Data Science Analysis

EP4130 — IIT Hyderabad

Stock Market Analysis

Gautham Gururajan
Raghav Girgaonkar

Contents

1 Abstract

Often, we find ourselves doubting the existence of real world applications of Data Analysis Theory, as it is not trivial to understand what how and where the standard tools are used

This paper aims to capture that very essence by making use of some known data analysis tools to help make predictions and inferences for and of the stock market.

We have chosen the stock market as there exists an abundance of data to check and validate certain hypotheses, and to analyze it, we have chosen the Neural Network prediction system.

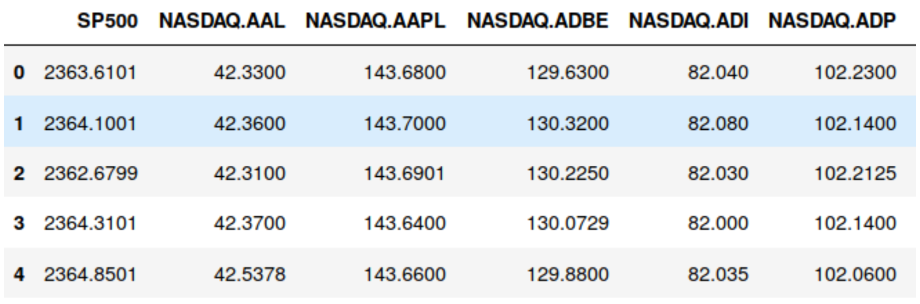
Here, we have chosen to analyze the S&P 500, which is an American stock market index based on the market capitalizations of 500 large companies having common stock listed on the NYSE, NASDAQ, or the Cboe BZX Exchange.

2 Introduction

The S&P 500 is an American stock market index consisting of the market capitalizations of 500 large companies listed on the NYSE, NASDAQ, or the Cboe BZX Exchange.

We acquired a dataset which consisted of the S&P 500 index along with the stock prices of the 500 companies. The data samples present in the data were minute-by-minute stock prices of all the 500 companies which include Apple, Boeing Company and the Goldman Sachs Group. In this project we have built a deep learning model in Tensor-Flow to predict the S&P 500 index using the data of the given 500 hundred companies. The data was properly cleaned and prepared by filling NaN values with LOC (Last Observation Carried forward) to ensure there were no gaps in the dataset.

Dataset can be seen in Figure 1.



	SP500	NASDAQ.AAL	NASDAQ.AAPL	NASDAQ.ADBE	NASDAQ.ADI	NASDAQ.ADP
0	2363.6101	42.3300	143.6800	129.6300	82.040	102.2300
1	2364.1001	42.3600	143.7000	130.3200	82.080	102.1400
2	2362.6799	42.3100	143.6901	130.2250	82.030	102.2125
3	2364.3101	42.3700	143.6400	130.0729	82.000	102.1400
4	2364.8501	42.5378	143.6600	129.8800	82.035	102.0600

Figure 1: First 5 rows and 6 columns of the dataset

3 The BP Neural Network

The *Back Propagation Neural Network*, is an extremely popular method used for prediction models. In this type of neural network, the inputs which are fed in, are propogated in the forward direction and then compared with the actual values which should be obtained. After this the *weights* of each of its *layers* are changed by a reverse process known as *backpropogating.* Thus the neural network *learns* after every batch of input is passed through it and corrects itself by adjusting its *weights* accordingly.

3.1 Basic Terminology

Neuron

The word *neuron* is inspired by the biological neuron present in our bodies as it performs the same function. Neurons pass on information in the form of electric signals in our body and similiarly in our neural network, a *neuron* accepts inputs and their weights and gives the corresponding output according to the ***Activation Function.***

Bias

A bias unit is an "*extra*" neuron added to each pre-output layer that stores the value of 1. Bias units aren't connected to any previous layer and in this sense don't represent a true "*activity*". A bias is introduced as an input to represent what the neural network will output when no inputs are given to it.

Activation Function

The activation function decides, whether a neuron should be activated or not by calculating weighted sum and further adding bias with it. The purpose of the activation function is to introduce non-linearity into the output of a neuron.

3.2 Working

A Neural Network is usually made up of multiple layers. Of course, one input and one output layer, and included are one or many hidden layers for the purpose of

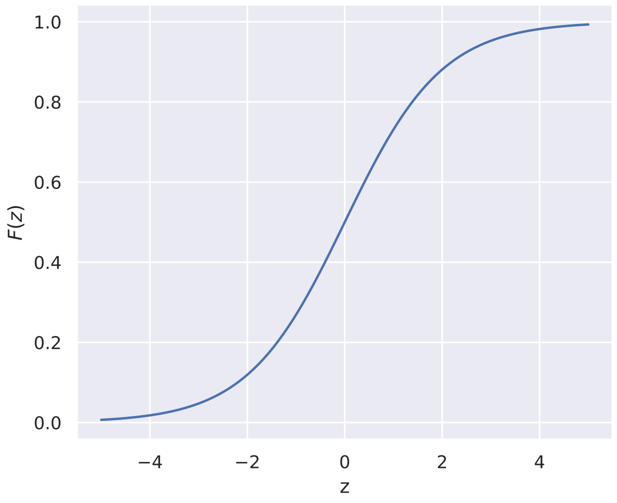


Figure 2: Activation Function: Sigmoid Function

filtering the data in a sense.

Let us assume a case of a total of M layers where $m = 1$ represents the first layer(Input) and $m = M$ represents the last layer(Output).

We shall use w^m_{jk} to denote the weight for the connection made between the kth neuron in the $(m-1)$th layer to the jth neuron in the mth layer.

Also, we use b^m_j for the bias of the jth neuron in the mth layer, and for the activation of the jth neuron in the mth layer, we use a^m_j

Further, the activation a^m_j is defined as

$$a^m_j = \sigma\left(\sum_{k=0} w^m_{jk} * a^k_{j-1} + b^m_j \right)$$

where σ is the activation function.

This then propogates in the forward direction of the neural network and the weights of each neuron in every layer is determined, this is called *Forward Propogation.*

The Cost Function is calculated as follows

$$C = \frac{1}{2n} \sum_x ||y(x) - a_L(x)||^2 = \frac{1}{n} \sum_x C_x$$

We then use the *Hadamard Product* to calculate the intermediate error function δ

$\delta^L = \nabla_a C \odot \sigma'(z^L)$ where $z^l = w^l a^{l-1} + b^l$

With learning rates η the weights are learned as follows

$w^m_{jk} \rightarrow w^m_{jk} - \eta \frac{\partial C}{\partial w^m_{jk}}$ and

$b^m_j \rightarrow b^m_j - \eta \frac{\partial C}{\partial b^m_j}$

We then use *Gradient Descent* to find the optimal parameters for our neural network.

3.3 Design Implementation

Obviously, amongst the many elements used, there must be a type of element or choice of function that gives us optimum performance depending on where it is applied. Following are the popular options of the same, and our design choices. Since our choice was to work with an extremely large data set, these design choices would have a noticeable difference in performance and run time.

Activation Function

: There are mainly 3 types of Activation Functions :

- Binary Function
- Linear Function
- Non-Linear Function

The problem with a binary function is that it does not allow multi-value outputs : for example, it cannot support classifying the inputs into one of several categories. Now, the only choice we have is between the Linear and the Non-Linear Functions. The problem with a linear function is that it does not allow Back Propagation - The derivatives are in no relation to the input as they are constant, so it won't be possible to go back and improve parameters. Also, no matter how many layers you have in your network, under a Linear Function they all collapse into a single one. Our Non-Linear Function does not face such problems. Within the domain of non-linear functions, there are many used, such as the Sigmoid, ReLu, Tanh function

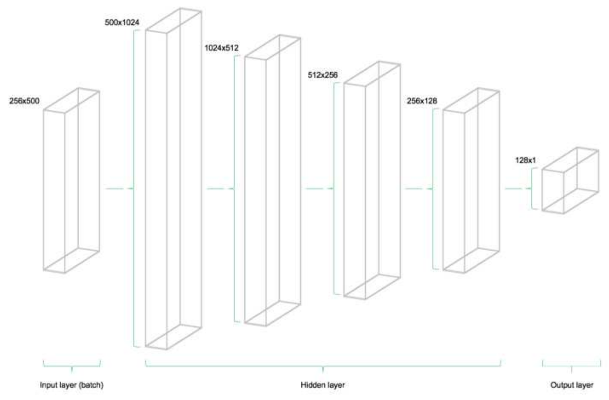


Figure 3: Schematic of the Neural Network used

etc. We have chosen the ReLu function for our model for the simple reason that defeats Sigmoid and Tanh type functions, with the property of reduced likelihood of Vanishing Gradients.

Number of Layers and Neurons

A case of having the number of neurons greater than the previous layer may result in *Overfitting*(Overitting occurs when the neural network has so much information processing capacity that the limited amount of information contained in the training set is not enough to train all of the neurons in the hidden layers), but having too less neurons will result in *Underfitting*(Underfitting occurs when there are too few neurons in the hidden layers to adequately detect the signals in a complicated data set).

The choice of number of layers and number of neurons have no specific answer, but conventionally, the number of neurons in the hidden layer is taken to be a number between the number of neurons in the hidden layer before it and the output number of neurons.

For the sake of easy visualization and aesthetic simplicity we have chosen every two consecutive hidden layer to differ by a multiplication factor of 2. A good read, here shows us that the number of hidden layers we'll need must be more than 2. Increasing the number of layers, as we see will cause an increase in run time which is unfavorable so we choose the number of layers in our model to be based on the best performance for least runtime(between 3 and 6 layers, checked with trial and error gives us a number of 4 Hidden Layers)as attached in Figure 3. Ultimately, the selection of architecture comes down to trial and error, and has no real rule of thumb, but the above choice of ours was the most straightforward logical trial.

Choice of Gradient Descent

There are various types of Gradient descent, ie: Batch Gradient Descent, Stochastic Gradient Descent, Mini Batch Gradient Descent etc.

The Stochastic Gradient Descent is a simple variant of classical gradient descent where the stochasticity comes from employing a random subset of the measurements (mini-batch) to compute the gradient at each descent. It also has implicit regularisation effects, making it suited for highly non-convex loss functions like in our case.

The Stochastic Gradient descent takes less computation time than it's other counterparts, and for this reason we choose it.

4 Results

We first plotted the S&P 500 index over time as shown in Figure 4. This is actually the lead of the S&P 500 index, meaning, its value is shifted 1 minute into the future. This operation is necessary since we want to predict the next minute of the index and not the current minute. Each row in the dataset contains the price of the S&P500 at t+1 and the constituent's prices at T=t.

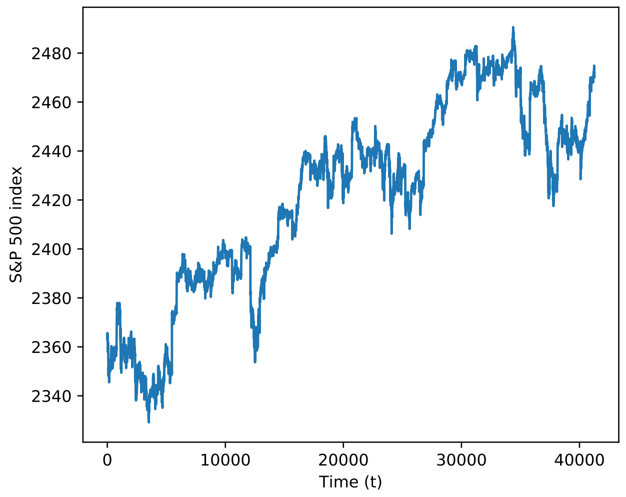


Figure 4: First 5 rows and 6 columns of the dataset

The data was then divided into batches and then passed on into the neural network where it *learned* over the course of 10 ***epochs***. The following figures display the prediction of the neural network(in orange) plotted over the original plot of Figure 4.

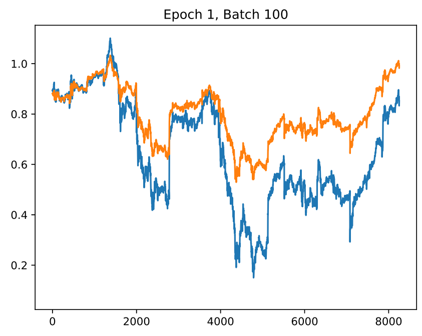


Figure 5: Epoch 1 batch 100

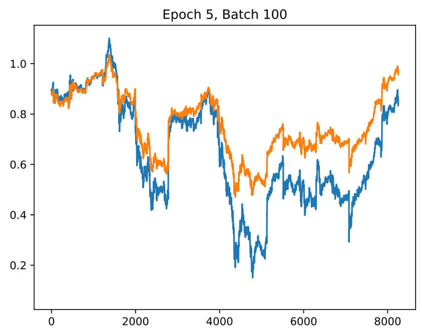


Figure 6: Epoch 5 batch 100

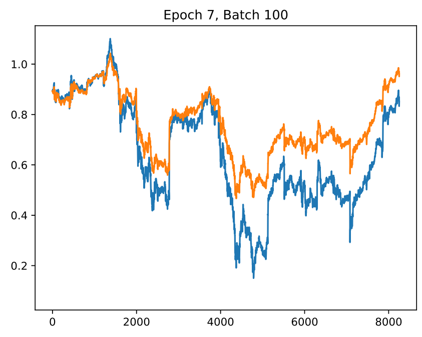


Figure 7: Epoch 7 batch 100

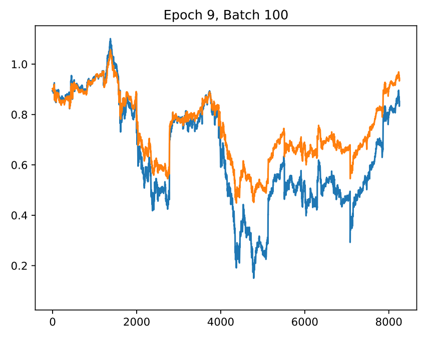


Figure 8: Epoch 9 batch 100

As we see the prediction of the Neural Network (in orange) converges to the actual plot (in blue) near the end of the 10th epoch.

5 Conclusion

The whole motivation behind this paper, was to use simple data manipulation tools and fairly simple data analysis theory to produce a strong real life application of the same. Through the Predicted vs Actual figure shown above, we see that even though we are not exactly accurate, we have managed to make a decent guess with acceptable error margins.

6 References

- https://www.heatonresearch.com/2017/06/01/hidden-layers.html
- https://arxiv.org/abs/1805.11317
- https://www.analyticsindiamag.com/how-stochastic-gradient-descent-is-solving-optimisation-problems-in-deep-learning/
- https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/

[1]

[1] All codes and figures can be found at https://github.com/ArcticSoup/Stock-Market-Analysis or https://github.com/Gautham-G/Market_Analysis