Stock Prediction Using RNNs

Andrei Neagu

ID: 40109412

Abstract

The goal of this project was to see if it possible to train RNNs to predict stock market results. This was to be done using and comparing GRU and LSTM recurrent neural networks. The experiments were done on Apple's stock and using the VIX and Nasdasq indices to perform the prediction. The experiments could not reliably and profitably predict stock gains for Apple.

1. Introduction

The goal of this project is to see if it is possible to predict stock gains from one day to the next using Recurrent Neural Networks. The data fed into the RNN consists of the Open, High, Low, Close, Adjusted Close, and Volume for a stock, the Vix index and the Nasdaq index. The Vix is a volatility index for the stock market and the Nasdaq index is an index that tracks the aggregate of stocks listed on the Nasdaq stock exchange. Ideally the stock we want to predict is listed on the Nasdaq stock exchange. The predictions are based on the difference between the close of the current day and the next, so we want to predict how much lower or higher the stock is going to close the next day. A secondary goal of the project was to see which model was better at making predictions between GRU and LSTM RNNs.

2. Methodology

2.1 Data Preprocessing

I downloaded data from yahoo finance using the yfinance library. This data is returned as a pandas dataframe which I then split into the features (X) set and the values to predict set (y).

The data is then mean normalized since the range of the values varies a lot for each feature. For example, the volume, which is the number of shares traded in a certain period, is in the order of hundreds of thousands or even millions but the closing price of a stock can be a few dollars only. Mean normalization is applied before the sequence creation and the train/test split of the dataset. This is because it is easier to normalize the data before it is turned into sequences. Mean normalization is necessary for the gradient descent to converge to a solution more efficiently when training an RNN.

After normalizing the data, we create sequences of N days to pass to the RNN. The sequences are created by passing a rolling window of N days over the whole dataset of features. For example, if we had 3000 x 17 features and to make predictions using the past 30 days to predict the next day's gain, we would end up with data of shape 2971 x 30 x 17. The first 29 days would be discarded since they wouldn't have full sequences of 30 days.

After creating the sequences, we split the data into training, validation, and test sets. In the code, the default values are 80% of the dataset will be used for the training set and 10% of the training set will be used for validation. The remaining 20% of the dataset will be used for testing.

The sets are then cast to tensors and sent to the GPU if there is one. The code should be run on a device that has a GPU or a service like Kaggle that provides online GPU computation.

2.2 RNN training

I used PyTorch to train the RNN because that is what we used in class, so I was familiar with it. After trying to train the models on my laptop I realized that my resources were very limited, and I could only train very small neural networks. So, I decided to upload the notebook to Kaggle to use the free GPU Accelerator they offer. The RNN class and the training loops were inspired by Lab 5. The Recurrent Neural Network consists of a choice of using gated recurrent units or long short-term memory networks. The output of the RNN is then linearly transformed to get a prediction. Mean Squared error is used as the loss function since we are trying to solve

a regression problem because the predictions are percentages. Dropout is also introduced to avoid overfitting and get better generalization.

2.3 Hyperparameter testing

Hyperparameter testing is performed on the learning rate, the size of the hidden layers, and the number of layers. I only chose these three hyperparameters my computational power was limited and adding more hyperparameters would increase complexity exponentially. Additionally, they seem to influence the model the most. I also ran this hyperparameter search for both GRU and LSTM models to compare the best resulting RNN of each. To perform hyperparameter search, I created nested loops that train an RNN for each list passed. The values I used for the hyperparameters were: lr = [0.0005, 0.005, 0.05], hidden_size = [10, 20, 30], num_layers = [1, 2, 3]. Other hyperparameters to consider if I was to have more computational power would be: batch size, number of epochs, and dropout rate.

2.4 Originality

There are RNN models for stock prediction out there since stock prediction is a good test case for RNNs, but I have not seen RNNs that incorporate the VIX Index in the features. The most similar one I have seen uses an index of multiple stocks that are in the same sector to predict gains.

2.5 Reproducibility Issues

I encountered issues with the reproducibility of results using PyTorch, unfortunately I could not fix this issue in time because it was more complex than I expected. The results have a higher variability when training models with a high learning rate such as 0.05 and low variability when training models with lower training rates such as 0.0005.

3. Experimental Results

The model was tested on apple stock data from 1990-01-01 to 2022-01-01 and with sequences of length 30.

The results of the experiments were mixed. I ran hyperparameter search on the GRU model with the following parameters:

rnn_type = 'gru'

lr = [0.0005, 0.005, 0.05]

hidden_size = [10, 20, 30]

num_layers = [1, 2, 3]

and the result with the lowest mse on the validation set was the one with hyperparameters: {'lr': 0.05, 'hidden_size': 30, 'num_layers': 1}. Although this is the result with the lowest mse calculated (0.00022200), we can see in Appendix 1.1 that there is an issue in the training phase because the learning rate is too high, and the losses vary wildly. Additionally, the testing gain% predicted on the test set also vary significantly and go as far as predicting losses of 200% in one day as can be seen from Appendix 1.3. However, the model with those hyperparameters was still profitable when running a trading simulation as can be seen from Appendix 1.4. The simulation consists of starting with $200,000 and buying (or holding if already bought) the stock when the model predicts that the next day will be positive and selling (or doing nothing if the stock wasn't bought) when it predicts that the next day will be negative. But even thought it was profitable, it did not outperform just holding the stock for that time period. The hyperparameter search also was inconclusive in discerning a pattern as can be seen from Appendices 1.5-1.8. But this was on a very limited dataset because I did not have the computational resources needed to perform a broader hyperparameter search.

The best result for the lstm hyperparameters search turned out to be the one with the hyperparameters: {'lr': 0.05, 'hidden_size': 10, 'num_layers': 2}. This resulted in an mse of

0.00022677 which is higher than the best result obtained from the gru model. The learning rate was also too high for this model and the losses have very big swings as can be seen from Appendix 2.1. This time the model performed very poorly and predicted a constant negative gain for all days, so the model never predicted a buy. This can be seen from Appendix 2.3 and 2.4. Finding a pattern in hyperparameter search was also inconclusive (Appendices 2.5-2.8).

It would seem that GRU models do a better job than LSTM models of predicting stock market gains, but this result was concluded with results that weren't anywhere near satisfactory because of the amount of data that was limited by computational power and also the high learning rate of 0.05 in the hyperparameter search that skewed the results.

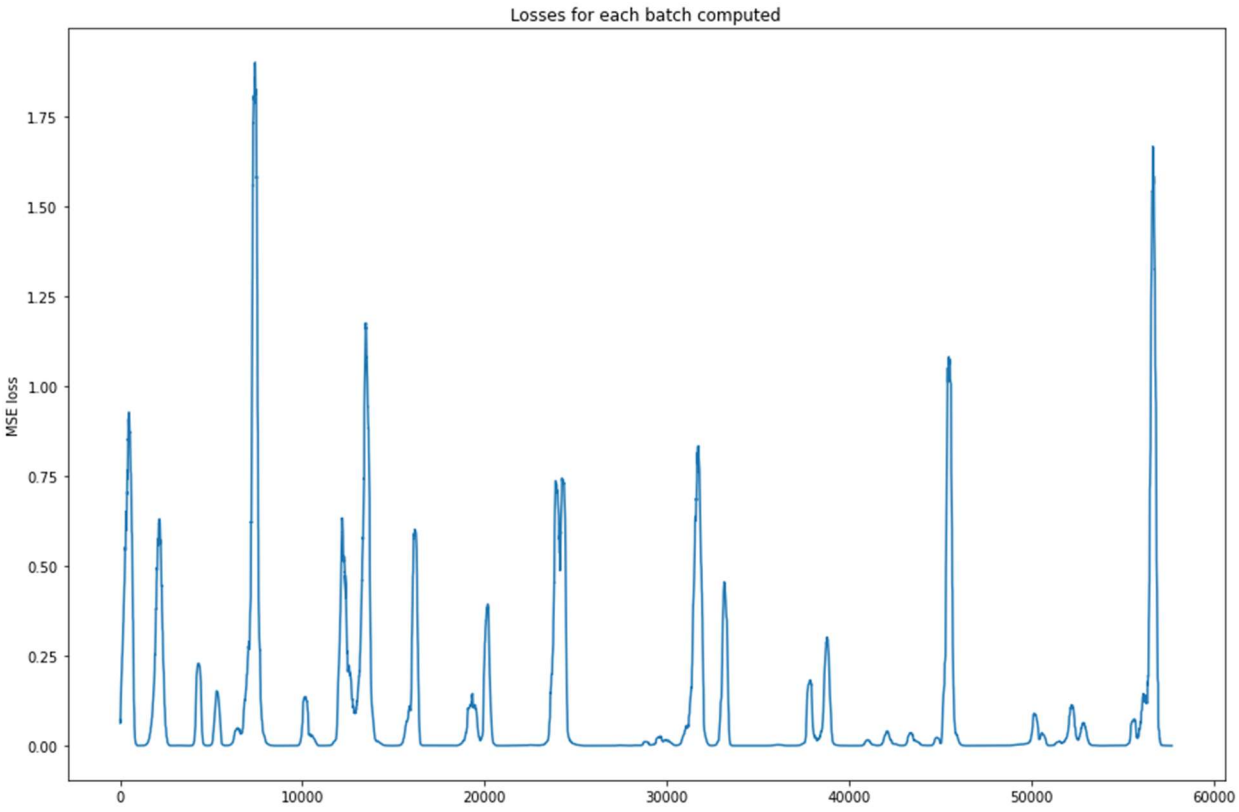
It would also seem that both GRU and LSTM RNNs fail at predicting stock market gains but I came across an outlier when experimenting with different hyperparameters and that is the gru model with the following hyperparameters: {'lr': 0.0005, 'hidden_size': 10, 'num_layers': 2} having an mse of 0.00090511. Although the model has a higher mse than the best models found by the hyperparameter search for gru models, in fact it is rated 24 out of 27, it managed to converge to a solution better than the best model found. This can be seen by the smooth losses curve from Appendix 3.1. It also outperformed just holding the stock and if a person traded on its predictions, they would end up with $1,595,027 versus $1,336,113 when just holding the stock. This can be seen from Appendix 3.4. Of course, this result was cherry picked and cannot be considered reliable because as we can see by the predictions on the validation set (Appendix 3.5) where holding nets you $391,204 and trading on the model's prediction nets you $202,334.

4. Conclusions

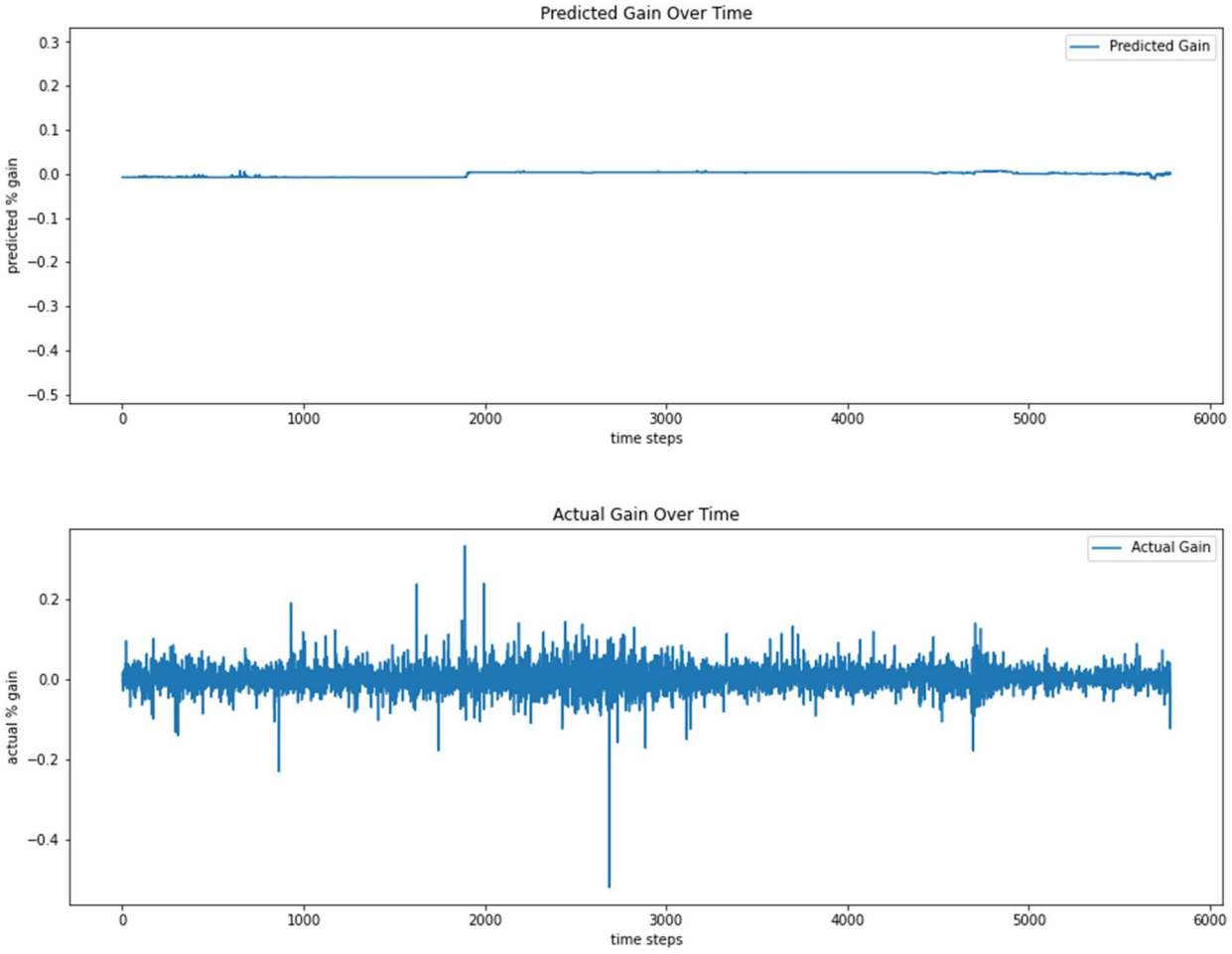
With the data I had and the experiments I performed; I could not reliably and profitably predict stock market gains but that does not mean that it is impossible. There are many factors why the model might have failed such as the hyperparameters I picked, lack of data and strategy used for the trading simulation. From my experiments, it looks as though GRU models do a better job than LSTM models at predicting stock market gains, but this was also limited by the hyperparameters.

In future experiments it would help to add more data or use different data and experiment using more hyperparameters because I was limited in my time and computational resources.

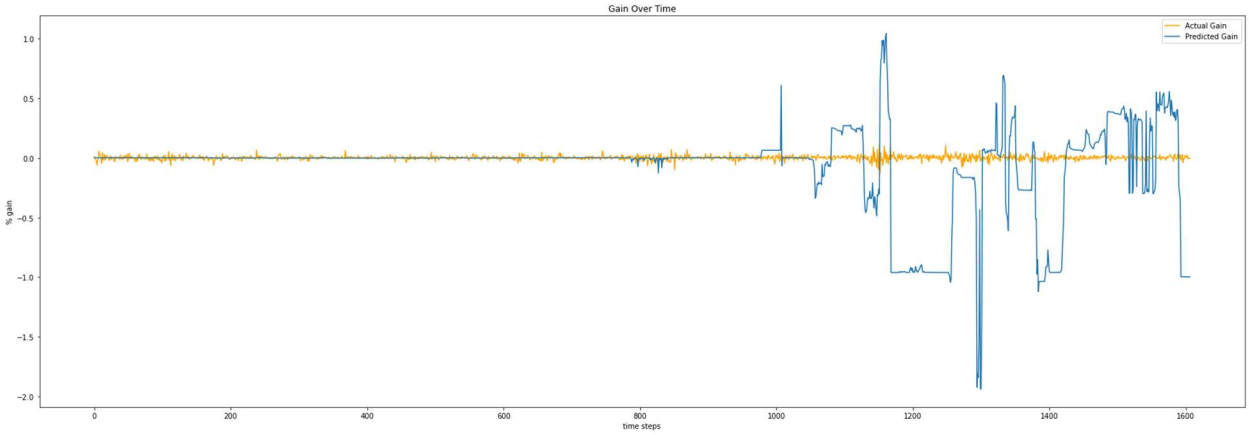
Appendices



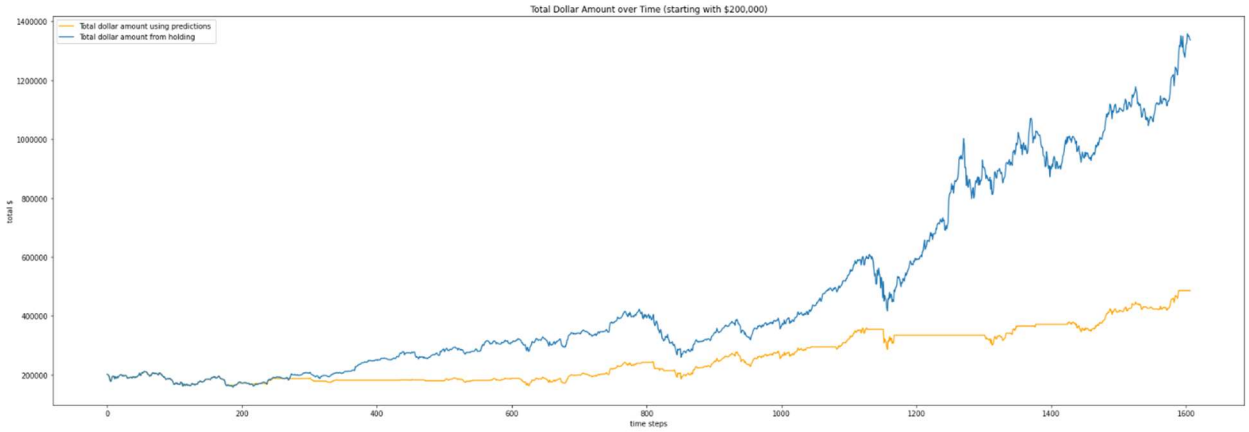
Appendix 1.1: Losses for each batch computed with GRU model (learning rate: 0.05, hidden size: 30, number of layers: 1)



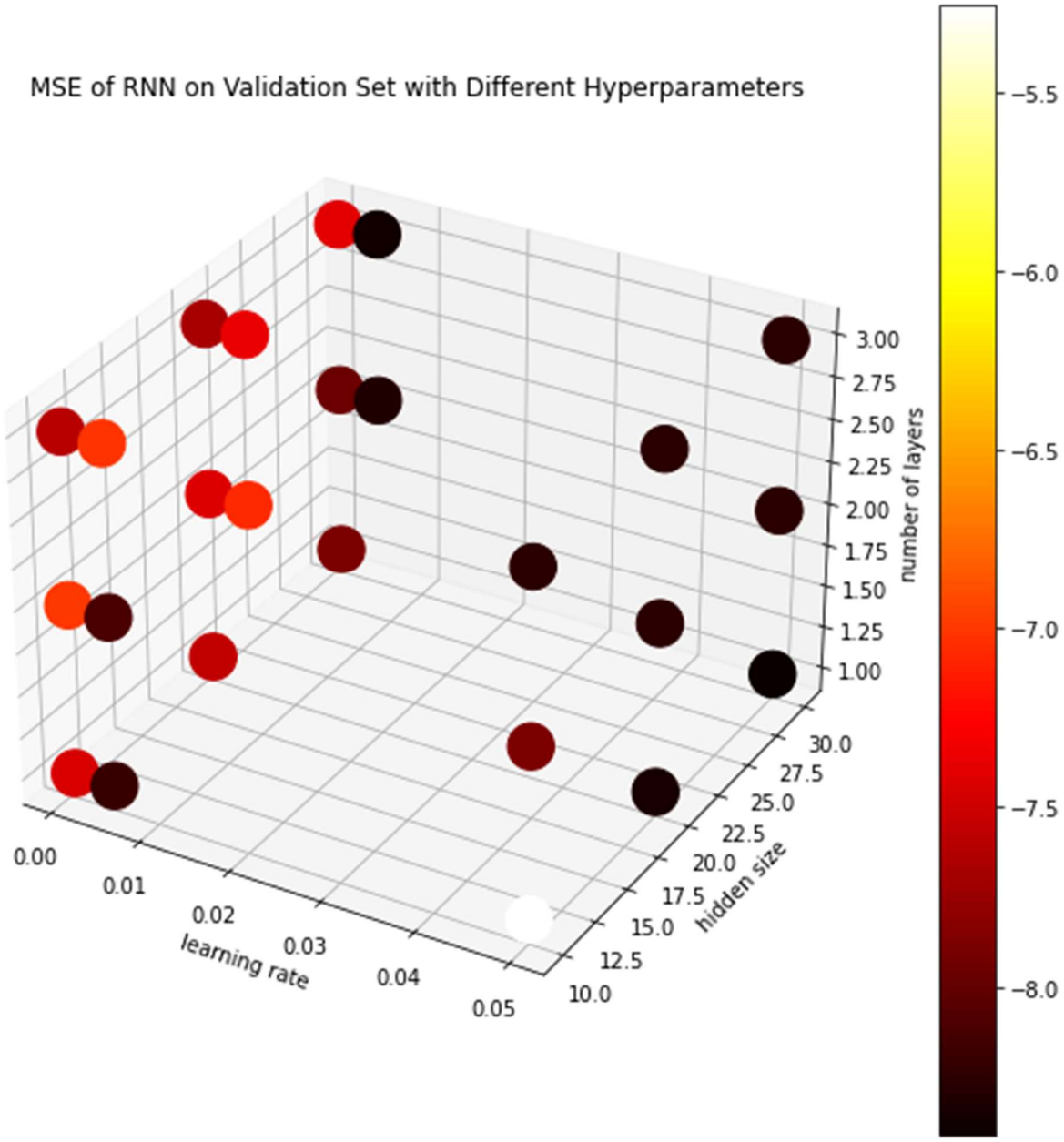
Appendix 1.2: Training %gain predictions vs actual values for GRU model (learning rate: 0.05, hidden size: 30, number of layers: 1)



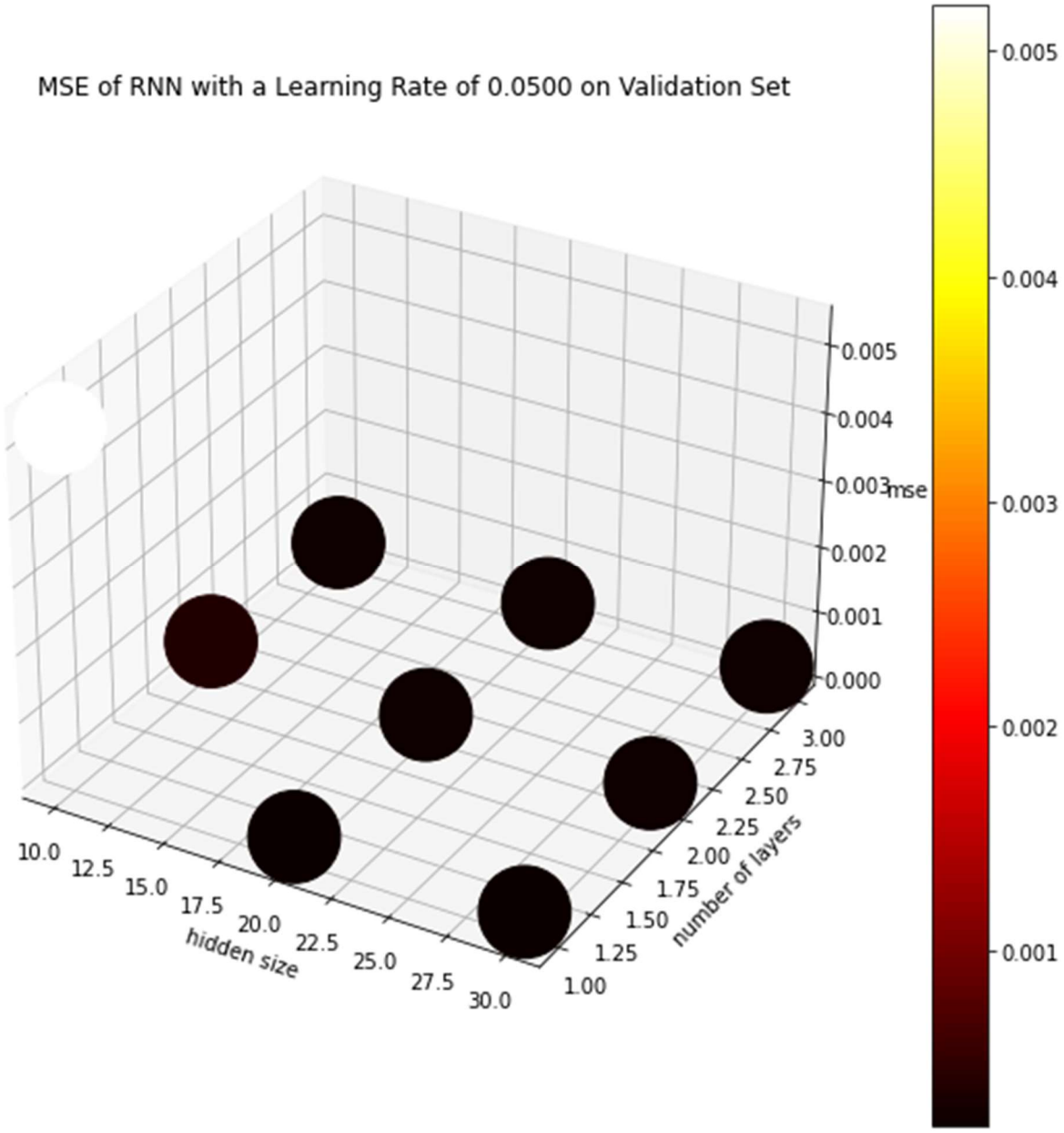
Appendix 1.3: Testing %gain predictions vs actual values for GRU model (learning rate: 0.05, hidden size: 30, number of layers: 1)



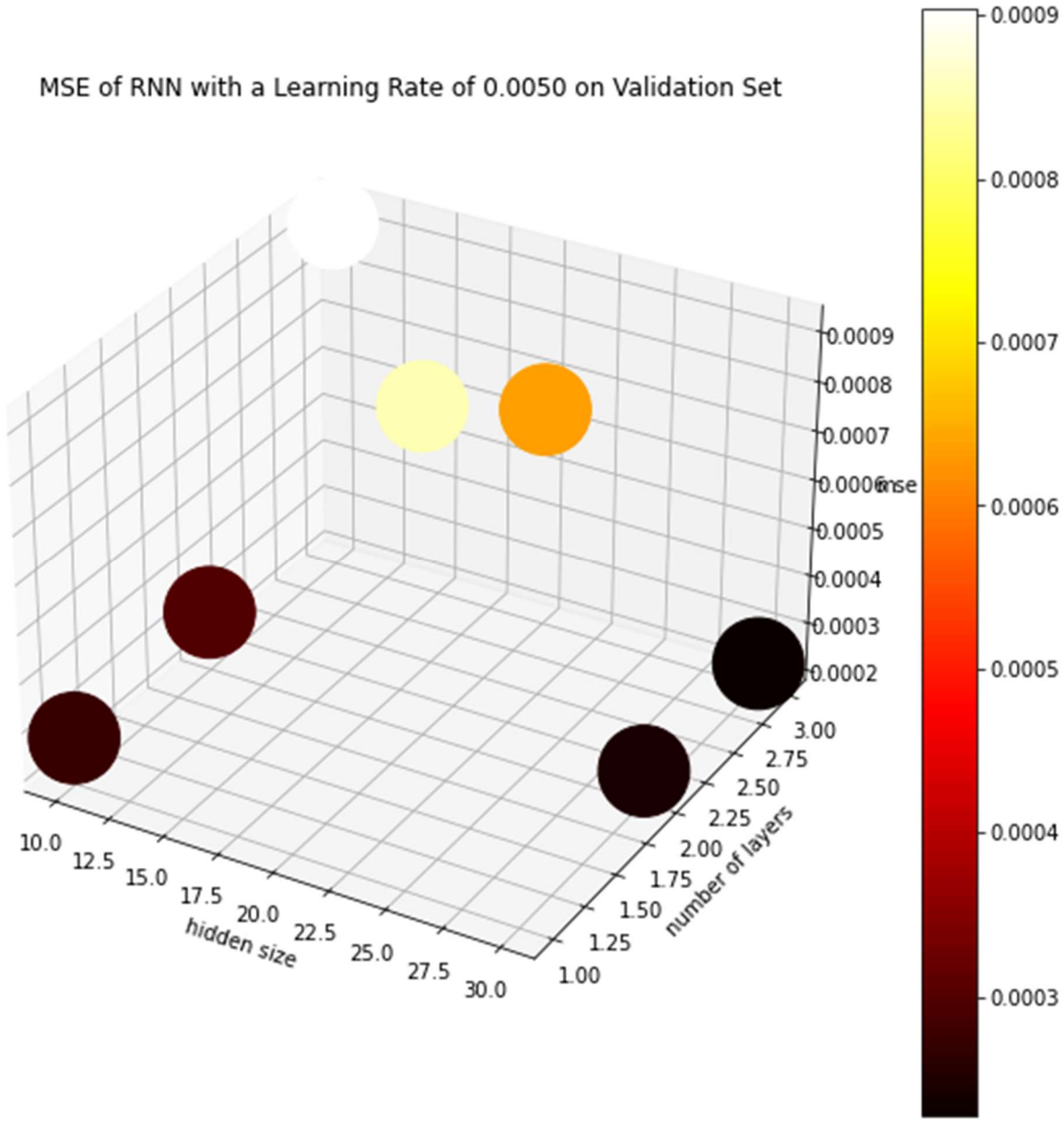
Appendix 1.4: Total Dollar Amount over Time for GRU model (learning rate: 0.05, hidden size: 30, number of layers: 1)



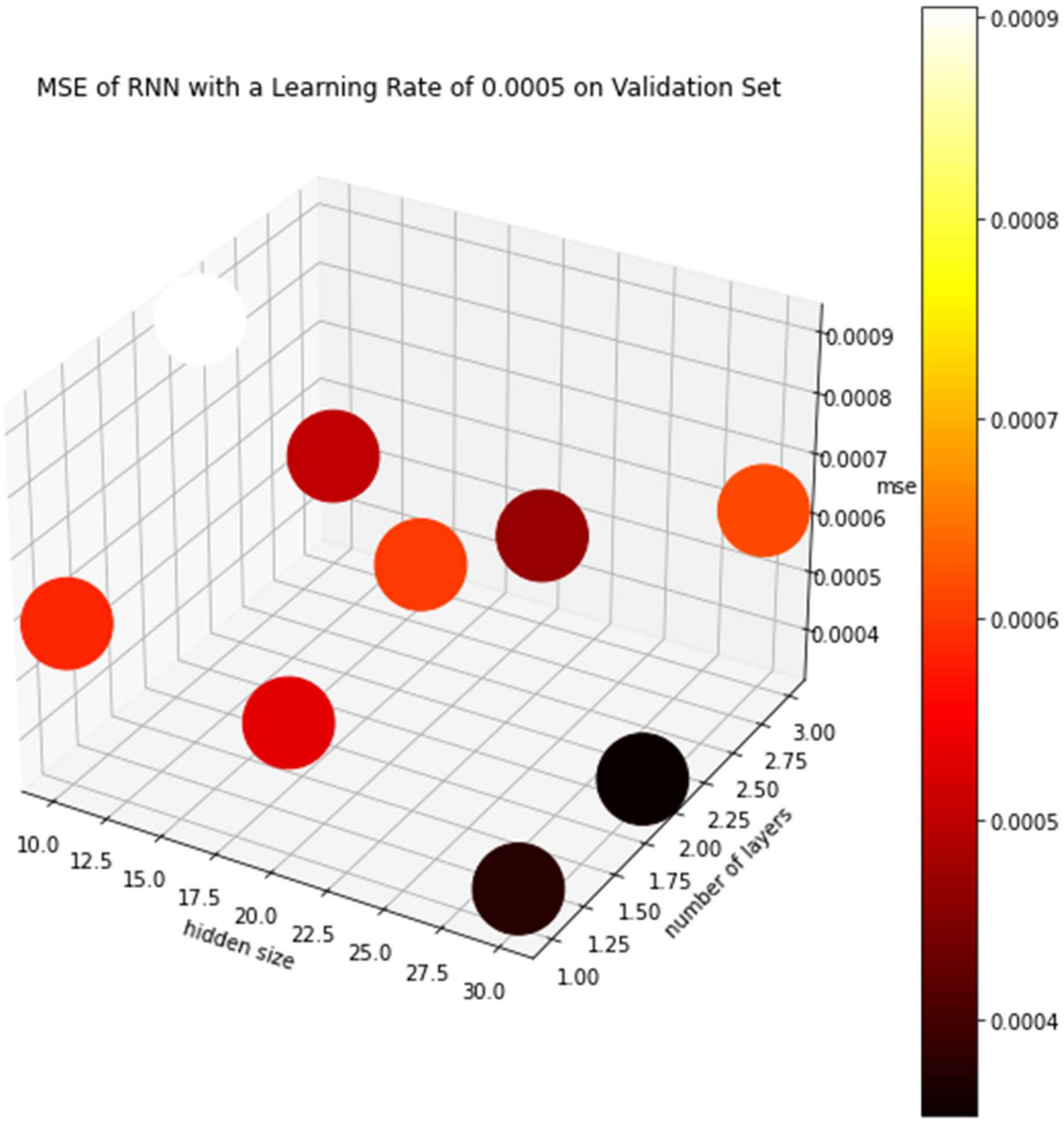
Appendix 1.5: Hyperparameter search results for GRU model



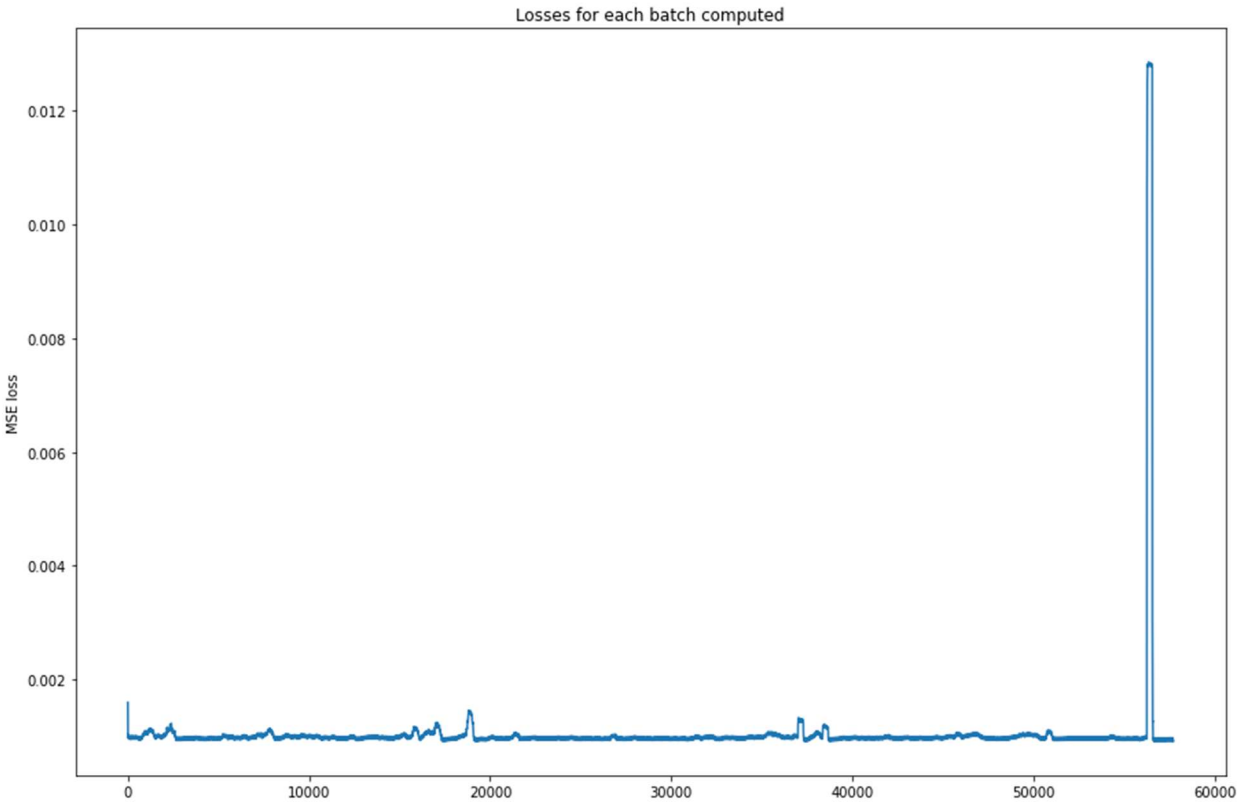
Appendix 1.6: Hyperparameter search results for GRU model with a learning rate of 0.05



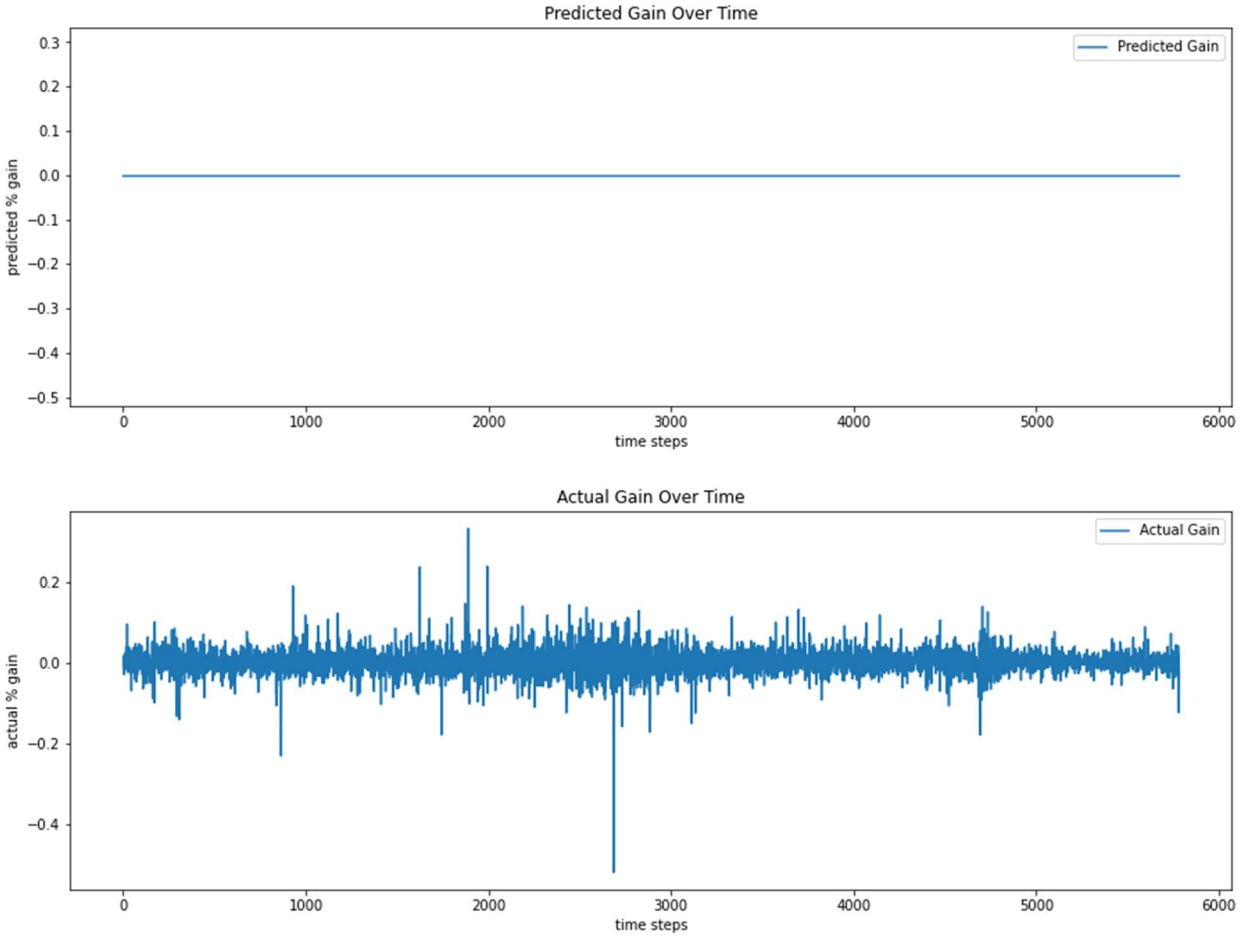
Appendix 1.7: Hyperparameter search results for GRU model with a learning rate of 0.005



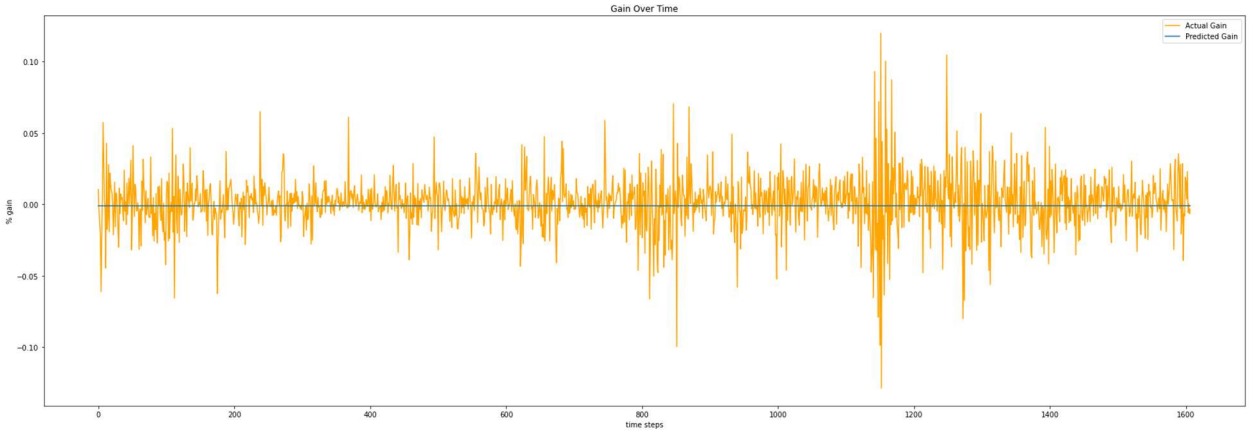
Appendix 1.8: Hyperparameter search results for GRU model with a learning rate of 0.0005



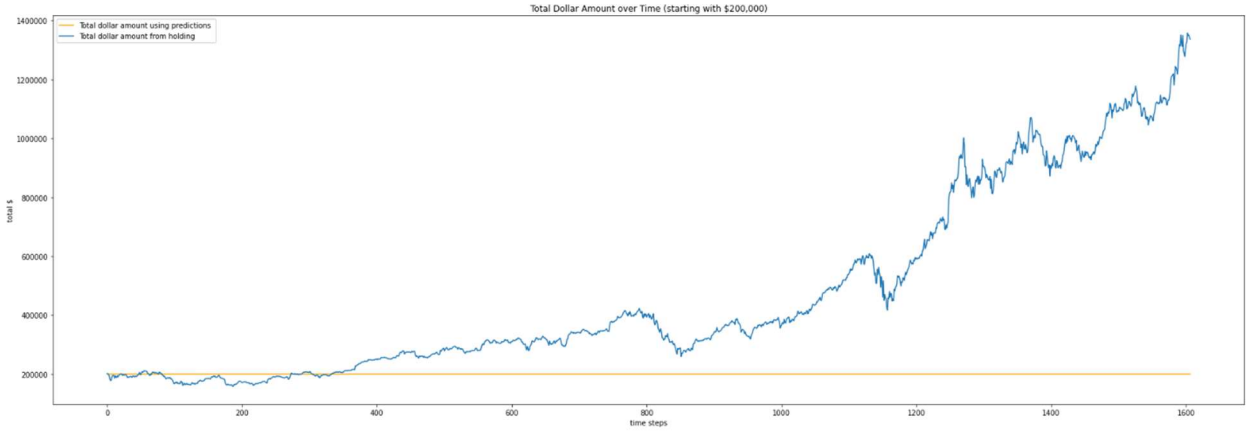
Appendix 2.1: Losses for each batch computed with LSTM model (learning rate: 0.05, hidden size: 10, number of layers: 2)



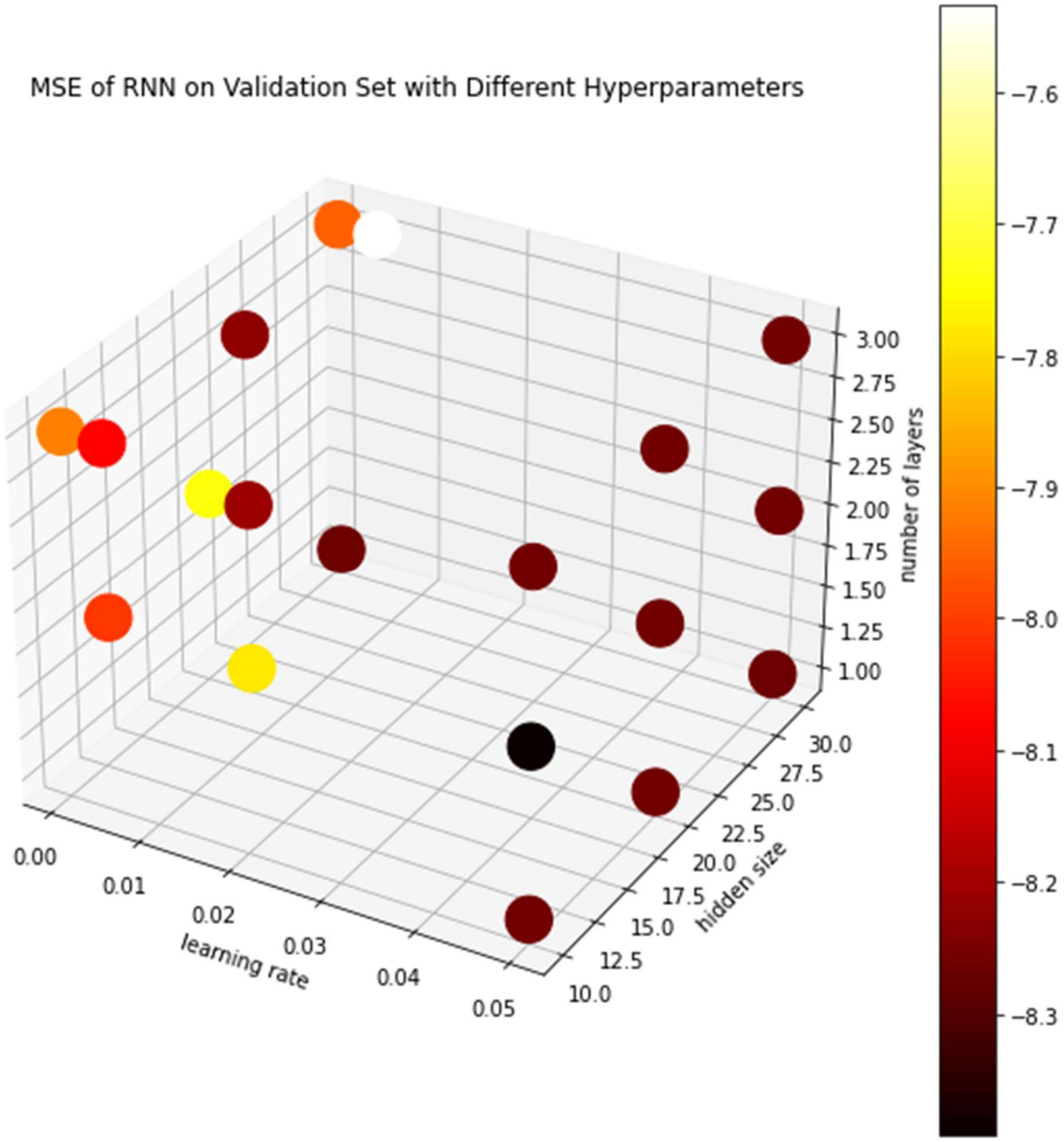
Appendix 2.2: Training %gain predictions vs actual values for LSTM model (learning rate: 0.05, hidden size: 10, number of layers: 2)



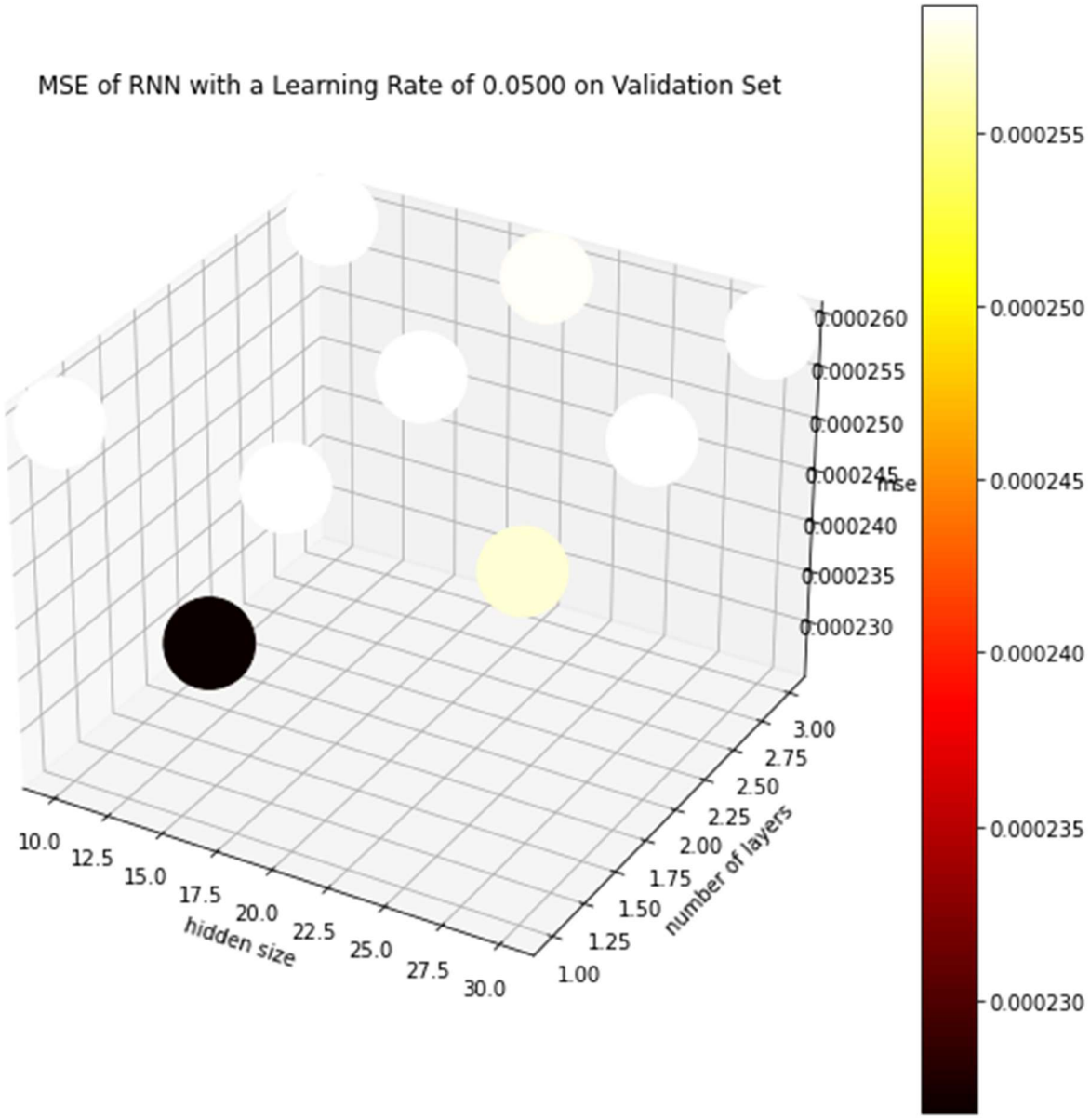
Appendix 2.3: Testing %gain predictions vs actual values for LSTM model (learning rate: 0.05, hidden size: 10, number of layers: 2)



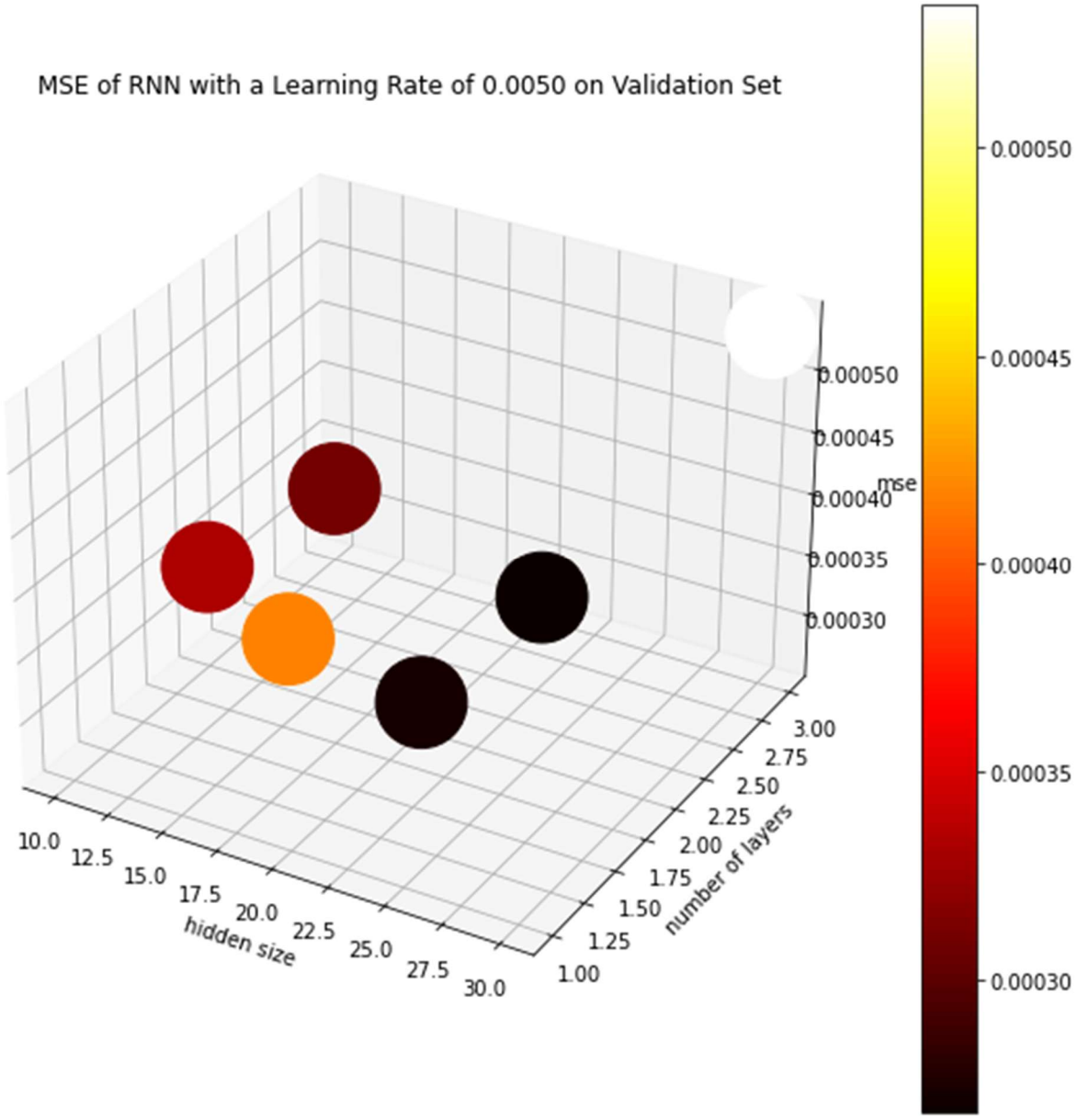
Appendix 2.4: Total Dollar Amount over Time for LSTM model (learning rate: 0.05, hidden size: 10, number of layers: 2)



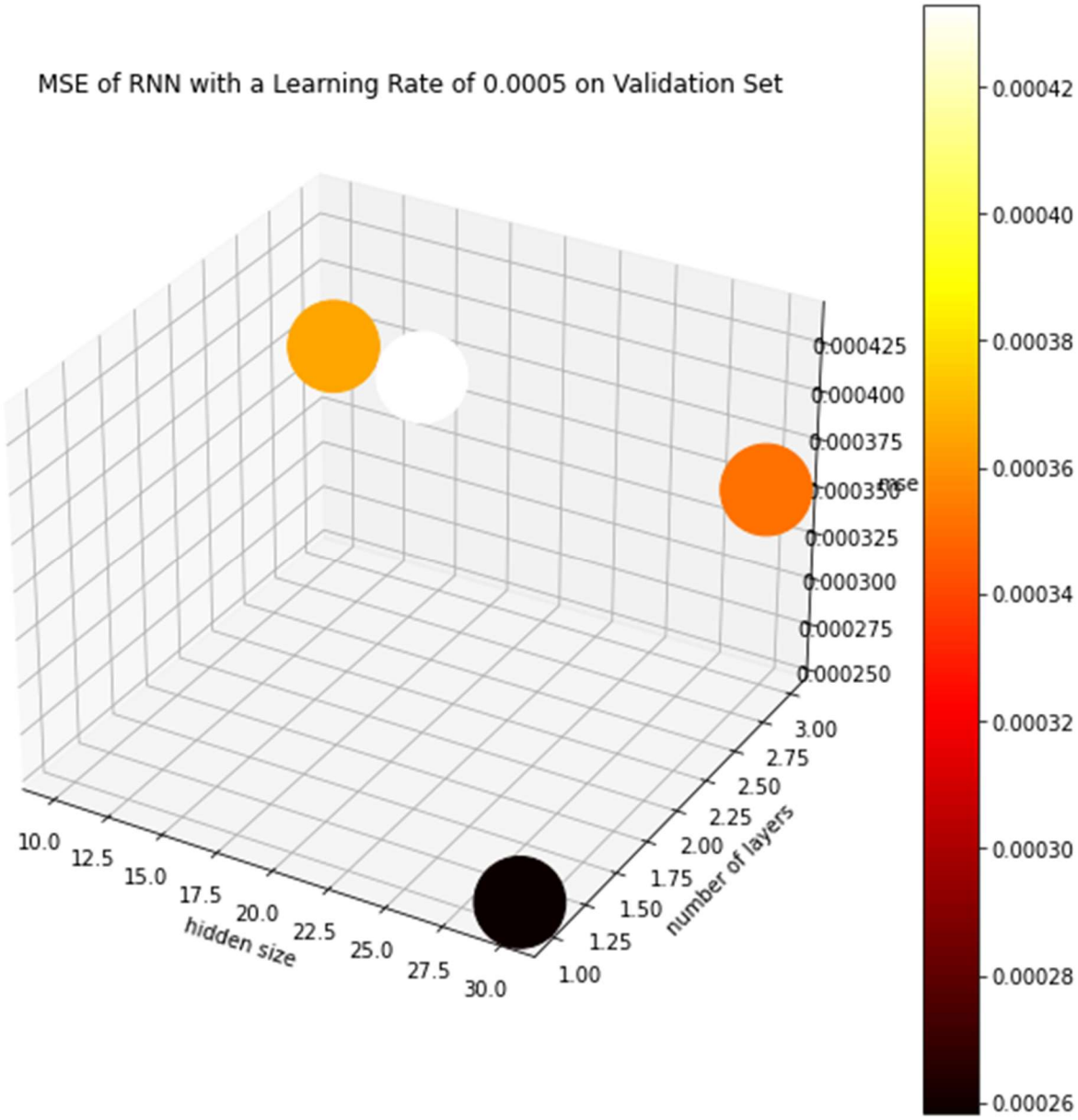
Appendix 2.5: Hyperparameter search results for GRU model



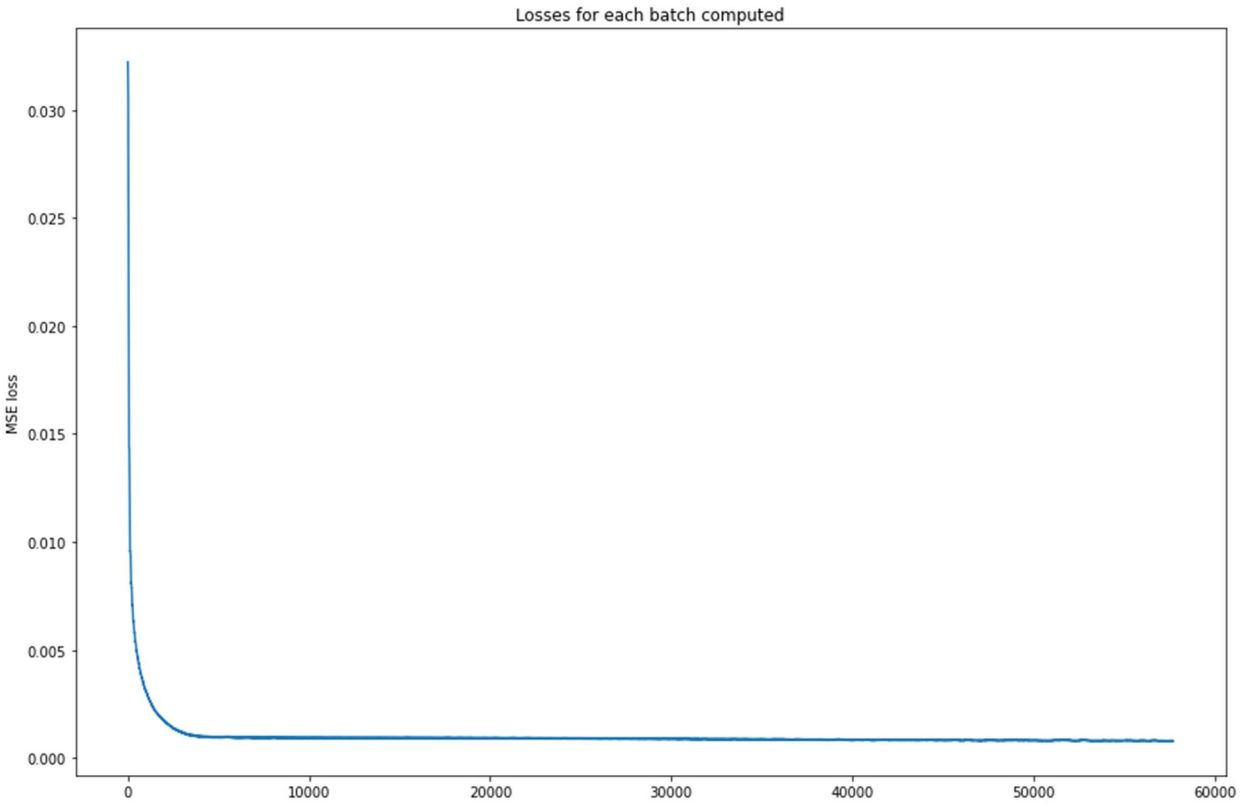
Appendix 2.6: Hyperparameter search results for LSTM model with a learning rate of 0.05



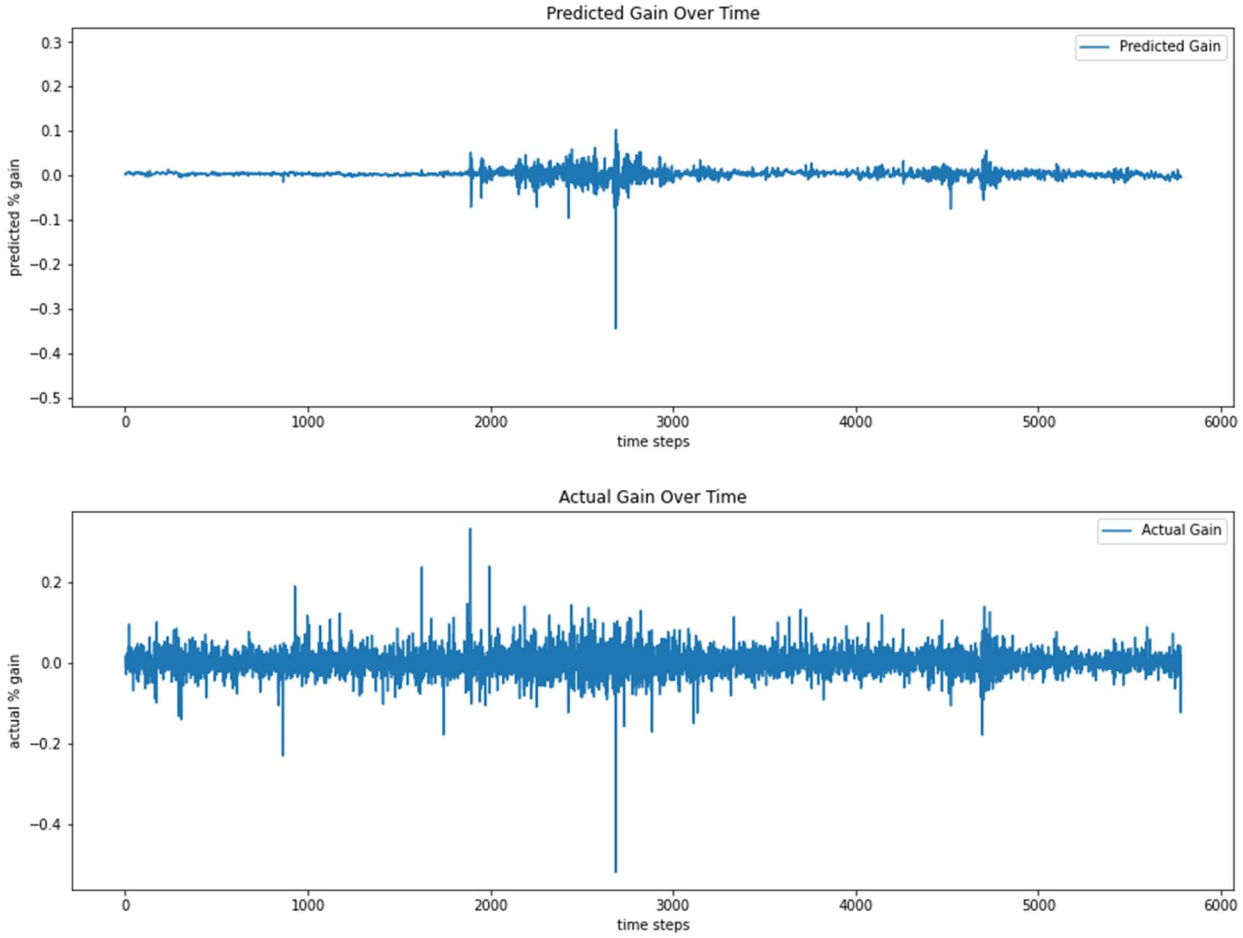
Appendix 2.7: Hyperparameter search results for LSTM model with a learning rate of 0.005



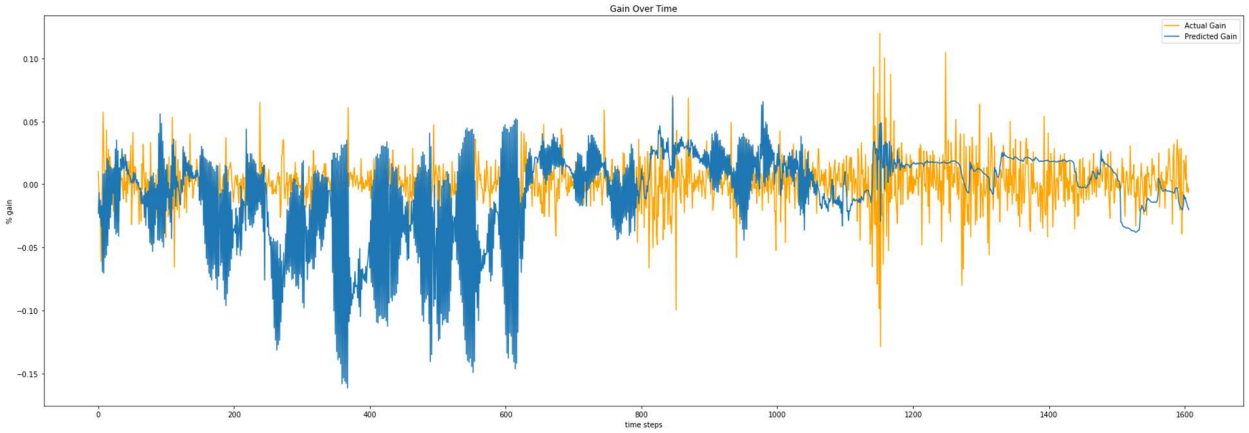
Appendix 2.8: Hyperparameter search results for LSTM model with a learning rate of 0.0005



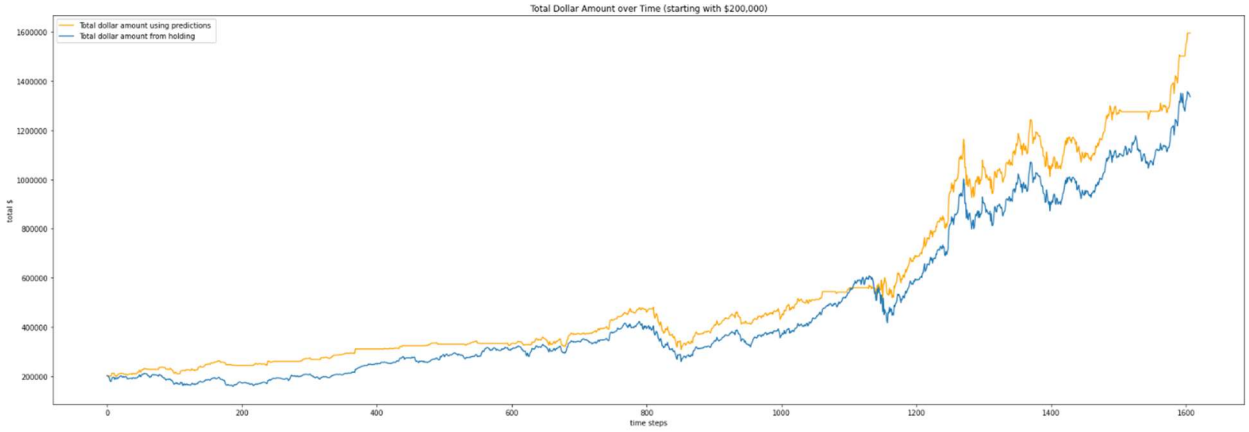
Appendix 3.1: Losses for each batch computed with GRU model (learning rate: 0.0005, hidden size: 10, number of layers: 2)



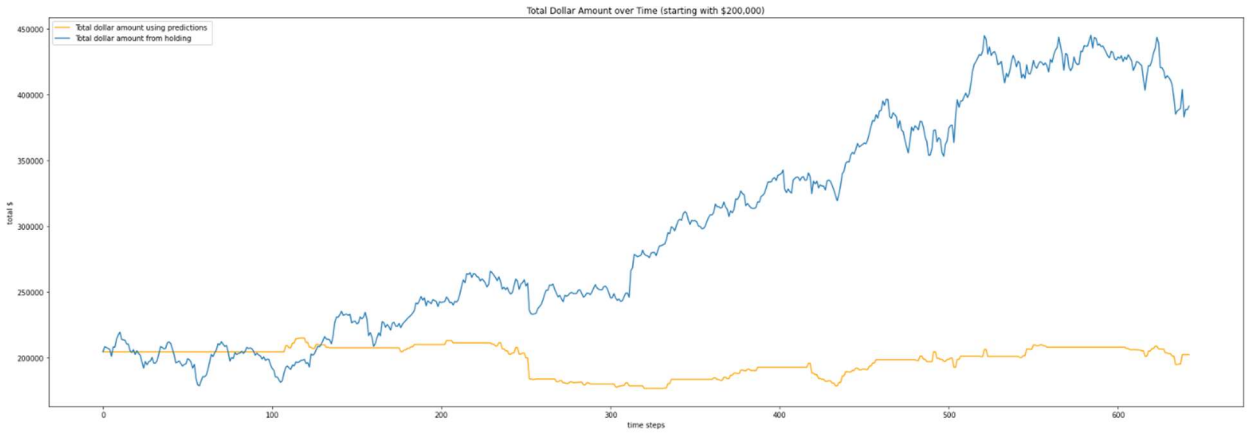
Appendix 3.2: Training %gain predictions vs actual values for GRU model (learning rate: 0.0005, hidden size: 10, number of layers: 2)



Appendix 3.3: Testing %gain predictions vs actual values for GRU model (learning rate: 0.0005, hidden size: 10, number of layers: 2)



Appendix 3.4: Total Dollar Amount over Time for GRU model (learning rate: 0.0005, hidden size: 10, number of layers: 2)



Appendix 3.5: Total Dollar Amount over Time on Validation Set for GRU model (learning rate: 0.0005, hidden size: 10, number of layers: 2)