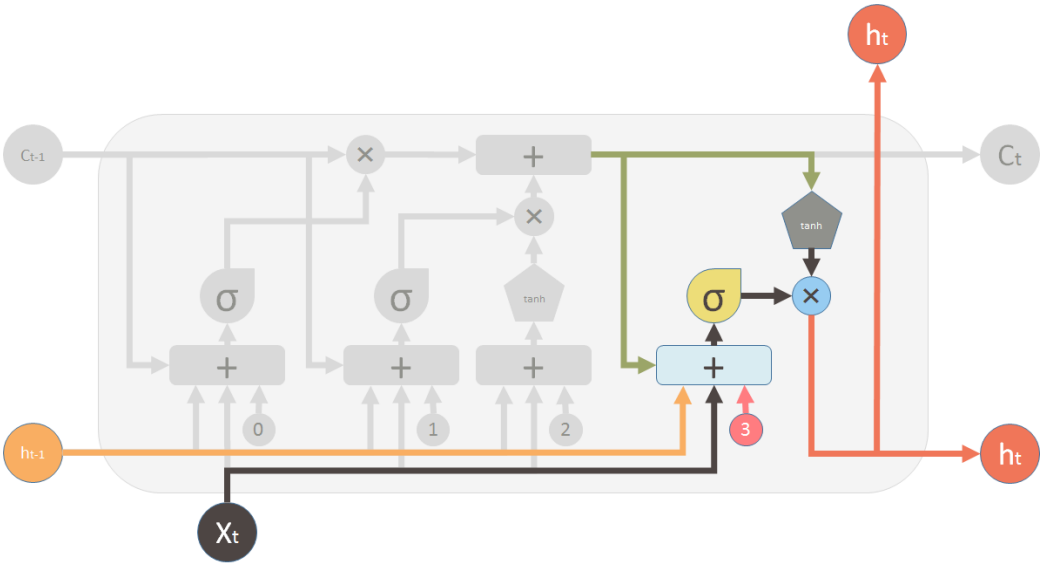


Figure 8: Last section of LSTM

Conclusion In total, because of the ability to be trained to forget unimportant information but still remember the past, it becomes the strongest RNN to use.

3 Results

In this section we describe the results.

4 Conclusions

We worked hard, and achieved very little.