Using RNN to Predict Stock Market Prices

Ford Smith

February 10, 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, we will be using Neural Networks. 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, we have been doing basic variations of this for hundreds of years: linear regressions. While some function are too difficult to create, we can create line of best fits 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.

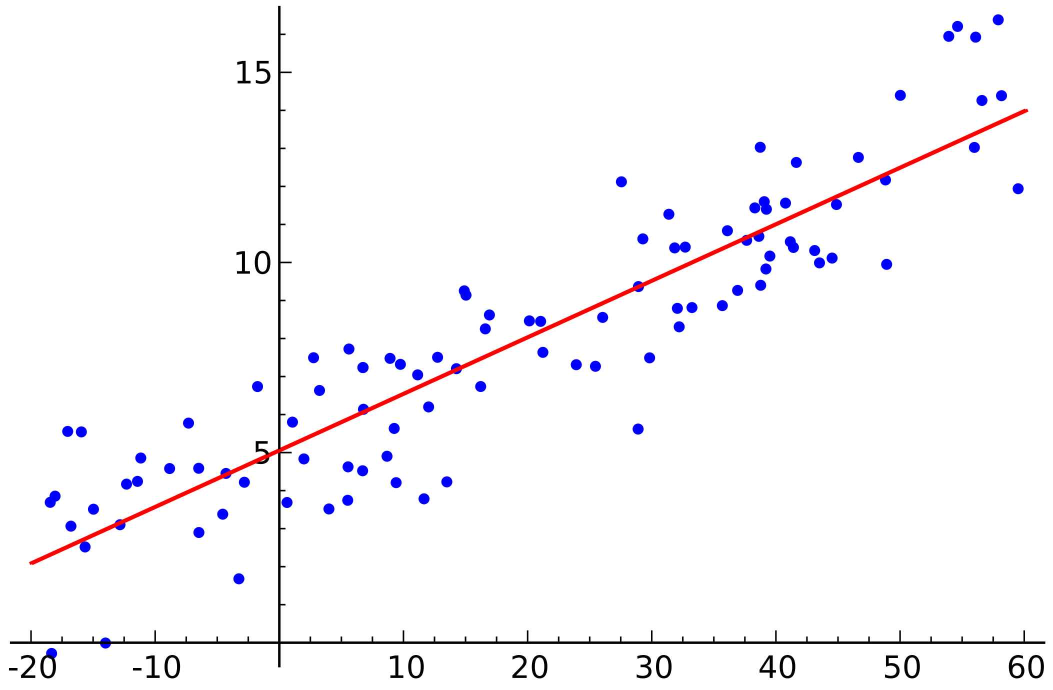

Figure 1: An example of linear regression used on input data to create a function to approximate more data

This type of learning is called supervised machine learning because through the process of approximating the function, we have input and output data that we can use to make the approximation much more accurate. Unsupervised learning, although not important in this experiment, is learning where don't have output values so we use several different learning methods to learn more about the inputs and how it is structured.

Skip forward about 150 years to the 1950s, we get to 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 it when put multiple together, it 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 when we have 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 on identify whether there was a cat in the image, one of the layers could be used to find if the image has a cat mouth or not while another focuses on finding the left cat ear. This, in turn, allows it to become more accurate. However, there becomes several serious issues when we do this. 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 we have a cost function C with respect to the weight w

in the network. We take the partial derivative of C in respect to w to see how quickly the cost changes when we change the weights. The reason back-propagation is so important was because it not only became a much faster learning algorithm, but also tells us detailed information about the change in the system in respect to the weights.

With a newer, faster, and all around better learning algorithm, we were still bottle necked by the actual perceptron itself. When we have some tiny change in its weights, it could cause a drastic change in the output, which actually makes training more difficult. If we can more finely control the weights to have more fine control over the output, we could more accurately train layers to get the results we want. This is where the sigmoid function comes in handy. If we replace the threshold with the sigmoid function, as pictured below, we will then be able more finely change the outputs with the inputs, giving us a higher degree of accuracy.

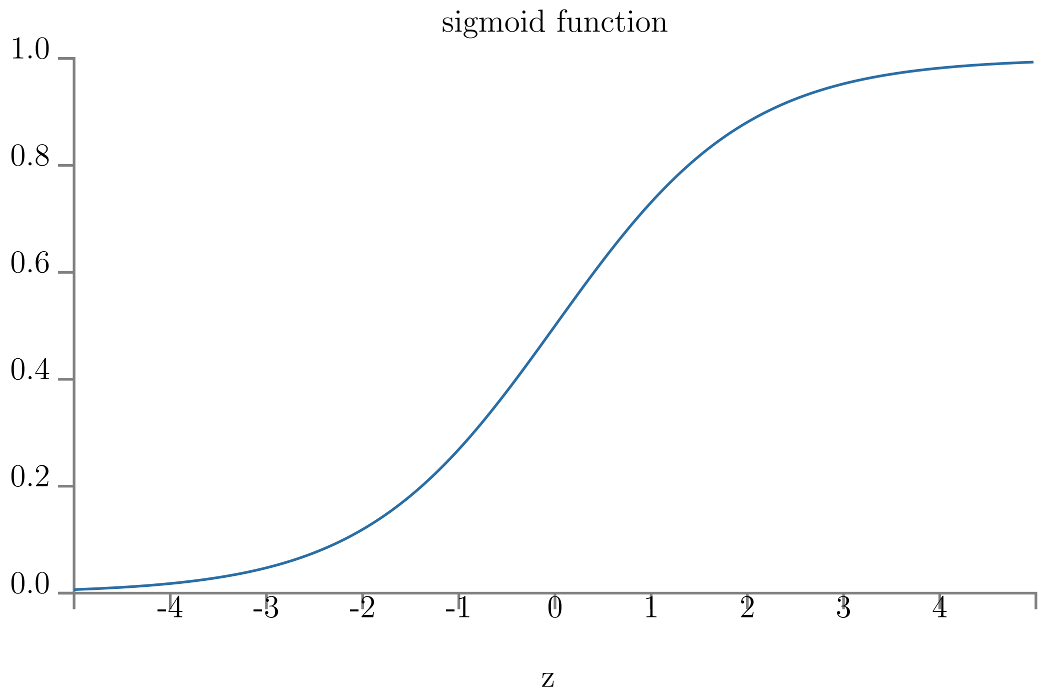

Figure 2: An example of a sigmoid function

2 Previous work

A much longer $\LaTeX\,2_\varepsilon$ example was written by Gil [**?**].

3 Results

In this section we describe the results.

4 Conclusions

We worked hard, and achieved very little.