MLP Stock Prediction

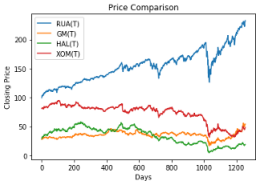
Banatwalla, Muhammad April 2nd, 2021

**1 Introduction**

For this homework assignment we are tasked with implementing a multi layer perceptron in order to predict the performance of a stock. We will test multiple versions of an MLP to determine the best one for our data

**1.1 Description of Data**

For the four stocks used in our data we chose the Russell 3000 Index (RUA), General Motors (GM), Halliburton Company (HAL), and Exxon Mobil Corporation (XOM). We chose these stocks because we are interested in seeing how energy (represented by Exxon), energy services (represented by Halliburton), and General Motors (a proxy for energy consumption), correlate to the overall performance of the stocks market (Russell 3000 is a proxy for the economy both consumers and businesses).The dataset is chosen from 2/12/2016 to 2/4/2021 of closing prices. Below is a graph of the data and correlation matrix:



1



Our target variable is the closing price of the General Motors stock on day T+1. In general, as the market goes up, these energy related services will go down. We can use this relationship to predict the performance of General Motors. For the input vector of our data, we will use 4 consecutive daily closing prices. In total, the dataset has 1255 rows and 16 features. Due to the poor performance and some oddities observed during the process, we cloned some randomly chosen cases in our data and increased the number of cases too 1504. The 16 features are shown below:

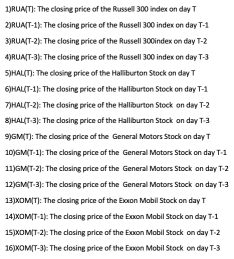
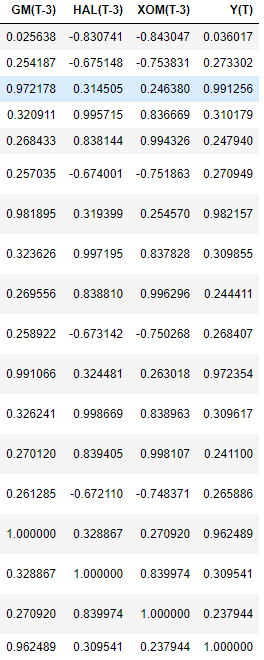
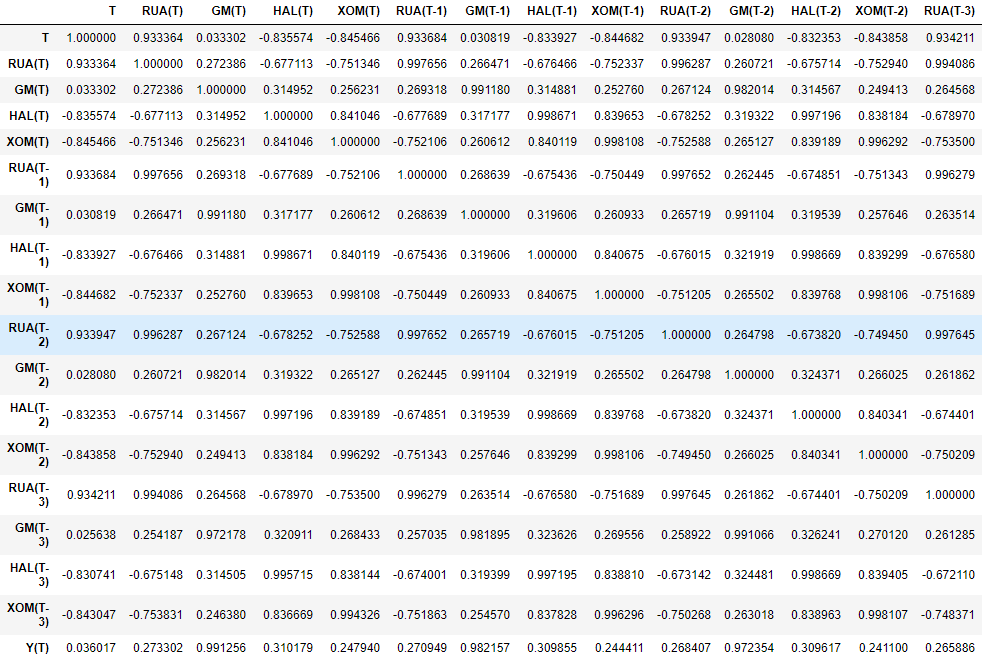


Figure 1: Features

2



Russell 3000 index is highly negatively correlated with Exxon mobil stocks for all the days. So as the Russell 3000 index increases the Exxon mobil stock is likely to decrease. Halliburton Stock is highly positively correlated with Exxon mobil stock and negatively correlated with Russell 3000 index. Exxon mobil stock is also negatively correlated with the Russell 3000 index. As Exxon Mobil, and Halliburton stocks increase, the Russell 3000 index tends to decrease.

T 0.036017

RUA(T) 0.273302

GM(T) 0.991256

HAL(T) 0.310179

XOM(T) 0.247940

RUA(T-1) 0.270949

GM(T-1) 0.982157

HAL(T-1) 0.309855

XOM(T-1) 0.244411

RUA(T-2) 0.268407

GM(T-2) 0.972354

HAL(T-2) 0.309617

XOM(T-2) 0.241100

RUA(T-3) 0.265886

3

GM(T-3) 0.962489

HAL(T-3) 0.309541

XOM(T-3) 0.237944

Y(T) 1.000000

Our target variable Y(T) the Closing price of the General Motors stock on day T+1 has a high positive correlation with its own stock prices for the previous 3 days. Then it has low positive correlation with the Russell 3000 index, Halliburton Stock, and Exxon Mobil Stock.

**1.2 PCA**

We utilized PCA to compute the number of principal components for 95% explained variance. This will give us a lower bound for the size of the hidden layer in our MLP. Below is the plot for the number of components versus explained variance.

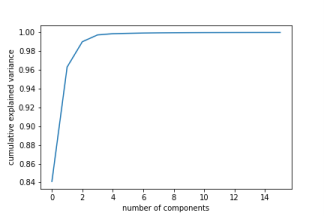


Figure 2: No. of Principal Components v. Cumulative Explained Variance

By principal component 2, we have reached 96% explained variance and therefore will choose h = 2 as the minimum for our h values.

**2 Multi Layer Perceptrons**

A multilayer perceptron is a feedforward artificial neural network. A MLP has Multiple layers and each layer can contain multiple neurons: the input to receive the signal, an output

layer to make a prediction about the input, and hidden layers in between. The first layer is called

4

the input layer, it contains the features in our training set so that each feature is represented by a neuron in the input layer. The final layer is called the output layer and it outputs a prediction for our response variable which could be a single value for regression tasks or an “n” dimensional vector for a classification task, where “n” is the number of classes. The layers in between are the hidden layers, there can be multiple hidden layers and each hidden layer can contain multiple nodes. Training involves adjusting the parameters the MLP takes, which includes the number of hidden layers and the nodes in each layer and also the weights and biases connecting the nodes in each layer. The aim of the MLP model is to minimize error. The MLP utilizes a supervised learning technique called backpropagation to change the weights and biases in order to minimize error.

**2.1 Training the MLP**

For our purposes we will use an MLP to predict the closing price of General Motors stock on day T+1. We will have an MLP with 3 layers: input, hidden layer, and output. We choose 4 different values for the size of our hidden layer to test. We use h = 2 from PCA as our minimum hidden layer size and hmax as 55 as 18 \* 55 + 1 = 991 *<*1003 = size of our training set. We arbitrarily chose h = 20, and h = 40 as our other two hidden layer sizes to test. For each of these h values we implemented automatic learning via Tensorflow with the adam gradient descent optimizer and used batch learning with a batch size of 25 and epochs = 8000. The batch size is a hyperparameter that defines the number of samples to work through before updating the internal model parameters.At the end of the batch, the predictions are compared to the expected output variables and an error is calculated. From this error, the update algorithm is used to improve the model.The number of epochs is a hyperparameter that defines the number times that the learning algorithm will work through the entire training dataset.One epoch means that each sample in the training dataset has had an opportunity to update the internal model parameters. An epoch is made of one or more batches. Additionally, our loss function is the MSE loss function. Below we share the results of the MLP with each h value:

For h = 2, below is the batch to batch evolution of the MSE and the norm of the gradient MSE/sqrt(# of parameters):

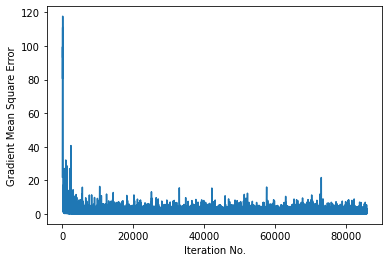
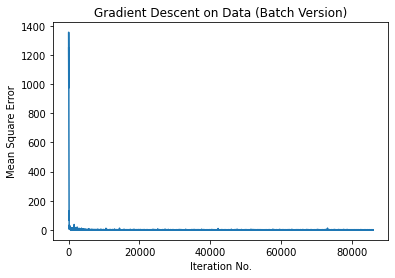


Figure 3: Batch to Batch MSE Evolution (h = 2)

We must specify the number of epochs for our learning algorithm. However, there is no

5

clear rule of how to choose the best parameters. We must try different numbers. So, we created this plot with the number of epochs on the x axis and the Terminal RMSE in the y axis to find the best number of epochs. The orange values represent the terminal RMSE on the testing set, whereas the blue is the terminal RMSE on the training set. Using the hidden layers with 2 neurons, the Terminal RMSE decreases very rapidly at the beginning until about 30 epochs, it then continues a constant descent until around 400 epochs. At 400 epochs we noticed that the Terminal RMSE of the training set decreases at a faster rate then the Terminal RMSE of the test set. The descent of both slows further down but it continues decreasing until around 1800 epochs where we start to notice it stabilizing. After 4000 eochs Training Terminal RMSE continues to go down slightly and the testing Terminal RMSE seems to be increasing which is an indication that we might be overfitting.

Now, the trainMSE and testMSE of the MLP:

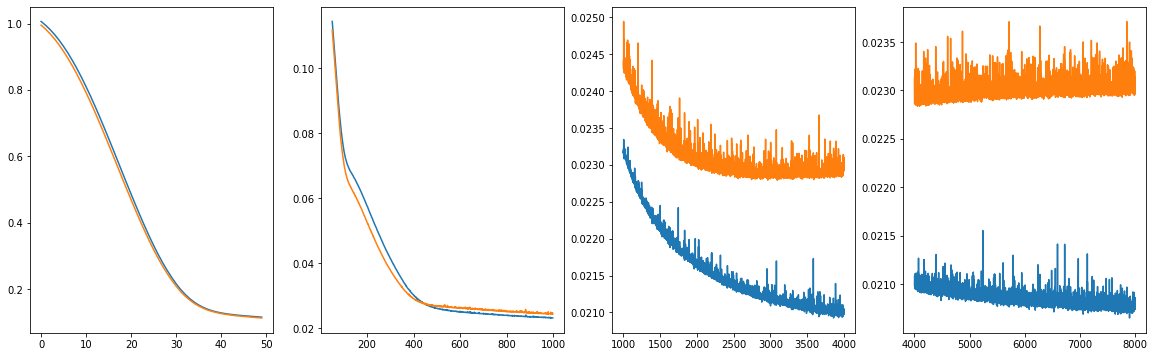


Figure 4: trainMSE and testMSE (h = 2)

The terminal RMSE function we used in the root of the loss function divided by the average of our prediction stock. The loss is a summation of the errors made for each example in training or validation sets, called the mean of squared errors. The lower the loss, the better the model is working. The loss is calculated on training and validation and it tells us how well the model is doing for these two sets

For h = 20, below is the batch to batch evolution of the MSE and the norm of the gradient MSE/sqrt(# of parameters):

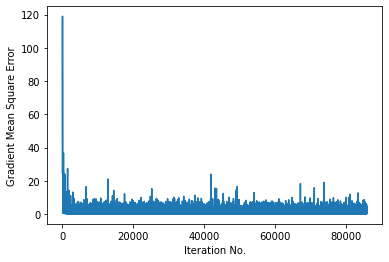
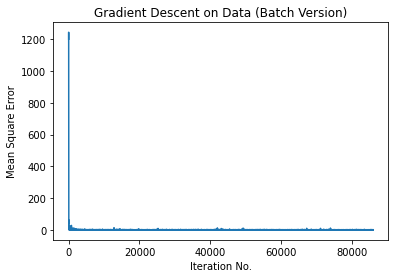


Figure 5: Batch to Batch MSE Evolution (h = 20)

6

Now, the trainMSE and testMSE of the MLP:

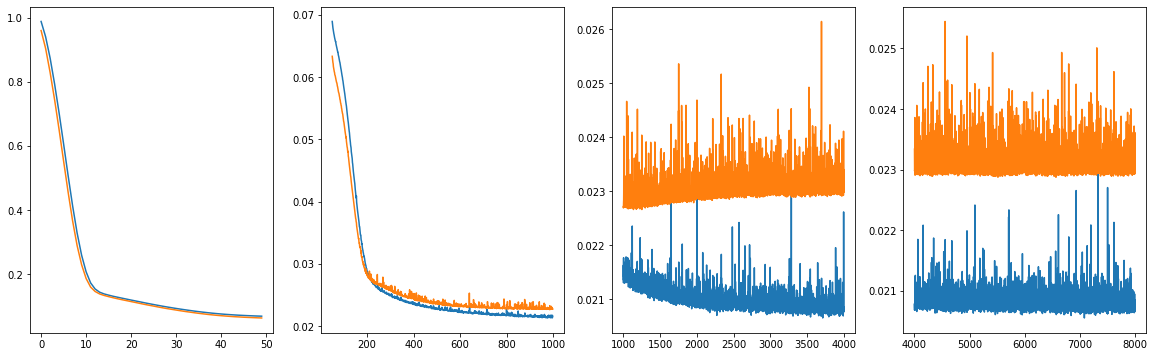


Figure 6: trainMSE and testMSE (h = 20)

Above we did twenty hidden neurons and plotted the epochs versus mean squared error. As shown in the plot above both the training set and test set performed similarly as with h2. Using the hidden layers with 20 neurons, the Terminal RMSE decreases very rapidly at the beginning until about 10 epochs, it then continues a constant descent until around 200 epochs. At 200 epochs we noticed that the Terminal RMSE of the training set decreases at a faster rate then the Terminal RMSE of the test set. The descent of both slows further down but it continues decreasing until around 900 epochs where we start to notice it stabilizing. After 1000 eochs Training Terminal RMSE continues to go down slightly and the testing Terminal RMSE seems to be increasing which is an indication that we might be overfitting. After 4000 epochs both seem to be stabilized once again, we would stop training at around 1000 epoch as it seems thats where our terminal RMSE is at a minimum.

For h = 40, below is the batch to batch evolution of the MSE and the norm of the gradient MSE/sqrt(# of parameters):

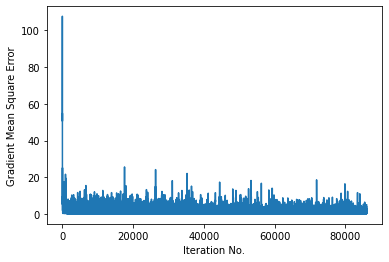
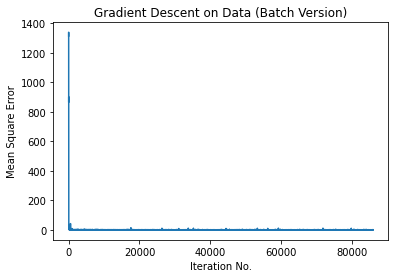


Figure 7: Batch to Batch MSE Evolution (h = 40)

Now, the trainMSE and testMSE of the MLP:

7

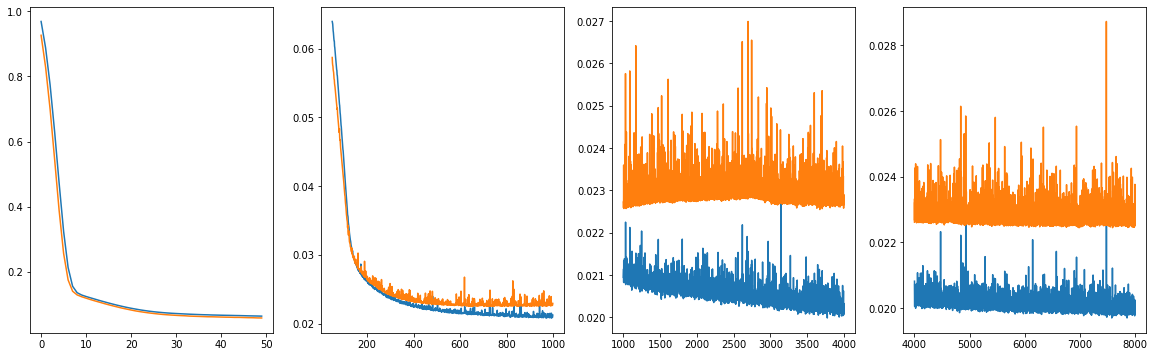


Figure 8: trainMSE and testMSE (h = 40)

When our neural network had 40 neurons in the hidden layer the Terminal RMSE decreases very rapidly at the beginning until about 10 epochs, it then continues a constant descent until around 160 epochs. At 200 epochs we noticed that the Terminal RMSE of the training set decreases at a faster rate then the Terminal RMSE of the test set. The descent of both slows further down but it continues decreasing until around 400 epochs where we start to notice it stabilizing. After 1000 eochs Training Terminal RMSE continues to go down slightly and the testing Terminal RMSE seems to be increase a little and then come back down. This could mean for a short amount of epochs our model was overfitting. After 400 epochs both seem to stabilize without changing much.

For h = 55, below is the batch to batch evolution of the MSE and the norm of the gradient MSE/sqrt(# of parameters):

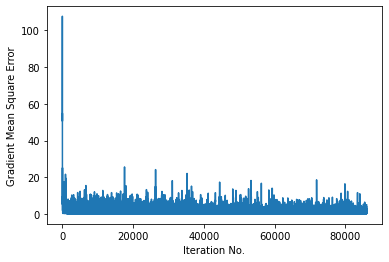
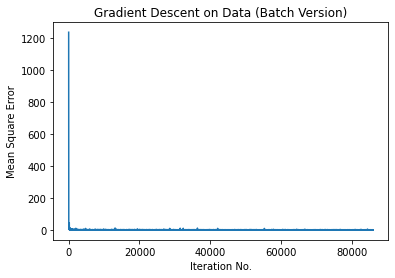


Figure 9: Batch to Batch MSE Evolution (h = 55)

Now, the trainMSE and testMSE of the MLP:

8

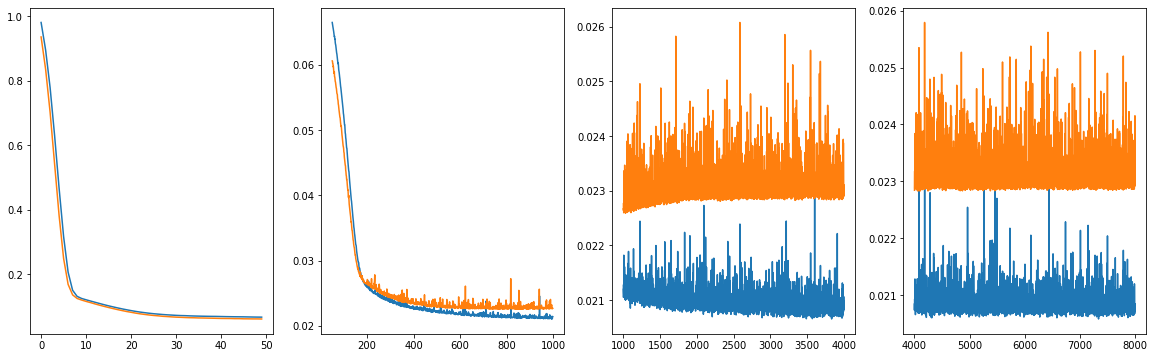


Figure 10: trainMSE and testMSE (h = 55)

When our neural network has 55 neurons in the hidden layer the Terminal RMSE decreases very rapidly at the beginning until about 8 epochs, it then continues a constant descent until around 180 epochs. At this point we also noticed that the Terminal RMSE of the training set decreases at a faster rate then the Terminal RMSE of the test set. The descent of both slows further down but it continues decreasing until around 1000 epochs where we start to notice it stabilizing. After 1000 epochs Training Terminal RMSE continues to go down slightly but the testing Terminal RMSE seems to be increasing which is an indication that we might be overfitting. Both stabilize after 3500 epochs though at there seems to be no change, we would stop training at around 1000 epochs as that's where our Terminal RMSE was the lowest.

**2.2 Finding Best MLP**

Ideally we would like to have tried several different combinations of layers and nodes as well as different cost functions and activation functions to choose the best MLP from several different permutations, however given the limitations of time we chose to stick with 3 hidden layers and only tried 4 different combinations for number of neurons. We then compared the performances of these 4 MLPs with different numbers of neurons using MSE and Terminal RMSE to choose the best MLP. The table below documents our findings. The average value of our target stock was 35.33134

|  |  |  |
| --- | --- | --- |
| h | MSE | Terminal RMSE |
| 2 | .6933 | 0.0235 |
| 20 | .6377 | 0.0226 |
| 40 | .6123 | 0.0221 |
| 55 | .6440 | 0.0227 |

Table 1: Minimum Value Loss for MLPs

9

The smallest minimum value loss was for the h = 40 MLP, so we will choose this as our best MLP.

**2.2.1 Neuron Activity of Best MLP**

To better understand this model, we will look at the average neuron activity for each neuron in the hidden layer of our best MLP. Below is a graph of the average neuron activity ordered from least active to most active:

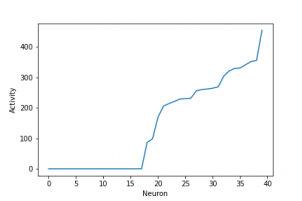


Figure 11: Neuron Activity for Hidden Layer

We also look at the 3 most active neurons and 3 most inactive neurons:

|  |  |
| --- | --- |
| Activity of 3 Most Active Neurons | Activity of 3 Most Inactive Neurons |
| 485.7745 | 0 |
| 474.0765 | 0 |
| 447.3477 | 0 |

Table 2: Top 3 Most Active and Most Inactive Neurons

10

The most active neuron of activity 485.7745 is located at index 32. The most inactive neuron of activity 0 is located at index 0.

In an artificial neural network, each neuron in a layer and is connected to each neuron in the next layer. When the inputs are transmitted between neurons, the weights are applied to the inputs along with the bias. Weights are one of the parameters that control the strength of the connection between two neurons. If the weight from node 1 to node 2 has greater magnitude, it means that neuron 1 has greater influence over neuron 2. A weight brings down the importance of the input value. Weights near zero means changing this input will not change the output. Negative weights mean increasing

this input will decrease the output. A weight decides how much influence the input will have on the output.

Below we list the weights for the most active neuron and the corresponding feature in ascending order:

|  |  |
| --- | --- |
| Feature | Weight |
| RUA(T-1) | -.6584 |
| GM(T-1) | -.431 |
| HAL(T-1) | -.3417 |
| XOM(T-3) | -.0452 |
| GM(T-3) | .1167 |
| RUA(T-3) | .1764 |
| XOM(T-1) | .1962 |
| HAL(T-2) | .2349 |
| HAL(T-3) | .3409 |
| XOM(T-2) | .3573 |
| RUA(T-2) | .3707 |
| GM(T-2) | .373 |
| RUA(T) | .3883 |
| GM(T) | .7487 |
| XOM(T) | .7514 |
| HAL(T) | .7654 |

Table 3: Weights and Corresponding Feature of Most Active Neuron

11

Here we have our weights for the most active neuron, the higher the weights the more influence we expect that particular feature to have on the price of our target stock General Motors(GM)’s price on T+1 days. Based on the table, the price of the stock of General Motors(GM), Exxon Mobile(XOM) and Haliburton(HAL) on any particular day has the most impact on the price of GM the next day.

Below we list the weights for the most inactive neuron and the corresponding feature in ascending order:

|  |  |
| --- | --- |
| Feature | Weight |
| GM(T) | -.0699 |
| XOM(T) | -.0400 |
| XOM(T-2) | -.0378 |
| HAL(T-2) | -.0232 |
| HAL(T) | .1001 |
| RUA(T-3) | .206 |
| RUA(T-2) | .2263 |
| RUA(T) | .3256 |
| XOM(T-3) | .4066 |
| HAL(T-3) | .4088 |
| GM(T-3) | .4359 |
| RUA(T-1) | .5898 |
| XOM(T-1) | .6439 |
| GM(T-1) | .699 |
| HAL(T-1) | .7123 |
| HAL(T) | .7654 |

Table 4: Weights and Corresponding Feature of Most Inactive Neuron

Here we have our weights for the most active neuron, the higher the weights the

12

more influence we expect that particular feature to have on the price of our target stock General Motors(GM)’s price on T+1 days. Based on the table, the price of the stock of General Motors(GM), Exxon Mobile(XOM) and Haliburton(HAL) on day (T-1) has the most impact on the price of GM the next day however since the activation of the neuron is low these weights will not have a large impact on our prediction.

Finally we would like to plot our predictions(in yellow) with our actual values(in blue), for this purpose we plotted our prediction with the actual results from our test set. We notice that our model does reasonably well at predictions however with the stock market where even the slightest change in price can have a huge impact, we would have liked access to better visualization tools to conduct our analysis. Once again due to the time limitations we had to settle for this.

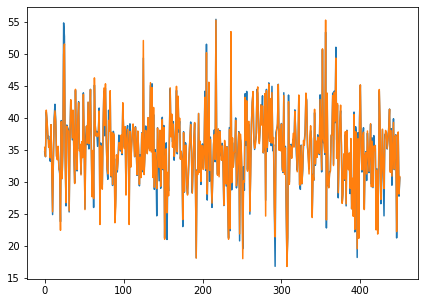


Figure 12: Stock Price Prediction v. Actual for MLP h = 40

3 Conclusion and Further Suggestions

While it is hard to pick a set way to improve our model we can suggest some ways that might improve the performance. We would have liked to try these out but were unable to do so due to limitations of time. We will divide the improvements into two subclasses , Accuracy and Efficiency. The Accuracy will suggest improvements that could be made to the model to improve the predictions for our target stock. Meanwhile Efficiency will deal with the time it takes to execute our code. Note that we understand that due to the “No Free Lunch Theorem” we could try different regression models and stumble upon one that might work better, we will base our improvements assuming we have to use the current model. Note also that we do not intend our suggestions to be comprehensive, but hope that these will improve the performance from what we have gotten.

To improve the accuracy we would have liked to try different combinations for our

13

MLP, we would have liked to experiment with more layers and different numbers of neurons in each layer to make our model deeper as opposed to wider. We could also try different feature selection models to improve the quality of our data after we got the weights at the end and we could pair this with different Weight initializations to improve our models further

Another thing that we would like to attempt to do would be to try and measure the results using different learning rates as well as different activation functions. We would also like to experiment with different batch sizes and number of epochs, we recognize that the norm for modern deep learning implementations is small batch sizes and large numbers of epochs however we would have liked to see if our data strays from the norm. We would also have liked to attempt regularization techniques such as dropout or drop connect and see how they improve our model. Similarly we could also use different optimizers other than ADAM such as SGD or RMSprop. Finally we can do something as basic as obtaining more data, one of the advantages that deep learning has over the more traditional models is that with more data its performance continues to improve, we could do this by obtaining more stock data from various sources. We would however have to be careful as with the volatility of the stock market, the old data can often become redundant.

As suggested before we could improve efficiency by using a fewer number of epochs and an even smaller batch size as long as that does not impact our accuracy, furthermore a faster learning rate could also help speed up the computation of our algorithm. Finally we could implement early stopping to prevent our algorithm from continuing to run needlessly.

All in all it is very hard to pinpoint exactly what needs to be done, we could add different stocks to our data set that would increase performance and we just have to try different combinations of number of layers and number of neurons in each layer and different optimizers as well as different learning rates, batch size and number of epochs etc. While this process would be extremely cumbersome we can hope that this will improve the overall performance of our model. A value judgement will need to be made though to figure out if the time spent on these experiments would be worth the improvements in the accuracy and efficiency of our model.

We tried to tweak some of the variables in an attempt to improve our performance, our findings are given in the table below:

14

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Index | Optimizer | num hidden | h of L1 | h of L2 | h of L3 | Epochs | Batch | Best Val Loss |
| 1 | ADAM | 2 | 40 | 20 | NA | 150 | 25 | 0.729278 |
| 2 | ADAM | 3 | 40 | 20 | 10 | 150 | 25 | 0.572377 |
| 3 | ADAM | 3 | 40 | 20 | 10 | 100 | 25 | 0.829467 |
| 4 | ADAM | 2 | 40 | 20 | NA | 100 | 25 | 0.794408 |
| 5 | RMSprop | 2 | 40 | 20 | NA | 100 | 25 | 0.708231 |
| 6 | RMSprop | 2 | 40 | 20 | NA | 150 | 25 | 0.589567 |
| 7 | RMSprop | 2 | 40 | 20 | NA | 150 | 30 | 0.596094 |
| 8 | RMSprop | 3 | 40 | 20 | 10 | 150 | 30 | 0.63804 |
| 9 | Adamax | 3 | 40 | 20 | 10 | 150 | 30 | 0.672716 |
| 10 | Adamax | 2 | 40 | 20 | NA | 150 | 30 | 1.014071 |
| 11 | Adagrad | 2 | 40 | 20 | NA | 150 | 50 | 498.708 |
| 12 | Adagrad | 2 | 40 | 20 | NA | 150 | 30 | 30.92106 |
| 13 | SGD | 2 | 40 | 20 | NA | 150 | 25 | 1238.405 |
| 14 | ADAM | 2 | 40 | 20 | NA | 100 | 25 | 0.537429 |
| 15 | SGD | 2 | 40 | 20 | NA | 100 | 25 | 1238.405 |

Additionally we understand that in the case of the stock market it is perhaps more important to predict if the price of our stock would go up or down as opposed to trying to predict the price itself. Our models did not perform very well in this respect with h2 getting the accuracy of 0.52, h20 getting 0.50, h40 getting 0.50 and h55 getting the accuracy of 0.51. While the models were able to catch major price shifts, we would in the future want to convert our regression task into a classification task and try to exclusively predict whether the price will go up or down.

15